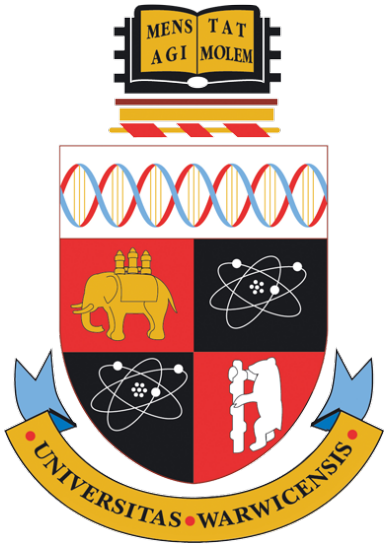
WARWICK
THE UNIVERSITY OF WARWICK



On the Efficiency of Finding and Using Tabular Data Summaries: Scalability, Accuracy, and Hardness

by

Charlie Dickens

Thesis

Submitted to the University of Warwick

in partial fulfilment of the requirements

for admission to the degree of

Doctor of Philosophy

Department of Computer Science

June 2021

Contents

List of Tables **iv**

List of Figures **v**

Acknowledgments **ix**

Declarations **xi**

Abstract **xii**

Acronyms **xiii**

Chapter 1 Introduction **1**

1.1 Contributions 3

Chapter 2 Background **5**

2.1 Models of Computation 6

2.1.1 The Classical Streaming Model 7

2.1.2 Classical Streaming Problems and Solutions 7

2.1.3 Projected Summary Model 9

2.1.4 Matrix Streams 11

2.2 Linear Algebra Background 15

2.2.1 Oblivious ℓ_2 Subspace Embeddings 16

2.2.2 Frequent Directions: Algorithm and Properties 18

2.2.3 ℓ_p Subspace Embeddings 20

2.2.4 Linear Algebra Problems 21

2.3 Optimisation Background 23

2.3.1 Direct Solver 23

2.3.2 Iterative Methods 24

2.4 Communication Complexity 28

2.4.1 Outlining Lower Bound Arguments 29

Chapter 3 Projected Frequency Estimation **31**

3.1 Introduction 34

3.2 Preliminaries and Definitions 35
3.2.1 Problem Definitions 36
3.2.2 Related Work 38
3.3 Contributions 39
3.3.1 Summary of Results 39
3.3.2 Coding Theory Definitions 41
3.3.3 Overview of Lower Bound Constructions 42
3.4 Lower Bounds for F_0 46
3.5 ℓ_p-Frequency Based Problems 49
3.5.1 ℓ_p Frequency Estimation 49
3.5.2 ℓ_p Heavy Hitters Lower Bound 50
3.5.3 F_p Estimation 53
3.5.4 ℓ_p-Sampling 55
3.6 Projected Frequency Estimation via Set Rounding 57
3.6.1 From α-nets to Projections 60
3.7 Concluding Remarks 65

Chapter 4 Streaming Deterministic Summaries in ℓ_p Norms 66
4.1 Introduction 66
4.1.1 Background 66
4.1.2 Summary of Contributions 69
4.1.3 Comparison to Related Work 69
4.2 Preliminaries and Notation 70
4.2.1 Convex Optimisation 72
4.3 Obtaining & Applying ℓ_p Subspace Summaries 72
4.3.1 Relative Error ℓ_p Subspace Embeddings 72
4.3.2 Application: ℓ_p Regression 76
4.3.3 Application: ℓ_1-Low Rank Approximation 78
4.4 Leverage Score Summaries in ℓ_p-Norm 82
4.4.1 Relating Local and Global Leverage Scores 84
4.5 Application: ℓ_∞-Regression 90
4.6 Experiments 96
4.7 Discussion 102

Chapter 5 Iterative Sketching for Least Squares Problems 104
5.1 Introduction 105
5.1.1 Constructing ℓ_2 Subspace Embeddings 106
5.1.2 The Sketch-and-Solve Model 109
5.1.3 Iterative Sketching Framework 112
5.1.4 Comparison of Sketch-and-solve and Iterative Sketching Frameworks 114

5.2 CountSketch in the Sketch-and-Solve Model 115
5.2.1 Structural Properties of CountSketch 116
5.2.2 Spectral Properties of CountSketch 117
5.3 CountSketch in the IHS model 119
5.4 Experiments 123
5.4.1 Synthetic Experiments 124
5.4.2 Real Dataset and Error-Time Performance 129
5.5 Conclusion 135

Chapter 6 Sketched Ridge Regression **137**
6.1 Introduction 138
6.1.1 Related Work: Ridge Regression with Random Sketches 139
6.1.2 Related Work: FD and Ridge Regression 141
6.1.3 Contributions 142
6.2 Structural Properties of $\hat{\mathbf{H}}$ 144
6.2.1 Exploiting the Löwner Order for $\hat{\mathbf{H}}$ 145
6.3 Iterative Methods using Frequent Directions for Ridge Regression 146
6.4 Improving Convergence with Robust Frequent Directions . . . 150
6.5 Time and Space analysis of Algorithm 9 151
6.6 Synthetic Data: Error & Sketch Size 154
6.6.1 Experimental Setup 154
6.6.2 $R = 0.1d$ 155
6.6.3 $R = 0.5d$ 156
6.6.4 Conclusions 156
6.7 Real Data Experiments 157
6.7.1 Experimental Setup 157
6.7.2 Experiment 1: Sketch Build Time. Figures 6.2 and 6.3 . 160
6.7.3 Experiment 2: Update Time. Figures 6.4 and 6.5 161
6.7.4 Experiment 3: Error vs iterations and time. Figures 6.6 and 6.7 162
6.7.5 Summary 164
6.8 Conclusion 165

Chapter 7 Discussion and Conclusion **168**
7.1 Results Summary 168
7.2 Future Work 170

List of Tables

2.1 Comparison of two direct solver approaches. 24

2.2 Comparison of the related iterative methods for ordinary least squares on full-rank data. The parameter m should be used to control the space and time usage per iteration and should be thought of as a tunable parameter that determines how well the Hessian is approximated. Consequently, different values of m will have an impact on how many iterations must be completed, represented by the $O_m(\cdot)$ notation. The role of m and the number of iterations will become clear in Chapter 5. 28

3.1 Comparison of parameter settings for projected F_0 lower bounds. Theorem 3.4.1 uses $\mathcal{C} = \mathcal{B}(d, k)$, corollaries use $\mathcal{C} = \mathcal{B}(d, d/2)$. Alice's set $T \subset \mathcal{C}$ so is no larger than 2^d, so we can always upper bound the size of the instance required for the lower bounds. . 48

5.1 Time and space costs to obtain oblivious subspace embeddings. All sketches except the CountSketch achieve the optimal embedding dimension of $O(d\varepsilon^{-2}\log(d/\delta))$. 107

5.2 Recap of the different weights used for measuring errors in regression. 123

6.1 Relative error ($\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 / \|\mathbf{x}^\star\|_2$) comparison of iFDRR after 5 iterations on synthetic data with effective rank $R = 0.1d$ using Frequent Directions (FD) and Robust Frequent Directions (RFD). The error when using FD is consistently larger by a factor of roughly 7 to 23. The parameter a is used for regularisation $\gamma_a = 2^a \|\mathbf{A}\|_F^2 / m$. 156

6.2 Relative error ($\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 / \|\mathbf{x}^\star\|_2$) comparison of iFDRR after 5 iterations on synthetic data with effective rank $R = 0.5d$ using Frequent Directions (FD) and Robust Frequent Directions (RFD). The error when using FD is consistently larger by a factor of roughly $1.5 \sim 1.6$. The parameter a is used for regularisation $\gamma_a = 2^a \|\mathbf{A}\|_F^2 / m$. 157

List of Figures

2.1 An example of the distributed summary model in a Merge − Reduce tree over blocks $\mathbf{B}_i$ of $\mathbf{A} \in \mathbb{R}^{n \times d}$. This computation tree represents 4 parties communicating with a coordinator in pairs. It could be implemented in a single streaming fashion over all 4 parties by computing the left subtree beneath $\mathbf{E}_1$ first, the right subtree beneath $\mathbf{E}_2$ second, followed by a final round of Merge − Reduce. 13

2.2 An example of the row arrival model in the Merge − Reduce language of Definition 2.1.4. A single participant reads the first block $\mathbf{B}_1$ of input matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ and performs Reduce($\mathbf{B}_1$) to obtain a small summary $\mathbf{C}_1$. Then the next block of input is read in by the operation Merge($\mathbf{C}_1, \mathbf{B}_2$) until the space budget is used and then Reduce(Merge($\mathbf{C}_1, \mathbf{B}_2$)) is executed, returning a small summary $\mathbf{E}_1$. This is repeated over all blocks of the input $\mathbf{A}$ until a single small summary $\mathbf{T}$ is returned. The error in using $\mathbf{T}$ is controlled through properties of the Reduce($\cdot$) operations. 14

3.1 Alice holds a subset T of the code $\mathcal{C} = \{\mathbf{w}_1, \ldots, \mathbf{w}_6\}$, using that to populate her membership vector $\mathbf{a}$. Bob queries algorithm $\mathcal{A}$ with his test vector $\mathbf{y}$ and a column set S. The algorithm returns `estimate` which Bob compares to some threshold `thld`. If `estimate` $\geq$ `thld` then Bob knows $\mathbf{y} \in T$ and can return 1 as $\mathbf{a}_{e(\mathbf{w})} = 1$ for all $\mathbf{w} \in T$. Otherwise, Bob reports 0 as Alice's vector $\mathbf{a}_{e(\mathbf{w})} = 0$ for all $\mathbf{w} \in \mathcal{C} \setminus T$. 43

3.2 Alice maps her word $\mathbf{w} \in T$ to a larger set of *child words* $\mathsf{star}^Q(\mathbf{w})$. For a code $\mathcal{C}$ with weight k, the number of words in $\mathsf{star}^Q(\mathbf{w})$ is Q^k. Alice will concatenate $\mathsf{star}^Q(\mathbf{w})$ for every $\mathbf{w} \in T$ to generate the input data $\mathbf{A}$. 44

3.3 All subsets of [6] except those of size 3 (indicated by the dashed box) are included in the 1/6 net $\mathcal{N}$. The index of the box indicates the size of subsets that are stored in that level with the ellipses marking that *all* subsets of that size are stored in the net. For example, both subsets $\{1, 2, 3, 4\}$ and $\{2, 3, 5, 6\}$ are included at the size 4 level of the net. However, any subset of [6] with size 3 is not included in the net. 58
3.4 Binary entropy $H(t)$ 59
3.5 Behaviour of an α-net $\mathcal{N}$ 59
3.6 Space-approximation tradeoff for $d = 20$ as α is varied from 0 to 1/2. Relative space is $2^{H(1/2-\alpha)d}/2^d$ is the fraction of total subsets stored and Approximation Factor is $2^{\alpha d}$. As α approaches 0, the net stores increasingly more subsets from $\mathcal{P}([d])$ so the relative space approaches 1 which also explains why the approximation factor is smaller. When α approaches 1/2, many fewer subsets from $\mathcal{P}([d])$ are stored so the relative space is small and approaches 0; as fewer subsets are stored, the error induced by set rounding will be higher, as illustrated. . . 64

4.1 Error vs Space Constraint, which is the budget or largest number of rows that can be stored, not the summary size. Total input size is $5,000,000 \times 11$ for U.S. Census Data and approximately $50,000 \times 90$ for YearPredictionsMSD. 98
4.2 Maximum Summary Size vs Space Constraint 99
4.3 Update time for local basis 100
4.4 Query time to solve optimisation 101
4.5 Total time cost 101

5.1 Error to model weights $\mathbf{x}^\dagger$ under an optimal solver, classical sketch, or IHS. 126
5.2 IHS optimal coefficient error between the sketched weights $\hat{\mathbf{x}}$ and the optimal weights $\mathbf{x}^\star$ over iterations. 127
5.3 IHS error to model weights vs iterations. The model error from the optimal weights given data $\mathbf{A}$ is denoted: Optimal $\approx \left\|\mathbf{x}^\star - \mathbf{x}^\dagger\right\|_{\mathbf{A}}$. 128
5.4 Error profile for IHS on California Housing dataset 131
5.5 Error profile for IHS on California Housing dataset for density 12.5% (top), 25% (middle), 50% (bottom). 132

5.6 Mean sketch time on different density data for sparse sketches (left) and mean sketch time over all 4 densities for dense sketches (right). Note the different y axis scale in each pane. The black dashed line in the right-hand pane is added for comparison. It plots the *largest* average sparse sketch time, namely SJLT on density of 75%. 133
5.7 Mean total sketch time (thin lines) and total update/gradient step time (thick dashed lines) marked on different density data for sparse sketches (left) and mean sketch time over all 4 densities for dense sketches (right). Note the different y axis scale in each pane. 135
5.8 Test error approximation factor when using IHS 136

6.1 Relative error $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 / \|\mathbf{x}^\star\|_2$ plotted on a log scale against the number of iterations for various choices of $\gamma = 2^a \|\mathbf{A}\|_F^2/m$ with $a \in \{-1, 0, 1, 2\}$. Left hand plot has $R = 0.1d$ and right hand plot has $R = 0.5d$. Frequent Directions is plotted in dashed lines and Robust Frequent Directions is in solid lines. Note the different scales on left and right hand plots. 154
6.2 *Total* sketch build time: CoverType. Black dashes indicate multiple rounds of sketching, rather than just one. 160
6.3 *Total* sketch build time: w8a. Black dashes indicate multiple rounds of sketching, rather than just one. 161
6.4 Update time: CoverType. Note the different y axis scales. . . . 162
6.5 Update time: w8a. Note the different y axis scales. 166
6.6 Error profile over time for CoverType 166
6.7 Error profile over time for w8a 167

List of Algorithms

1 Frequent Directions (FD) and Robust Frequent Directions (RFD). 19

2 Projected frequency estimation by query rounding 61

3 Streaming Deterministic ℓ_p Subspace Embedding 74
4 Derandomised ℓ_1 low rank approximation of [SWZ17] 80
5 Deterministic ℓ_1 low rank approximation on a stream 81
6 Deterministically finding rows of high leverage on a stream . . . 89
7 Deterministic Approximate ℓ_∞ Regression 91

8 Iterative Hessian Sketch (IHS) 120

9 Iterative Frequent Directions Ridge Regression 146

Acknowledgments

First and foremost, I would like to thank my supervisor, Graham Cormode. Graham's supervision throughout my years at Warwick has always been helpful and reassuring. Every opportunity I've sought, every change of research interest I've had, every mistake I've made, Graham has been there to offer his support and advice. I certainly would not be in a position to submit this thesis were it not for Graham's guidance and for that I am extremely grateful.

Secondly, I'd like to thank David Woodruff. David has been a fruitful collaborator in some of the works that constitute this thesis. I have learnt so much from conversations with David and I'm grateful for this experience. Thanks also to Jeff Phillips & Theo Damoulas for taking the time to provide helpful feedback in improving this thesis.

Over the course of my time at Warwick I have had the privilege of being able to move around quite a lot. Along the way, I have met so many people and there are too many to thank individually (for which I apologise profusely!). However, special thanks are in order for the following people or organisations:

- The Department of Computer Science at Warwick has been a great environment for me to learn. I have always felt supported and encouraged to take any opportunity that helped me grow as a researcher. Particular thanks to Sharon Howard for her patience in resolving issues caused by my forgetting how things work after various stints away, alongside helping me arrange those trips in the first place!

- The Alan Turing Institute for supporting my Enrichment Year and providing a great atmosphere (and a great place to watch the 2018 FIFA World Cup). The fourth floor lab with Nikolas Kuhlen, Prateek Gupta, Nathan Cunningham & Julien Vaes was great fun.

- The Simons Institute for accommodating me during the *Foundations of Data Science* program; the program organisers for providing an inspiring and stimulating environment; and all of the people that I met there and who made me feel welcome.

- Tom Diethe for hosting me as an intern and the entire research lab at Amazon Cambridge; it was great to study something totally new. Also, particular thanks go to Pablo Garcia Moreno for (organising the best table football league and) leading us down the enjoyable but challenging Bayesian nonparametrics rabbit hole, Eric Meissner and Shuai Tang for being great fun, in Cambridge and beyond.

- Lee Rhodes, Jon Malkin, and Alex Saydakov for another great intern experience, albeit in much different circumstances. Thanks to you all, Joonsuk Bae, and the wider Verizon Media team for pushing forward with my internship offer under extremely challenging and uncertain circumstances. I can't wait to join you guys soon!

I would like to thank my Mom* and Dad for always supporting me. Your belief in education, but importantly your belief in me, is what started me on this path. Although you may not understand the work I do (don't worry, sometimes I don't either!), this journey would not have been possible without your constant encouragement to keep making the most of new opportunities. You have given me the confidence to find new challenges and that has put me in the position I am today. I am grateful for the support of my sister, Ellie, who always provides a hilarious antidote to the difficulties of work. Thanks also to my Nan for her perpetual encouragement and reminding me (deliberate or otherwise) always to be happy.

Finally, a very special thanks to my wonderful partner Lola Ogunsanya for all of her love and support. Being in different cities, countries, and timezones, coupled with near-constant moving around has made the past five years difficult at times. You have been there every step of the way. You have consoled me during the lows and celebrated with me during the highs: this thesis would not exist without you.

*This is how we say/spell it in *The Black Country*, not American English!

Declarations

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree. The work presented was carried out by the author. Parts of this thesis have been previously published by the author in the following:

Chapter 3

[CDW21] Graham Cormode, Charlie Dickens, and David P. Woodruff. Subspace exploration: Bounds on projected frequency estimation. In *ACM Principles of Database Systems (PODS)*. ACM, 2021

Chapter 4

[CDW18] Graham Cormode, Charlie Dickens, and David P. Woodruff. Leveraging well-conditioned bases: Streaming and distributed summaries in Minkowski p-norms. In *International Conference on Machine Learning*, 2018

Chapter 5

[CD19] Graham Cormode and Charlie Dickens. Iterative hessian sketch in input sparsity time. In *NeurIPS Workshop: Beyond First-Order Optimization Methods in Machine Learning*, 2019

Chapter 6

The results of this chapter have not been previously published.

Abstract

Tabular data is ubiquitous in modern computer science. However, the size of these tables can be large so computing statistics over them is inefficient in both time and space. This thesis is concerned with finding and using small summaries of large tables for scalable and accurate approximation of the data's properties; or showing such a summary is hard to obtain in small space. This perspective yields the following results:

- We introduce projected frequency analysis over an $n \times d$ binary table. If the query columns are revealed *after* observing the data, then we show that space exponential in d is required for constant-factor approximation to statistics such as the number of distinct elements on columns S. We present algorithms that use smaller space than a brute-force approach, while tolerating some super constant error for the frequency estimation.

- We find small-space deterministic summaries for a variety of linear algebraic problems in all p-norms for $p \geq 1$. These include finding rows of high leverage, subspace embedding, regression, and low rank approximation.

- We implement and compare various summary techniques for efficient training of large-scale regression models. We show that a sparse random projection can lead to fast model training despite suboptimal theoretical guarantees than dense competitors. For ridge regression we show that a deterministic summary can reduce the number of gradient steps needed to train the model compared to random projections.

We demonstrate the practicality of our approaches through various experiments by showing that small space summaries can lead to close to optimal solutions.

Acronyms

COO COOrdinate List.

FD Frequent Directions.

FDRR Frequent Directions Ridge Regression.

iFDRR Iterative Frequent Directions Ridge Regression.

IHS Iterative Hessian Sketch.

ML Machine Learning.

OLS Ordinary Least Squares.

RFD Robust Frequent Directions.

RFDRR Robust Frequent Directions Ridge Regression.

RLA Randomised Linear Algebra.

RR Ridge Regression.

SJLT Sparse Johnson-Lindenstrauss Transform.

SRHT Subsampled Randomised Hadamard Transform.

SVD Singular Value Decomposition.

Chapter 1

Introduction

Tabular data pervades modern data analysis and machine learning (ML). These data tables are perceived as having rows corresponding to different "observations" and columns that take values over a set of measurable "features" or "attributes". This abstract starting point allows us to be domain-independent. For example, the rows of the table may be time-ordered with each column representing time series that could be used for, say, monitoring financial or health signals [BJRL15, Coq21, SRD+18]. Alternatively, the table may represent a sequence of unrelated observations, for instance, each row being an image with features representing the colour of a pixel [LMB+14, WBW+11].

By exploiting large and rich datasets, machine learning has led to insightful discoveries such as AlphaFold, [JEP+20] and technologies such as ride-hailing apps [UberMLEng17], media recommendations [KBV09], and email filtering [CL98, GJC20]. These works focus on achieving *more* with data. Our focus is related but complementary. Large datasets are often highly redundant and this causes an algorithmic headache [UT19]. The cost in time or space of solving a given task exactly could scale superlinearly with respect to the input sizes, either the number of observations n, or the dimensionality of the feature space d. Common pitfalls might be that the algorithm for solving a task simply takes too long (i.e. it is polynomial time or worse in either n or d) [BJW19], or requires storing almost the entire input. However, even simple queries such as counting the number of distinct values in a dataset require linear space to evaluate [AMS99a] which is problematic when the number of samples is huge, as may be common in modern data. Both of these issues render exact analyses unscalable. This thesis is concerned with mitigating this problem: *can we do almost as well as solving the original problem, but without doing so? Is it possible to create succinct "summaries" of the dataset which permit more efficient solutions than solving the original instance?*

Encoding the data in such large tables provides a common starting point for a machine learning pipeline. These tables might be arbitrary arrays, or

matrices when we are studying linear algebraic computations. Amongst many others, key steps in this pipeline are data preparation or analysis followed by "model training". The former seeks to understand basic patterns or properties of the data, such as the number of distinct values in a column, the median value, or various other frequency statistics. The latter is concerned with using this knowledge of the data to build a succinct description of the dataset which can be used for predicting output values from future input values.

Corporations on the scale of Facebook and Walmart can regularly yield customer behaviour data on the scale of terabytes to petabytes per day [SKIW17]. Due to this scale, even seemingly simple tasks regarding data management can become difficult. For example, the data tables may be so large that they can only be read once or may never be held completely in one single location. We will consider abstractions of these concerns in our computation models and algorithm design.

Our focus in this thesis is on both *finding* and *using* small summaries of large tabular data. We should think of data summarisation as a two dimensional axis over a "computational cost" independent variable and some dependent variable, "utility", measuring how well a summary performs for a given task. One should think of the computational cost as an adjustable parameter depending on our constraints, such as available compute time, space consumption, the communication cost of moving data between distributed locations, or the number of times one can view data. The goal is to minimise the overall computation. First, we find a small description of the data that is cheap to obtain. Next, we use this summary to perform a tractable approximate analysis. The utility of the summary should come from a guarantee that it is efficient to compute and the approximate analysis is close to that had no summarisation been performed.

The starting point of all aspects of this thesis is to find a summary for a given computational task over which we can prove guarantees on its utility. Some pertinent high-level questions arise from this perspective that form the central body of this thesis. Although the importance placed on each individual question will change depending on the specific task at hand, they provide a useful lens through which to view the entirety of this work. A central question is always how to find such a summary *efficiently*, under notions of efficiency that may change depending on the task. The next question is what guarantees can we give upon using a summary, is it *accurate* (under differing notions of accuracy or error)? We will strive to be as frugal as possible in expending our computational effort; yet there must be fundamental limits on the computational resources we need before our guarantees break down.[1] Understanding this limit is what is known as the *hardness* of certain problems

[1] Otherwise, trivial or empty summaries would always suffice, which clearly is not true!

to approximate using small summaries. Of course, not all problems are created equal, so our task is three fold:

1 **Scalability.** To find efficiently computable summaries of tabular data.

2 **Accuracy.** To provide guarantees on the fidelity of a summary compared to the original input, meanwhile;

3 **Hardness.** To acknowledge the inherent computational difficulty of different problems in order to understand the overall scalability of our solution, in comparison to established methods.

1.1 Contributions

We outline the high-level contributions in each chapter and how they relate to the central theme of the thesis. More detailed statements on the contributions can be found in each chapter. The thesis follows the outline given below.

Chapter 3. Projected Frequency Estimation

We give the first analysis of frequency estimation problems in the novel *projected summary model* to be defined in Section 2.1.3. The input is an array $\mathbf{A} \in \{0,1\}^{n \times d}$ and a *column query* $S \subseteq [d]$ that is revealed *after* observing the data. We study problems such as counting the number of distinct elements on the columns S of $\mathbf{A}$, termed a "projection". We also study other problems related to projected frequency analysis, such as row sampling and heavy hitters. The majority of our results show that these problems are hard and require space exponential in d to achieve even a constant factor approximation.

For upper bounds, we show that a small "net" $\mathcal{N}$ of subsets from the power set $\mathcal{P}([d])$ can be chosen to avoid enumerating a sketch for every set in $\mathcal{P}([d])$. The idea is that every $U \in \mathcal{N}$ has an associated sketch for the given frequency problem. On receiving the query columns S, we "round" S to a neighbour $S' \in \mathcal{N}$ and return the estimate from the sketch for S'. Through the notion of closeness, measured under symmetric difference, we can control the additional deterministic error from returning the approximation on S' rather than S. Whilst $|\mathcal{N}|$ may still be exponential in d, we show how to control the size of the net while characterising the error incurred. Specifically, if $N = 2^d$ and $0 < c < 1$, then we achieve a N^c approximation in $N^{O(c)}$ space.[2]

Chapter 4. Deterministic ℓ_p Summaries

We study matrix estimation and fundamental linear algebraic problems such as regression and low-rank approximation in general ℓ_p norm. The tool that we

[2]The constants in $O(\cdot)$ are small, problem dependent, and explicitly known.

use is an ℓ_p well-conditioned basis. However, finding such a basis costs $O(nd^5)$ time so is not a scalable approach. Thus, we give a streaming algorithm that computes small parts of such a basis based on local views of data. Our methods are the first fully deterministic algorithms to approximate these problems using small space for all $p \geq 1$. The data summaries that we use have size $\text{poly}(d) \times d$, independent of n, and are evaluated over streaming data. We apply these summaries to obtain provable guarantees on ℓ_p regression and low rank approximation. Finally, we give a hardness result for ℓ_∞ regression showing that our additive error guarantee cannot be improved to relative error without substantially increasing the space usage.

Chapter 5. Iterative Sketching for Least Squares Problems

We study scalable variants of iterative algorithms for training regression models using sketches. Much prior work has shown only coarse approximation of the optimal regression weights whereas we seek high accuracy. Our results show that sparse random projections can be used in the "Iterative Hessian Sketch" framework. We show that using sparse sketches can yield a speedup roughly $2 \sim 20$x faster than the best dense competitors.

Chapter 6. Sketched Ridge Regression

For the special case of ridge regression we adapt the iterative sketch method and use a single Frequent Directions sketch. In this setting, although Frequent Directions sketch takes longer to obtain, it uses a third of the total iterations used by both sparse and dense random projections. Overall, this yields total running time similar to the fastest sparse methods but terminates in fewer steps. This could be a huge benefit as in practice revisiting the data could bear a substantial cost.

Chapter 2

Background

Modern data can be both large, having in excess of millions of rows, and high-dimensional, with hundreds or thousands of columns. Often, datasets $\mathbf{A} \in \mathbb{R}^{n \times d}$ in tabular form are used to model interactions over the samples (rows) according to certain features (columns). In order to gain insights from such data, practitioners need to apply data analysis techniques to these tables at scale. We study two classes of problems: frequency estimation over column restrictions of $\mathbf{A}$ and linear algebraic operations on $\mathbf{A}$. When studying frequency analysis, we will use the mathematical notation of matrices, but refer to $\mathbf{A}$ as an array. Meanwhile, for the remainder of the thesis, we will use the language of linear algebra. Indeed, for modern machine learning, linear algebra is the workhorse underpinning crucial computations so it is critical to understand how matrix operations can be performed at scale.

Despite the necessity of linear algebra primitives, their cost is often considerable, sometimes superlinear in one of the dimensions of the input data. Common examples are inverting an $n \times n$ matrix in $O(n^3)$ time or performing an SVD in $O(nd^2)$ time (if $n > d$). These complexities are not acceptable in modern applications when both n and d can be large, causing a major bottleneck. Consequently, the task of finding scalable approximations to these fundamental primitives has generated substantial research interest [Lib13, HMT11, Woo14b].

Our interest is not solely on linear algebraic problems. Even at scale practitioners would like to answer "simple" queries of their input data. For example, if $\mathbf{A} \in \{0,1\}^{n \times d}$ represents a survey of n people over d yes/no (1/0) questions, then simple queries such as *"how many people answered yes to questions* $1, 2, \ldots, k$ *(for some* $k \leq d$ *but reasonably large)?"* are of practical interest. Prior work has generally focused on the single-column setting over a large domain. With the growing scale of data, it is pertinent to study higher dimensional queries. For these types of problems, we will introduce a novel model of computation and describe the class of problems under consideration. Unlike the linear algebra problems, these are more simple to internalise as

they are roughly related to estimating frequencies observed under column projections. Simply put, column projections are restrictions of the input table (or questions, in the above example) to fewer columns than are present in the entire set.

Chapter Outline

This chapter defines many of the concepts that are used throughout the thesis. Each subsection should be regarded as an almost self-contained introduction to the technical material with more sophisticated discussions deferred to the necessary chapters.

- In Section 2.1 we define the *computation models* in which our results apply. Section 2.1.1 defines the 'standard' streaming model which is a fundamental starting point. Following this, some classical problem definitions for the streaming model are given in Section 2.1.2. Sections 2.1.3 and 2.1.4 are independent of each other but both build upon 2.1.1 and 2.1.2. Section 2.1.3 defines the model used for high-dimensional data perceived as an array, rather than a stream of items. Section 2.1.4 defines two single-pass models for estimating linear algebraic functions on streams defined over matrices and one multi-pass model that is implicitly and regularly used in machine learning.
- Section 2.2 gives an overview of the linear algebra material we need. The starting point is to define different notions of matrix estimation that we study in the Euclidean norm, first under random projections (Section 2.2.1) and then with deterministic sketching (Section 2.2.2). In Section 2.2.3 we extend the notion of matrix estimation introduced for ℓ_2 to arbitrary $\ell_{p>0}$. We define the specific problems that will be considered using these tools in Section 2.2.4.
- Section 2.3 presents the optimisation background. These ideas are the starting points that our iterative regression algorithms build upon.
- The final part of this chapter is an overview of the tools we use from communication complexity. Section 2.4 presents problem definitions and overviews the type of arguments we will make to show space lower bounds. It is independent of the linear algebra work outlined in the preceding sections.

2.1 Models of Computation

Throughout this thesis we will encounter various models of computation that address different problems. These can be seen as variants of the standard

streaming model introduced presently. The classical model of computation that is used for comparison is the *RAM model of computation*. In this setting, all data is available and can be accessed in constant time. Simple operations such as *scalar* addition and multiplication require $O(1)$ time [CLRS09]. Many *fast* algorithms for approximating linear algebraic properties implicitly operate in this model [LMP13, CP15, CLM$^+$15]. This is due to the assumption that small amounts of the data can be accessed quickly and the data is available in its entirety at any given time. Although many algorithms in this model run in "input-sparsity" time, they may require multiple views of the data.

The RAM model is a natural model for computation. However, the assumptions that all data is available and easily accessible are not reasonable for truly large-scale data. In fact, modern data may be so large that only a small amount of it (or small space function of the data) can be stored. This is the motivation behind the *streaming model* proposed by [AMS99b]. The vital contribution of [AMS99a] was to initiate the study function approximation of data streams using only a small amount of working space. It is this perspective we extend for matrix computation.

2.1.1 The Classical Streaming Model

The data arrives sequentially as a $\mathsf{stream} = (x_1, x_2, \ldots, x_n)$ and on observing the final item of the stream, the algorithm must return an answer to a given problem [CY20]. The items x_i may be numeric or any other type that can be appropriately mapped to a numeric universe. We refer to both as *univariate streams* as mathematically they can be mapped to a stream of scalars. The algorithm can keep a small space function, summary or subset of the data, but it is not possible to revisit particular items. If revisiting the data is necessary, then the entire dataset must be traversed once again.

2.1.2 Classical Streaming Problems and Solutions

In this relatively new model of computation, there has been great interest in approximating even the most basic functions using small space. The notion of approximation that we are typically interested in is as follows:

Definition 2.1.1. *Let $a, b > 0$ and let $x^\star \in \mathbb{R}$ be some quantity. An (a, b)-approximation to $x^\star$ is some $\hat{x} \in \mathbb{R}$ such that: $ax^\star \leq \hat{x} \leq bx^\star$. Often we may write $\hat{x} \in [ax^\star, bx^\star]$ to denote this ordering.*

For the purposes of this thesis, Definition 2.1.1 may hold deterministically with certainty or it may hold with some probability of success, say at least $1 - \delta$ for $\delta \in [0, 1]$. A gold standard in randomised approximation is to obtain $a = 1 - \varepsilon$ and $b = 1 + \varepsilon$, referred to as a $1 \pm \varepsilon$ approximation, in space

polynomial in $1/\varepsilon$. A "diamond standard" can be regarded as achieving $1 \pm \varepsilon$ approximation with space polynomial on $1/\varepsilon$ and $\log 1/\delta$ (rather than, say, $1/\delta$). If the running time is also polynomial in the size of the input, then this would be called a *Fully Randomised Approximation Scheme* and further details can be found in [MR95, Chapter 11]. Indeed, we will later see such distinctions in the context of subspace embeddings.

Studying the space complexity of approximate frequency analysis over data streams was initiated in [AMS99b]. In particular, they showed that certain *Frequency Moments* that can be approximated in logarithmic space. This class of problems can be characterised through the following notation, suppose that $\mathsf{stream} = (x_1, x_2, \ldots, x_n)$ with each $x_j \in [N]$. The number of times item i is seen is denoted $f_i = |\{j : \mathsf{stream}_j = i\}|$, known as its frequency. The *frequency vector* is $f = (f_i)_{i=1}^N$. For $k > 0$ we define the k^{th} *Frequency Moment* as

$$F_k = \sum_{i=1}^{N} f_i^k.$$

If $k = 0$ then let $\mathbf{1}(u)$ be the indicator function for u so that,

$$\begin{aligned} F_0 &= \sum_{i=1}^{N} \mathbf{1}(f_i \neq 0) \\ &= |\{i : f_i \neq 0\}| \end{aligned}$$

is simply the number of distinct elements present in frequency vector f over the stream. Under this abstraction, there are many natural data analysis queries that can be asked of a large stream. For example, F_0 counts how many distinct items are present in stream, F_2 is *Gini's index of homegeneity*, a commonly-used economic metric, and for larger p, F_p can measure the skewness of a data stream. There are many other problems, some of which we will later study, such as heavy hitters, which asks us to return all items which contribute more than some fraction of $\|f\|_p$. Another is ℓ_p-sampling, which also asks us to approximately sample with respect to the estimated item frequencies.

Example 2.1.1. *Suppose* $\mathsf{stream} = (x, x, x, y, x, y, x, z)$. *Then* $f_x = 5, f_y = 2, f_z = 1$. *Hence,* $F_0(\mathsf{stream}) = 3$ *as there are three distinct items and* $\|f(\mathsf{stream})\|_2^2 = 30$. *If we fix* $p = 1$ *then the total frequency is* $\|f\|_1 = 8$ *and thus the* ℓ_1*-sampling problem would be to sample* x, y, z*) with a distribution approximately* $(5/8, 2/8, 1/8)$. *We also see that* x *is an* ℓ_1*-heavy hitter at a threshold of at most* $5/8$ *as it accounts for* $5/8$ *of the total mass.*

The "classical" streaming model we have described captures both impressive theoretical and useful practical results, we outline only a few here:

- F_0: [KNW10] provide an space optimal algorithm for estimating F_0 up to $1 \pm \varepsilon$ in $O(\varepsilon^{-2} + \log N)$ space. An alternative algorithm, the *HyperLogLog* has also been deployed for its practicality in estimating F_0 [HNH13].

- For heavy hitters, there are various approaches. Some common methods are the CountSketch of [CCFC02] which can find ε heavy hitters in ℓ_2 using $O(\varepsilon^{-2} \log N)$ space. This is space-optimal if the stream has deletions as well as insertions and $O(\log N)$ update time. Other works try to achieve similar space but optimise for update time [BCI+17] or improve the space bound if the stream only contains insertions. There are similar approaches, such as the CountMinSketch which can find ε heavy hitters in ℓ_1 using space $O(\varepsilon^{-1} \log N)$, again with $O(\log N)$ update time [CM05]. The CountMinSketch has seen applications in database technology [Hab16] as well as large-scale privacy enhancing technology [Tea17].

- For general frequency moments [Woo04] showed that $\Omega(\varepsilon^{-2})$ space is necessary which matches the $\Omega(\varepsilon^{-2})$ lower bound for F_0 [IW03].

- The situation is a little more complex for ℓ_p sampling. However, the space complexity has been fully characterised and behaves as $O(\log N \log(1/\delta))$ [JW21] closing a line of work established in 2010 [MW10, JST11].

These results only apply in the standard "univariate" streaming setting. As the dimensionality of datasets is typically large, we are also interested in modifications of the streaming model that can adapt to queries over higher dimensions. Also, for machine learning we are interested in streams of data that represent matrices. We now introduce some variants on the standard streaming model.

2.1.3 Projected Summary Model

Before introducing this model we will provide a motivating example: suppose that the matrix $\mathbf{A} \in \{0,1\}^{n \times d}$ represents which users i have purchased item j from an online marketplace. We could say that a "purchasing pattern of user i on item set $S \subseteq [d]$" is the binary string observed in row $\mathbf{A}_i$ on columns S. Specifically, the binary string $(\mathbf{A}_{i,j_1}, \mathbf{A}_{i,j_2}, \ldots, \mathbf{A}_{i,j_{|S|}})$. In this setup, an analyst may wish to answer a question such as, *"how many different purchasing patterns are observed on items $j = 1$ to 20 over all users?"* Since the set of columns $\{1, 2, \ldots, 20\}$ is known in advance, the analyst could encode a binary representation over the first $|S|$ columns into integers from the set $\{0, 1, 2, \ldots 2^{|S|-1}\}$. Following this transformation, the input would now be univariate and thus they could apply a standard streaming algorithm such as those described in Section 2.1.2.

Of course, if the column set S is known in advance, then it is simple to transform to a univariate problem. Hence, this problem becomes of interest when the columns of $\mathbf{A}$ are *not known* until $\mathbf{A}$ has been fully observed. The motivation here is that both n and d are large but a sample of cn rows of length d could be stored for a small constant c. However, d is sufficiently large so that the length 2^d frequency vector, whose entries index all subsets of $[d]$, is too large to store, as is the raw $n \times d$ input. When $\mathbf{A}$ is observed an analyst would like to answer basic statistics of the data which are restricted to certain columns of $\mathbf{A}$ but they are not known until the analyst has already seen the data. This generalisation of the streaming model is called the *Projected Summary Model*. Before defining the model we will provide an example.

Example 2.1.2. *Suppose* $\mathbf{A} \in \{a, b\}^{4\times 3}$ *with column indices* $\{1, 2, 3\}$ *given below. If* $S = \{1, 2\}$*, then we obtain* $\mathbf{A}^S$ *and hence* $f(\mathbf{A}, S) = (2, 1, 1, 0)$*.*

$$\mathbf{A} = \begin{bmatrix} a & a & a \\ b & a & b \\ a & b & b \\ a & a & a \end{bmatrix} \qquad \longrightarrow \qquad \mathbf{A}^S = \begin{bmatrix} a & a \\ b & a \\ a & b \\ a & a \end{bmatrix}$$

The vector $f = f(\mathbf{A}, S)$ *is the frequency vector over which we seek to compute statistical queries such as* $\|f\|_0$*. In this example,* $f_{aa} = 2, f_{ab} = 1, f_{ba} = 1$ *so* $\|f\|_0 = 3$ *(there are three distinct rows in* $\mathbf{A}^S$*), and* $\|f\|_2^2 = 6$*. We use the mapping* $aa \mapsto 00 = 0, ab \mapsto 01 = 1, ba \mapsto 10 = 2, bb \mapsto 11 = 3$ *to materialise* f*. Any other bijection from* $\{a, b\}^2 \mapsto \{0, 1, 2, 3\}$ *would produce* $\tilde{f}$ *isomorphic to* f *with the same frequency statistics.*

Definition 2.1.2 (Projected Summary Model [CDW21])**.** *The data* $\mathbf{A}$ *is received but is too large to fit in memory so can be perceived as a data stream (we make no assumption on the arrival order of items).* ***After*** *observing* $\mathbf{A}$ *a column query* S *is presented which restricts* $\mathbf{A}$ *to only the columns (or projected subspace)* S*, written* $\mathbf{A}^S$*. The frequency vector over* $\mathbf{A}$ *on column set* S *is* $f(\mathbf{A}, S)$*. The problem is to evaluate a function* $\mathcal{F}$ *over* f*, denoted* $\mathcal{F}(f(\mathbf{A}, S))$*. Examples of functions* $\mathcal{F}$ *are the* ℓ_p*-norms* $\mathcal{F}(f(\mathbf{A}, S)) = \|f(\mathbf{A}, S)\|_p$*.*

Remark 2.1.1. *We are tasked with understanding how* $\mathcal{F}(f(\mathbf{A}, S))$ *can be estimated in small space or to show hardness results. For upper bounds, the approach is to design a summary of* $\mathbf{A}$ *during the observation phase which approximates statistics of* $\mathbf{A}^S$*. The approximations are accessed through the vector* $f(\mathbf{A}, S)$*. Note that functions are taken over the underlying vector* $f(\mathbf{A}, S)$ *rather than the raw input data from column projection.*

Typical problems that we will consider are column projected variants of frequency estimation problems as introduced in Section 2.1.2. For example, for

the frequency vector $f(\mathbf{A}, S)$ we may want to approximate the *projected count distinct problem on* $\mathbf{A}^S$, denoted $\mathcal{F}(f(\mathbf{A}, S)) = \|f(\mathbf{A}, S)\|_0$. Similarly, we have analogues of the projected frequency estimation and ℓ_p sampling that will be defined formally in Chapter 3.

2.1.4 Matrix Streams

The previous section studied data presented as a $n \times d$ table which is too large to explicitly compute frequency vectors. Now we switch focus to understand how such a table might be represented on a stream. Understanding how the streaming model can be adapted for large-scale matrix data has been the subject of recent algorithms and machine learning research [CW09, GRB+19]. One common method of storing (sparse) matrices is in the *COOrdinate List (COO)* format which is a list of tuples $(i, j, \mathbf{A}_{ij})$ denoting the row i, column j and value $\mathbf{A}_{ij}$ of the matrix in that location. A further abstraction of the COO format would allow multiple $(i, j, \mathbf{a}_{ij})$ triples, where $\mathbf{a}_{ij}$ now represents an additive 'update' to the value at location i, j rather than just its value. Matrices expressed in such a form are not fully recorded and may even be evolving over time. Examples of the former can be seen in the well-known LIBSVM repository [CL11] that stores datasets for many common machine learning tasks through updates to the nontrivial locations in $\mathbf{A}$.

In essence, matrix streams are representations of an underlying matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ that is not explicitly recorded. Rather, the stream that defines $\mathbf{A}$ is a sequence of rows of $\mathbf{A}$, or a sequence of triples over timesteps t with $(i^{(t)}, j^{(t)}, \mathbf{a}_{ij}{}^{(t)})$. Then the value $\mathbf{A}_{ij} = \sum_t \mathbf{a}_{ij}{}^{(t)}$. Generalisations of these approaches also apply to tensors [TYUC19].

Row Arrival Streaming Model

Although the standard streaming model is natural for matrix computation, it has also been shown to be somewhat restrictive compared to our first alteration. In [GLPW16] the authors state: *"Our paper shows that the row-update model is strictly easier"* than a model allowing updates $(i, j, \mathbf{a}_{ij})$ to $\mathbf{A}$ as described above. Rather than allowing arbitrary updates, if we relax to row-wise updates then [GLPW16] shows that more practical results can be obtained by demonstrating tighter bounds for matrix approximation.[1] More concretely, rather than the stream revealing updates to the entries of $\mathbf{A}$, instead the rows of $\mathbf{A}$ are revealed one-by-one. That is, if $\mathbf{A}_i \in \mathbb{R}^{1 \times d}$ is a row from

[1] Strictly speaking, this claim is made for a slightly different bound, namely regarding low-rank approximation, than the error guarantee we will study for Frequent Directions. To achieve such a guarantee, [CW09] show a tight space bound of $\Theta((n+d)k/\varepsilon)$, while in [GLPW16] demonstrate an algorithm achieving the same guarantee in $O(dk/\varepsilon)$ space. The latter is better by a factor of nk/ε, see Section 6 and Table 5 of [GLPW16] for details.

the matrix $\mathbf{A}$, then $\mathsf{stream} = (\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n)$. The order in which the rows appear may be arbitrary or adversarial. Although we previously motivated matrix streaming through sparse updates, we need not be tied to this model as both sparse and dense data is common in machine learning. Examples of a typical dataset that might be suitably analysed in this model are image datasets which are fed as d dimensional feature vectors to large neural networks [WBW+11, LMB+14].

Definition 2.1.3 (Row Arrival Streaming Model)**.** *[Lib13, CDW18] Let $\mathbf{A} \in \mathbb{R}^{n \times d}$ be the input data which is fed to the algorithm a row at a time: $\mathsf{stream} = (\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n)$. Let $m \ll \max(n, d)$ be the largest number of rows that can be stored. The stored subset is used to compute local statistics which are used to determine which of the stored rows should be kept or discarded. Further rows are then appended and the process is repeated until the full matrix has been read. An approximation to the given problem is then computed by solving on the reduced subset of rows.*

Although we have stated Definition 2.1.3 by assuming the summary is a subset of the rows from $\mathbf{A}$, this need not be the case. Some summary techniques keep small functions of the stored rows. For example, we could keep random linear combinations by hashing an entire row into buckets (using a row arrival variant of [CW13]) or maintain deterministic linear combinations [Lib13] of the rows for the summary, but the idea remains the same.

Distributed Summary Streaming Model

We now introduce a close relative of the row arrival streaming model which is motivated by the modern tendency to store data in a distributed fashion. One can imagine an environment where data is stored across multiple data centres and a central coordinator would like to evaluate some query over the entire dataset. Of course, each of the data centres could transmit their local data (which itself could be large) to the central coordinator. This may entail significant costs in time, space, and communication. Rather than pay this cost, we propose a distributed model whereby each data centre would send only a summary of their local data to the central coordinator. The coordinator can then append all the summaries to approximate their problem, or perform further reduction rounds on the currently held data.

Definition 2.1.4 (Distributed Summary Streaming Model [CDW18])**.** *Let $\gamma \in (0, 1)$ be a small constant and $\mathbf{A} \in \mathbb{R}^{n \times d}$ be the input matrix. Suppose that $\mathbf{A}$ is partitioned into $2L = n^{1-\gamma}$ disjoint blocks $\mathbf{B}_1, \ldots, \mathbf{B}_{2L}$ each containing (at most) n^{γ} rows. These blocks will be stored at the leaves of a computation tree $q_1, q_2, \ldots, q_{2L}$. At the leaf node q_i, the data is reduced by some summary technique $\mathbf{C}_i = \mathrm{Reduce}(\mathbf{B}_i)$. The smaller matrix $\mathbf{C}_i$ is then passed to a parent node*

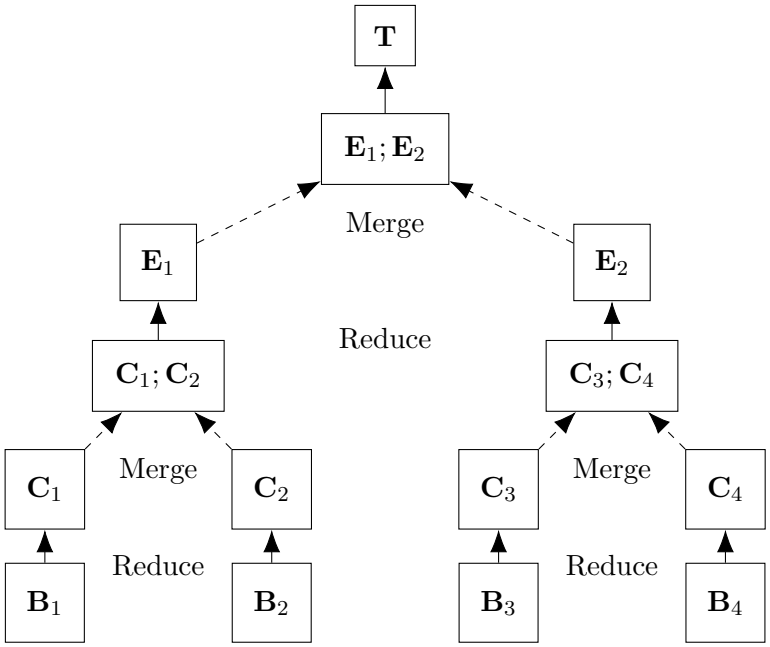


Figure 2.1: An example of the distributed summary model in a Merge − Reduce tree over blocks $\mathbf{B}_i$ of $\mathbf{A} \in \mathbb{R}^{n \times d}$. This computation tree represents 4 parties communicating with a coordinator in pairs. It could be implemented in a single streaming fashion over all 4 parties by computing the left subtree beneath $\mathbf{E}_1$ first, the right subtree beneath $\mathbf{E}_2$ second, followed by a final round of Merge − Reduce.

p_i which is the parent node of q_i and q_{i+1}. At node p_i, an operation is performed to merge the two smaller matrices $\mathbf{C}_i, \mathbf{C}_{i+1}$ to obtain $\mathbf{D}_{p_i} = \text{Merge}(\mathbf{C}_i, \mathbf{C}_{i+1})$. This process is repeated using the same Merge$(\cdot)$ *and* Reduce$(\cdot)$ *operations with the subsequent matrices being passed up the $O(1/\gamma)$ levels in the tree until we reach the root node where a single summary of bounded size is obtained. It is this final summary that is used to compute an approximation to the given problem.*

Although there might be light synchronisation at each level in the tree, because each node only computes a summary and returns to its parent, we will not include this cost in our model. Since our interest in this model will primarily be linear algebraic problems, we claim this is a reasonable cost to avoid. Usually for linear algebra algorithms, the time complexity is polynomially dependent on one of the input dimensions, hence even for moderately sized inputs, this should be more significant than synchronisation. For simplicity, Definition 2.1.4 is written using only two nodes that communicate their summaries which results in a binary computation tree, as illustrated in Figure 2.1, however this can be extended to greater than 2 communicating parties.

Remark 2.1.2. *The two models are quite close: the row arrival streaming model can be seen as a special case of the distributed model with only one*

participant who individually computes a summary, appends rows to the stored set, and reduces the new summary. This is represented as a deep binary tree, where each internal node has one leaf child as illustrated in Section 2.1.4. Likewise, the Distributed Summary Model can be implemented in a full streaming fashion over the entire binary tree.

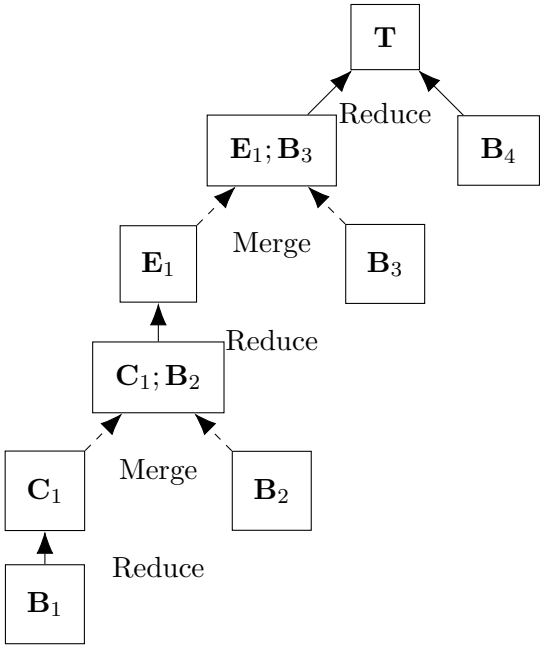


Figure 2.2: An example of the row arrival model in the Merge − Reduce language of Definition 2.1.4. A single participant reads the first block $\mathbf{B}_1$ of input matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ and performs Reduce($\mathbf{B}_1$) to obtain a small summary $\mathbf{C}_1$. Then the next block of input is read in by the operation Merge($\mathbf{C}_1, \mathbf{B}_2$) until the space budget is used and then Reduce(Merge($\mathbf{C}_1, \mathbf{B}_2$)) is executed, returning a small summary $\mathbf{E}_1$. This is repeated over all blocks of the input $\mathbf{A}$ until a single small summary $\mathbf{T}$ is returned. The error in using $\mathbf{T}$ is controlled through properties of the Reduce($\cdot$) operations.

Multi-Round Optimisation Model

Our previous computation models have been motivated by the strict requirement that the data may only be viewed once. However, often in machine learning, the dataset is too large to be stored in memory however, it can be queried for decomposable functions such as matrix-vector products. We introduce the *Multi-Round Optimisation* model for optimisation tasks such as regression in this setting.

Definition 2.1.5 (Multi-Round Optimisation)**.** *Let the input data $\mathbf{A} \in \mathbb{R}^{n\times d}$ have n observations of feature vectors from $\mathbb{R}^d$ and $n > d$. We operate in a row-wise or entrywise streaming model of computation over $\mathbf{A}$. Potentially,*

both n and d are large. Operations linear in n and d are acceptable but a superlinear dependency on either in a product is not, for example $n^{1+a}d^{1+b}$ for $0 < a, b$. The task will be to design a summary of $\mathbf{A}$ using a pass over the data. Any further access to the data will be for 'cheap' or 'simple' functions that are linear in the input size, such as evaluating an inner product or gradient, for example.

When the data is held by a single party setup, Definition 2.1.5 is reminiscent of the matrix streaming model as introduced in Definition 2.1.3. However, we will also permit entrywise arrival as well as row arrivals so that sparse sketches can be exploited. We also have the relaxation that the data can be traversed again when necessary. Similarly, in comparison to Definition 2.1.4, this model can be conceptualised as a distributed environment in which many users can communicate their local summaries *and* 'simple' functions over the input to a central coordinator. In the latter, the cost of communication could be high so both the number of communication steps *and* the size of communicated objects should be minimized. This setting is reminiscent of the so-called *federated learning* model [KMA+19] yet the key difference is the central coordinator trains a model based upon approximate information given by the participants rather than averaging each of the users' models.

2.2 Linear Algebra Background

We introduce the basic linear algebraic notions that underpin the second part of this thesis. The norm we will typically choose is the *entrywise norm* defined over vectors and matrices as follows:

Definition 2.2.1. *Let $\mathbf{A} \in \mathbb{R}^{n\times d}$ and $p > 0$. The* entrywise ℓ_p-Norm *is*

$$\|\mathbf{A}\|_p = \left(\sum_{ij} \mathbf{A}_{ij}^p\right)^{1/p}.$$

When $\mathbf{A}$ is a vector, the entrywise norm is known as the Minkowski norm. A special case when $p = 2$ is known as the *Euclidean norm* for vector inputs and denoted $\|\cdot\|_2$. If the input is a matrix then $p = 2$ is known as the *Frobenius norm* and denoted $\|\cdot\|_F$. This distinction allows us to reserve the standard notations $\|\mathbf{A}\|_2$ and $\|\mathbf{A}\|_{\text{op}}$ for the *operator* or *spectral norm* over matrices $\|\mathbf{A}\|_2 = \max_{\mathbf{x}\neq\mathbf{0}} \|\mathbf{A}\mathbf{x}\|_2 / \|\mathbf{x}\|_2$.

We will first introduce the notion of an ℓ_2-subspace embedding as a mechanism for approximating vector norms. This is often (but not always) achieved by a random linear transform $\mathbf{S} \in \mathbb{R}^{m\times n}$ that reduces the height of input data $\mathbf{A}$ from n to m, with $m = \text{poly}(d/\varepsilon)$, independent of n. Subspace embeddings are

useful as they can reduce the computational burden of dealing with matrices. If we can return a sketch $\mathbf{SA}$ of the matrix $\mathbf{A}$ in time T_{sketch}, then, ideally, any expensive operations we would usually perform on $\mathbf{A}$ using $\text{poly}(n, d)$ time can be replaced with the same operation on $\mathbf{SA}$ which will now cost $\text{poly}(d/\varepsilon)$ time. Overall, the time complexity will scale as $T_{\mathsf{sketch}} + \text{poly}(d/\varepsilon)$ which should be close to the time needed to simply *read* the data if T_{sketch} is not prohibitively large. Recall that the notation $a = (1 \pm \varepsilon)b$ is shorthand for $(1 - \varepsilon)b \leq a \leq (1 + \varepsilon)b$.

Definition 2.2.1. *[ℓ_2 Subspace Embedding] Let $\mathbf{A} \in \mathbb{R}^{n \times d}$. A matrix $\mathbf{S} \in \mathbb{R}^{m \times n}$ is a $(1 \pm \varepsilon)$ subspace embedding for the column space of $\mathbf{A}$ if for all $\mathbf{x} \in \mathbb{R}^d$:*

$$\|\mathbf{SAx}\|_2^2 = (1 \pm \varepsilon) \|\mathbf{Ax}\|_2^2 .$$

We may often abuse terminology and state that $\mathbf{S}$ and $\mathbf{SA}$ are subspace embeddings for $\mathbf{A}$. The subspace embedding property is invariant to the basis representing $\mathbf{A}$, so it is sufficient to show that it holds for any orthonormal basis for the column space of $\mathbf{A}$.

2.2.1 Oblivious ℓ_2 Subspace Embeddings

Here, we briefly survey two ways to construct ℓ_2-subspace embeddings. A more thorough treatment is given in Section 5.1.1. Our focus will be on random linear maps $\mathbf{S}$ that are independent of $\mathbf{A}$. These are referred to as being *oblivious*, since they can be sampled prior to viewing $\mathbf{A}$. Two examples of suitable random linear maps which achieve the subspace embedding property, Definition 2.2.1, with m random projections and failure probability δ are:

1. Gaussian sketch: sample a *Gaussian* matrix $\mathbf{S} \in \mathbb{R}^{m \times n}$ whose entries are iid normal $\mathbf{S}_{ij} \overset{\text{iid}}{\sim} \mathcal{N}(0, 1/\sqrt{m})$. Set $m = (d + \log(1/\delta))/\varepsilon^2$ so that $\mathbf{SA}$ takes $O(nd^2)$ time to compute.

2. $\mathsf{CountSketch}$ [CW13]: $\mathbf{S} = \mathbf{PD}$ with every column of $\mathbf{P} \in \mathbb{R}^{m \times n}$ being a uniformly selected canonical basis vector $\mathbf{e}_i \in \mathbb{R}^m$ and $\mathbf{D}$ a diagonal matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ with $\mathbf{D}_{ii} \overset{\text{iid}}{\sim} \{\pm 1\}$. Set $m = O(d^2\varepsilon^{-2}\delta^{-1})$. Note that $\mathbf{S}$ need only be defined implicitly so that it can be applied as the data is read in time $O(\mathsf{nnz}(\mathbf{A}))$. In this setup we have used uniform random sampling, however, 4-wise and 2-wise independent hash functions suffice for the row and sign selection, respectively [MM13, NN13].

These two examples highlight key compromises that must be understood when using subspace embeddings. One could use the Gaussian sketch for a very small summary in the optimal embedding dimension of $m = O(d \log(d/\delta)\varepsilon^{-2})$, yet such a summary is slow to obtain thus inhibiting its scalability. On the other

hand, a CountSketch summary can be returned quickly in time proportional to viewing the data. However, it may require more projections (d^2 in comparison to d) and has exponentially worse dependency on the failure probability (δ vs $\log 1/\delta$). There is a spectrum of compromises that can be made which cover the entire tradeoff between the space of the CountSketch and the Gaussian approach while having more acceptable time complexities, see [Woo14b] for a full review and Section 5.1.1 for our specific use-cases.

Data-Aware Embeddings

Alternative methods for obtaining subspace embeddings can be found by sampling according to the so-called *leverage scores*. Leverage scores have a history of study in statistics [CH$^+$86, VW81] but have recently been applied to summarising large matrices [DMMS11, CMM17]. The rough idea is that datapoints with a higher leverage score contribute more to a particular direction of the column space and thus are more unique. Consequently, they are more important to composing the column space of $\mathbf{A}$.

Definition 2.2.2. *Let $\mathbf{A} \in \mathbb{R}^{n\times d}$ have full column rank and $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$. The leverage scores of $\mathbf{A}$ are the row norms of an orthonormal basis for the column space of $\mathbf{A}$:*

$$\ell_i = \left\|\mathbf{e}_i^\top \mathbf{U}\right\|_2^2.$$

Although this definition appears as if the ℓ_2-leverage scores are basis dependent, one can in fact show that they are invariant with respect to the chosen basis through the quantity:

$$\ell_i = \mathbf{e}_i^\top \mathbf{A}(\mathbf{A}^\top \mathbf{A})^{-1}\mathbf{A}\mathbf{e}_i.$$

Therefore, any convenient basis will suffice to compute the leverage scores. A line of results that uses randomised leverage score sampling to achieve subspace embeddings is given in [Mah11, AM15, CMM17]. Complementary results using deterministic sampling approaches [PKB14a, McC18] are also possible. Leverage score sampling can be shown to provide a subspace embedding in the optimal projection dimension (or sample complexity) of $m = O(d\log(d/\delta)/\varepsilon^2)$ [Mah11, Woo14b].

There are some downsides to leverage score sampling. The first is that exact leverage score sampling requires obtaining an orthonormal basis in time $O(nd^2)$ which is as expensive as the SVD. Alternatively, we could find approximate leverage scores in $O(nd\log n)$ time [DMIMW12]. Nevertheless, in both cases, one would need another pass over the data to perform the sampling. Increasing the number of rows in the sample can avoid the need to revisit the data. One approach is an online sampling algorithm which inflates the sample size by $\log(\varepsilon\left\|\mathbf{A}\right\|_2^2)$ [CMP16]. On the other hand, sampling can preserve data sparsity

in the summary which random projections may not. This could be beneficial for sparse optimisation solvers that can exploit data sparsity when using the summary after the sketching step, for example [Gurobi].

2.2.2 Frequent Directions: Algorithm and Properties

An alternative type of data-aware embedding that deserves special mention is Frequent Directions (FD) [Lib13, GP14, GLPW16]. Unlike the subspace embeddings above, Frequent Directions obtains only one-sided error which is controlled through the size of the sketch. Frequent Directions is implemented in Algorithm 1 and estimates the SVD of a matrix $\mathbf{A}$ by performing streaming updates to a small number of stored directions.

Frequent Directions: Theoretical Properties

We will exploit that FD approximately preserves the norm of matrix-vector products after sketching in a similar way to random projections. Theorem 2.2.1 outlines the guarantees obtained by the returned summary $\mathbf{B} \in \mathbb{R}^{m \times d}$ of FD & *Robust Frequent Directions* (RFD). In [Hua18] it is shown that RFD improves the accuracy of FD by a factor of 2 by adding "regularisation" to every stored direction. Let $\mathbf{A}_k$ be the best rank-k approximation to $\mathbf{A}$. We use $\Delta_k = \|\mathbf{A} - \mathbf{A}_k\|_F^2$ & $\alpha = 1/m - k$ to write the bounds for both FD & RFD (2.1).

Theorem 2.2.1 ([GLPW16, Hua18])**.** *Let* $\mathbf{A} \in \mathbb{R}^{n \times d}$ *and* $m = \lceil k + 1/\varepsilon \rceil$. *Define* $\Delta_k = \|\mathbf{A} - \mathbf{A}_k\|_F^2$ *&* $\alpha = 1/m - k$. *The* (Robust) Frequent Directions *algorithm processes* $\mathbf{A}$ *one row at a time, returns a matrix* $\mathbf{B} \in \mathbb{R}^{m \times d}$ *and a scalar* δ *such that:*

$$\left\| \mathbf{A}^\top \mathbf{A} - \left(\mathbf{B}^\top \mathbf{B} + \delta \mathbf{I}_d \right) \right\|_2 \leq \alpha' \Delta_k. \tag{2.1}$$

If $[\mathbf{B}, \delta] = \mathit{FD}(\mathbf{A})$, *then* $\delta = 0$ *&* $\alpha' = \alpha$. *Else if* $[\mathbf{B}, \delta] = \mathit{RFD}(\mathbf{A})$, δ *is adaptively chosen and* $\alpha' = \alpha/2$.

A different way of writing the Frequent Directions guarantee is:

$$\|\mathbf{A}\mathbf{x}\|_2^2 - \frac{\|\mathbf{A} - \mathbf{A}_k\|_F^2}{m - k} \|\mathbf{x}\|_2^2 \leq \|\mathbf{B}\mathbf{x}\|_2^2 \leq \|\mathbf{A}\mathbf{x}\|_2^2. \tag{2.2}$$

The $\|\mathbf{A} - \mathbf{A}_k\|_F^2/m - k$ becomes $\|\mathbf{A} - \mathbf{A}_k\|_F^2/2(m - k)$ and $\mathbf{B}$ should be replaced with $(\mathbf{B}^\top \mathbf{B} + \gamma \mathbf{I}_d)^{1/2}$ if the robust version is used.

The amortised running time of FD is $O(ndm)$ to return a $m \times d$ summary $\mathbf{B}$. Obtaining this running time requires a mildly different implementation to Algorithm 1 which uses a buffer of size $2m \times d$. The SVD is only updated when the buffer is full and then the bottom half of singular directions are pruned out.

Algorithm 1: Frequent Directions (FD) and Robust Frequent Directions (RFD).

Input: Data $\mathbf{A} \in \mathbb{R}^{n \times d}$, sketch size m, method $\mathsf{Sk} \in \{\mathsf{FD}, \mathsf{RFD}\}$
Output: $\mathbf{B} \in \mathbb{R}^{m \times d}$

1 Initialise $\mathbf{B} \leftarrow \mathbf{0}_{2m \times d}$ $\rho \leftarrow 0$ ▷ Parameter for RFD
2 **for** $i = 1 : n$ **do**
3 Insert row $\mathbf{A}[i, :]$ into all zeros row of $\mathbf{B}$
4 **if** $\mathbf{B}$ *has no zero rows* **then**
5 $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^\top = SVD(\mathbf{B})$
6 $\alpha \leftarrow \sigma_m^2$
7 $\rho \leftarrow \rho + \alpha/2$
8 $\mathbf{B} \leftarrow \sqrt{\max\left(\mathbf{\Sigma}^2 - \alpha \mathbf{I}_m, 0\right)}$
9 **end**
10 **end**
11 **if** $\mathsf{Sk} = \mathsf{FD}$ **then**
12 $\delta \leftarrow 0$ ▷ $\delta = 0$ for standard FD
13 **end**
14 **else**
15 $\delta \leftarrow \rho$ ▷ δ adaptively chosen for RFD
16 **end**
17 **return** $\mathbf{B}, \delta$

This avoids the need to compute the SVD at every step; details can be found in [GLPW16]. The projection dimension should be chosen as $m = \lceil k + 1/\varepsilon \rceil$ to achieve the stated guarantee. Because we choose $m = O(1/\varepsilon)$, the running time is $O(nd/\varepsilon)$ which is faster than a single SVD on $\mathbf{A}$ for $m < d$.

Frequent Directions differs from randomised matrix sketching by *deterministically* updating the top singular directions observed in the data stream. Only the most important (or the most frequently occurring) are kept. Additionally, (2.2) shows that FD achieves deterministic one-sided error that will always underestimate a matrix-vector product. This contrasts the two-sided $(1 \pm \varepsilon)$ approximations of randomised methods from Definition 2.2.1. The space usage grows as $1/\varepsilon$ for FD which is better than the $1/\varepsilon^2$ dependency of randomised methods. Although it takes longer to obtain an FD sketch, FD more accurately approximates $\mathbf{A}^\top \mathbf{A}$ at a given projection dimension m than randomised methods [GLPW16]. FD is also a mergeable summary so operates in the distributed summary model [ACH+13], and can be adapted to sparse data [GLP16] so should still be regarded as a scalable sketch despite having time complexity $O(nd/\varepsilon)$.

Shrinkage

One might ask why the shrinkage step Line 8 of Algorithm 1 is necessary. Greedy incremental heuristics for the SVD do not perform this step but can be

shown to perform arbitrarily badly [Hua18]. Without shrinkage, setting only the final singular value to zero [Bra02], a small number of high norm directions seen early in the stream can prevent many orthogonal rows arriving afterwards with lower norm from being represented. This is because small norm directions never have large enough singular value in each of the SVD calls to enter the sketch. Hence the greedy approach would always zero out the smallest/final direction, meaning such "smaller" rows will never be promoted into the sketch despite the fact they constitute an orthogonal component of the column space [Gha17, Chapter 4.2]. In contrast, Line 8 of Algorithm 1 always subtracts off the mass of the removed directions. Thus, even if there are a (very) small number of high-norm orthogonal "corrupting" rows of $\mathbf{A}$, these corruptions will be outweighed by the normal rows.

Building Upon Frequent Directions

We use Frequent Directions in two different ways. The first is to build upon the *algorithmic idea* of FD. In Chapter 4, we design a matrix summary that operates in the same model and in a similar fashion to FD. That is, our summary reads m rows, computes a basis, prunes out rows, and then continues over the remainder of the stream. Although the algorithm operates similarly to FD, we obtain a In Chapter 6, we directly exploit the Frequent Directions guarantees for scalable training of a ridge regression model. We will exploit the structure of FD as well as its competitive space guarantee to design a highly efficient "preconditioner" for an iterative scheme to estimate the regression weights.

2.2.3 ℓ_p Subspace Embeddings

Subspace embeddings are not restricted to only the ℓ_2 norm and we can introduce a similar concept over arbitrary $p > 0$.

Definition 2.2.2. *Let $\mathbf{A} \in \mathbb{R}^{n\times d}, p > 0$ and $c_1, c_2 \geq 0$ be constants. A matrix $\mathbf{T} \in \mathbb{R}^{m\times d}$ is a (c_1, c_2) ℓ_p-subspace embedding for the column space of $\mathbf{A}$ if for all $\mathbf{x} \in \mathbb{R}^d$:*

$$c_1 \|\mathbf{Ax}\|_p^p \leq \|\mathbf{Tx}\|_p^p \leq c_2 \|\mathbf{Ax}\|_p^p .$$

Definition 2.2.2 is simply a generalisation of Definition 2.2.1 to arbitrary $p > 0$. Note that we alter the presentation in using $\mathbf{T}$ rather than $\mathbf{SA}$. We may of course take $\mathbf{T} = \mathbf{SA}$ for ℓ_2, similar representations hold when $\mathbf{S}$ is the so-called *Fast Cauchy Transform* in ℓ_1 [CDMI$^+$16], or related constructions for $\ell_{p\in[1,2]}$ [MM13]. In ℓ_1, one can use sparse random projections to obtain a $(c_1, c_2) = (1, O(d \log d))$ ℓ_1-subspace embedding which is also shown to be optimal in [WW19]. However, it will be convenient to use the presentation

from Definition 2.2.2 due to the algorithms we present in Chapter 4 which return summaries that are not linear transformations of the input data.

2.2.4 Linear Algebra Problems

The two classes of linear algebra problems we study are *ℓ_p-regression* and *low-rank approximation*. The ℓ_p-regression task is to find the hyperplane which has the smallest distance to all points in $\mathbf{A}$ under the ℓ_p loss. This reconstruction can then be used for later prediction tasks. Similarly, we can think of low-rank approximation as trying to approximately reconstruct the input data using fewer features than are present in the original input.

ℓ_p-Regression

Definition 2.2.3. *Let $\mathbf{A} \in \mathbb{R}^{n \times d}$ be the sample-by-feature input data and $\mathbf{y} \in \mathbb{R}^n$ be the* target vector*. The (exact) ℓ_p-*regression problem *for $0 < p < \infty$ is to evaluate:*

$$\min_{\mathbf{x} \in \mathbb{R}^d} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_p^p \tag{2.3}$$

The approximate ℓ_p-regression problem *is to find an $\hat{\mathbf{x}} \in \mathbb{R}^d$ such that:*

$$\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{y}\|_p^p \leq c \cdot \min_{\mathbf{x} \in \mathbb{R}^d} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_p^p \tag{2.4}$$

There are various related problems that are covered by Definition 2.2.3. For example, when $p = 2$ we obtain *ordinary least squares regression* and $p = 1$ yields *least absolute deviation regression*. Rather than optimising over all $\mathbf{x} \in \mathbb{R}^d$, occasionally we may study variants which use a convex constraint set $\mathcal{K}$. The constraint set may define certain ℓ_p-balls which can result in penalised forms of the regression problem (2.3) such as *ridge regression*, *LASSO regression* or a combination of both, *elastic net regression*. We will not study the cost of projection onto these constraint sets, which sometimes can be expensive. Our focus on the general algorithmic techniques for solving a large instance of regression through a reduction to a smaller instance.

In Chapter 5 we will study how sketches can be used to solve the approximate ℓ_2 regression problem in the above setup. The main idea that is used to approximate these regression tasks is to sketch down to a small dimension using a subspace embedding and solve the reduced-size problem. The fastest algorithm for ℓ_2 regression uses a $\mathsf{CountSketch}$ for the embedding to achieve a $c = 1 + \varepsilon$ in (2.4) in time $O(\mathsf{nnz}(\mathbf{A})) + \mathrm{poly}(d/\varepsilon)$. For ℓ_1 regression, a similar but more involved argument can also be shown to achieve $c = 1 + \varepsilon$ in time $O(\mathsf{nnz}(\mathbf{A})) + \mathrm{poly}(d/\varepsilon)$ [MM13].

Low-Rank Approximation

A second fundamental problem in Numerical Linear Algebra is that of *Low-Rank Approximation*. This problem is motivated by redundancy in the dataset, typically over the spectrum of the input data. The input data is of size $n \times d$ which can be large. If many of the directions are irrelevant, uninformative, or noisy, it can be much cheaper to store a *rank-k* approximation that exploits low-dimensional structure in the data. Such an approach yields a factorisation of $\mathbf{A}$ into two matrices $\mathbf{U} \in \mathbb{R}^{n\times k}$ and $\mathbf{V} \in \mathbb{R}^{d\times k}$ so that $\mathbf{A} \approx \mathbf{U}\mathbf{V}^\top$. The benefit of low rank approximation is that now only $O((n+d)k)$ values need to be stored which can be substantially less than $O(nd)$ if k is small. Secondly, matrix-vector products can now be carried out in time $O((n+d)k)$ which is again a saving over $O(nd)$. Applications of low-rank approximation include non-negative matrix factorisation [FI11] which is useful in recommender systems, latent semantic analysis, and forms of clustering [DHS05].

Definition 2.2.4. *Let $\mathbf{A} \in \mathbb{R}^{n\times d}$ be the sample-by-feature input data. The (exact)* ℓ_p-low rank approximation problem *for $0 < p < \infty$ is to solve*

$$\min_{\mathbf{A}':\mathrm{rank}(\mathbf{A}')=k} \left\|\mathbf{A}-\mathbf{A}'\right\|_p.$$

The approximate ℓ_p-low rank approximation problem *is to return $\hat{\mathbf{A}}$ which satisfies:*

$$\left\|\mathbf{A}-\hat{\mathbf{A}}\right\|_p \leq c \cdot \min_{\mathbf{A}':\mathrm{rank}(\mathbf{A}')=k} \left\|\mathbf{A}-\mathbf{A}'\right\|_p \tag{2.5}$$

The solution matrix in both settings should be returned in factored form *as $\mathbf{U}\mathbf{V}^\top$ with $\mathbf{U} \in \mathbb{R}^{n\times k}$ and $\mathbf{V} \in \mathbb{R}^{d\times k}$ to obtain computational benefits.*

Solving Low Rank Approximation

For a *limited* class of problems certain solutions are known. The SVD of $\mathbf{A}$ yields the optimal solution in both Frobenius ($p = 2$) and spectral norms [EY36]. However, this takes $O(nd^2)$ time and $O(nd)$ space so sketched approximations have been proposed. The fastest sketching algorithm for low-rank approximation in Frobenius norm is given by [CW17] in time $O(\mathsf{nnz}(\mathbf{A})) + (n+d)\,\mathrm{poly}(k)$ to obtain $c = 1+\varepsilon$. When assumptions such as Gaussian noise or the presence of outliers are made, $p = 1$ may be more robust than $p = 2$ [SWZ17]. However, solving the low-rank approximation problem for $p = 1$ is known to be NP-hard [GV18]. For $p = 1$, a solution obtaining $c = \log d\,\mathrm{poly}(k/\varepsilon)$ in time $O(\mathsf{nnz}(\mathbf{A})) + (n+d)\,\mathrm{poly}(k/\varepsilon)$ was given in [SWZ17].

Sampling approaches are common for ℓ_2-low rank approximation in machine learning. In this setting, one asks for a structured low-rank approximation $\hat{\mathbf{A}} = \mathbf{CUR}$ where $\mathbf{C}$ is a subset of the columns of $\mathbf{A}$, $\mathbf{U}$ is a low rank matrix,

and $\mathbf{R}$ is a subset of rows from $\mathbf{A}$ [MD09]. This is closely related to the Nyström method [WS01] which can also be approximated using only a linear number of kernel function evaluations [MM17]. However, a full comparison to sketching for kernel methods is outside the scope of this thesis and the reader should consult [MW17, DKM20, GM13].

2.3 Optimisation Background

We will introduce sketching analogues for the following methods for solving regression. The methods introduced below are standard so this section can be skipped for readers familiar with using direct solvers, gradient descent, and Newton methods for regression. For simplicity, we focus solely on standard "deterministic optimisation" algorithms rather than stochastic optimisation methods (such as stochastic gradient descent). Nevertheless, extending sketches into the stochastic optimisation regime could be an exciting future direction [YCRM17]. For simplicity, we focus on ordinary least squares, which has the objective function, gradient and Hessian given below. Although we focus on this simple case, many of the ideas directly map across to penalised or constrained forms of ℓ_2 regression.

$$f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2 \tag{2.6}$$

$$\nabla f(\mathbf{x}) = \mathbf{A}^\top(\mathbf{A}\mathbf{x} - \mathbf{y}) \tag{2.7}$$

$$\nabla^2 f(\mathbf{x}) = \mathbf{A}^\top\mathbf{A}. \tag{2.8}$$

2.3.1 Direct Solver

We split the time costs into *setup time* and *solve time*. The setup time is the preprocessing time required to put the data into a form so that one can simply call a 'solve' operation that we treat as a black box. There are roughly two approaches that a direct solver may use to solve regression when $n \gg d$ which are introduced below and summarised in Table 2.1.

1. Compute a QR decomposition or SVD of the input matrix [GVL13]. Suppose we obtain the SVD of $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top \in \mathbb{R}^{n\times d}$ requiring a *setup time* of $O(nd^2)$ and $O(nd)$ space to store the orthonormal basis $\mathbf{U} \in \mathbb{R}^{n\times d}$. The solution vector is then $\mathbf{x}^\star = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^\top\mathbf{y}$. Hence, the *solve time* is $O(d^2 + nd)$, assuming the SVD is given.

2. Setup the normal equations $\mathbf{A}^\top\mathbf{A}\mathbf{x} = \mathbf{A}^\top\mathbf{y}$ explicitly in time $O(nd^2)$. This can be done on a data stream using rank-one updates of the rows of $\mathbf{A}$. The space usage is $O(d^2)$ which is sublinear in n unlike the SVD

Method	Time		Space
	Setup	Solve	
SVD	$O(nd^2)$	$O(d^2 + nd)$	$O(nd)$ for $\mathbf{U}$
Normal Equations	$O(nd^2)$	$O(d^3)$	$O(d^2)$ for $\mathbf{A}^\top\mathbf{A}$

Table 2.1: Comparison of two direct solver approaches.

approach above. The worst-case solve time is then $O(d^3)$ assuming that $\mathbf{A}^\top\mathbf{A}$ has no special structure.

As computing the SVD of $\mathbf{A}$ is not practical for large-scale data, we will assume that the regression problem is solved using method 2. The cost is then $O(nd^2)$ setup time, $O(d^3)$ solve time and $O(d^2)$ space. The space bound is $O(d^2)$ as outer products of rows $\mathbf{A}_i\mathbf{A}_i^\top \in \mathbb{R}^{d\times d}$ can be accumulated. This applies even in the row-arrival (Definition 2.1.3) or multi-party computation models (Definitions 2.1.4 and 2.1.5), highlighting that the models we have introduced capture common machine learning settings.

2.3.2 Iterative Methods

A variety of iterative methods can also be used for solving regression that take multiple computation rounds to update the weights. These methods operate in the *multi-round optimisation model* (Definition 2.1.5). As the data is viewed many times, we measure the cost of a single iteration in time and space. Another concern is the number of iterations rounds we must take. Every round needs another pass over the data, which may be acceptable for small data, but is a substantial cost if the data is large or distributed. Surveying all of these methods is beyond the scope of this chapter so we detail only the most pertinent. Table 2.2 provides a brief summary for readers familiar with these methods.

Gradient Descent

The time cost of $O(nd^2)$ may be prohibitive if d is moderate to large, so to avoid this cost gradient descent is employed which only depends linearly on n and d for some desired accuracy. Gradient descent is an iterative scheme that exploits linear approximation of the quadratic objective function about a fixed point. The iterates are defined as:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \nu_t \nabla f(\mathbf{x}^{(t)}) \tag{2.9}$$

The parameter ν_t is referred to as the *step size* and *strongly* affects the convergence behaviour of a gradient descent. For general convex functions

gradient descent needs to the number of steps to grow linearly in $1/\varepsilon_\star$ as $O(1/\varepsilon_\star)$ to return an estimate within $\varepsilon_\star$ relative error [BBV04, Chapter 9].[2] $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 \leq \varepsilon_\star \|\mathbf{x}^\star\|_2$. This is referred to as *sublinear convergence*.

On the other hand, since data $\mathbf{A}$ is full-rank, the least squares function is *strongly convex*. This means that the second derivate $\nabla^2 f(\mathbf{x}) = \mathbf{A}^\top \mathbf{A}$ has eigenvalues bounded between positive constants L_1 and L_2. For the least squares objective function, we have $\sigma^2_{\min}(\mathbf{A}) \leq \lambda_i(\nabla^2 f(\mathbf{x})) \leq \sigma^2_{\max}(\mathbf{A})$ so f is strongly convex with $L_1 = \sigma^2_{\min}(\mathbf{A})$ and $L_2 = \sigma^2_{\max}(\mathbf{A})$. For strongly convex functions the number of steps needed to achieve $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 \leq \varepsilon_\star \|\mathbf{x}^\star\|_2$ grows logarithmically in $1/\varepsilon_\star$. This is an exponential improvement over the general convex setting above and is referred to as *linear convergence*.[3]

However, in both settings, the step size η_t must be correctly tuned. The theoretically optimal step size for a gradient step is $\eta_t = 2/(\sigma^2_{\max}(\mathbf{A}) + \sigma^2_{\min}(\mathbf{A}))$ which can be pessimistic in practice and requires knowledge of the singular values of $\mathbf{A}$ [BBV04, Chapter 9]. Moreoever, we may have no knowledge of the spectrum of $\mathbf{A}$ meaning that we do not know how to set the step size ahead of time. Hence, gradient descent is brittle to changes in η_t and its effect on the parameter E.

One way of seeing the effect of the step size is the following argument. Suppose that $\mathbf{A}$ is full-rank. Initialise $\mathbf{x}^{(0)} = \mathbf{0}_{d\times 1}$ and select $\nu_t = 1/\sigma^2_{\max}(\mathbf{A})$. Then:

$$\begin{aligned}
\left\|\mathbf{x}^{(t+1)} - \mathbf{x}^\star\right\|_2 &\leq \left\|\mathbf{I}_d - \frac{1}{\sigma^2_{\max}(\mathbf{A})}\mathbf{A}^\top\mathbf{A}\right\|_2 \left\|\mathbf{x}^{(t)} - \mathbf{x}^\star\right\|_2 \\
&\vdots \\
&\leq \left\|\mathbf{I}_d - \frac{1}{\sigma^2_{\max}(\mathbf{A})}\mathbf{A}^\top\mathbf{A}\right\|_2^{t+1} \|\mathbf{x}^\star\|_2
\end{aligned}$$

which will descend towards $\mathbf{x}^\star$ provided that $\left\|\mathbf{I}_d - \frac{1}{\sigma^2_{\max}(\mathbf{A})}\mathbf{A}^\top\mathbf{A}\right\|_2 < 1$. Equivalently, by the SVD of $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ and orthonormality of $\mathbf{V}$:

$$\left\|\mathbf{I}_d - \frac{1}{\sigma^2_{\max}(\mathbf{A})}\mathbf{A}^\top\mathbf{A}\right\|_2 = \left\|\mathbf{I}_d - \frac{1}{\sigma^2_{\max}(\mathbf{A})}\mathbf{\Sigma}^2\right\|_2.$$

Since $\mathbf{\Sigma}^2$ is diagonal the matrix in the norm above is diagonal so the spectral norm is the largest diagonal entry. This occurs at the *smallest* singular value of $\mathbf{A}$. Hence:

$$\left\|\mathbf{I}_d - \frac{1}{\sigma^2_{\max}(\mathbf{A})}\mathbf{A}^\top\mathbf{A}\right\|_2 = 1 - \frac{\sigma^2_{\min}(\mathbf{A})}{\sigma^2_{\max}(\mathbf{A})} \in (0, 1).$$

[2] Thought of as the number of decimal place to which the estimate and solution agree.
[3] Sometimes also confusingly called exponential convergence.

Let $E = 1 - \sigma^2_{\min}(\mathbf{A})/\sigma^2_{\max}(\mathbf{A})$ so that after $t+1$ iterations the error is E^{t+1}. For a given accuracy tolerance $\varepsilon_\star$, we must complete T iterations so that $E^T = \varepsilon_\star$. Thus, T grows as

$$T = \frac{\log(1/\varepsilon_\star)}{\log(1/E)} \tag{2.10}$$

which has a satisfying dependence on $\log(1/\varepsilon_\star)$. On the other hand, it depends heavily on $1/\log(1/E)$ which can be large even for full-rank data $\mathbf{A}$. A similar argument holds if we choose the optimal η_t with adjusted constants.

Absent of knowledge about the spectrum of $\mathbf{A}$, one often resorts to heuristics to obtain an appropriate step size. Methods such as approximate line search only change the (asymptotic) number of steps required for convergence by constants [BBV04]. Although performance looks reasonable if f is strongly convex and the correct step size can be tuned, if either of these assumptions is not met, performance degrades to the general convex setting (even for OLS) and the number of steps required grows as $T = O(1/\varepsilon_\star)$ which is slow.

In terms of computational complexity, the time cost *per iteration* is simply $O(nd)$ as only inner products are needed for the gradient. Equation (2.7) can be computed from right to left in the parentheses so $\mathbf{A}^\top\mathbf{A}$ is not materialised. We execute T gradient steps and if $T < d$, then this can be attractive in comparison to a direct solver. In terms of space, only $O(d)$ *working space* is needed to store vectors at any given iteration, however, multiple passes of the data are necessary and the number of passes is dependent on the accuracy as well as the quantity E.

In summary, gradient descent is simple and requires only inner products of gradient vectors. When compared to our computation models from Section 2.1.4, one can consider gradient descent as a distributed protocol that requires communicating $O(d)$ sized vectors, however, the slow convergence also means *many* (possibly expensive) rounds of communication are necessary to obtain accurate estimators. In the models from Section 2.1.4, we can consider the central coordinator as holding approximate weights $\hat{\mathbf{x}}$ at any given time point. A pass through the dataset is taken to evaluate $\nabla f(\hat{\mathbf{x}}) = \mathbf{A}^\top(\mathbf{A}\hat{\mathbf{x}} - \mathbf{y})$ which is used to update the weights.

Newton's Method

Newton's method is conceptually similar to gradient descent except a quadratic approximation to f is used at every point rather than a linear approximation. The Newton iterates are mild variants on the gradient method ([BBV04, Chapter 9]):

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - [\nabla^2 f(\mathbf{x}^{(t)})]^{-1}\nabla f(\mathbf{x}^{(t)}). \tag{2.11}$$

The key difference is that rather than a step parameter, we evaluate the second derivative at every iterate. Note that we need not obtain the inverse explicitly and can solve the linear system

$$\nabla^2 f(\mathbf{x}^{(t)}) \left(\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}\right) = -\nabla f(\mathbf{x}^{(t)}). \tag{2.12}$$

Measured by number of iterations, convergence using Newton's method is faster than for gradient descent[4] but it comes at the cost of (i) forming the Hessian matrix in space $O(d^2)$ space and (ii) solving a linear system needing $O(d^3)$ time per iteration. Consider the specific case of ordinary least squares and an initial guess of $\mathbf{x}^{(0)} = \mathbf{0}_d$. Newton's method will converge in a single step as the update is equivalent to solving the normal equations. There is a tradeoff to strike; substantially fewer iterations are necessary but at a much greater cost, so it isn't immediate from a practical or theoretical perspective which should be preferred.

Approximate Newton Method

The high cost of a single Newton step often makes it impractical to use. Hence, we are more interested in approximating the Hessian matrix $\mathbf{H} = \nabla^2 f(\mathbf{x})$ to estimate the solution of (2.12). The approximation $\hat{\mathbf{H}}$ should be cheaper to compute than obtaining the Hessian exactly.

The main idea is that the Newton method uses $\mathbf{H}^{-1}\nabla f(\mathbf{x})$ at every step. This "normalises" all of the directions so that taking a unit step length is appropriately scaled based on the singular values' magnitude. We will use this idea with an estimate of the Hessian to take *approximately* unit-sized step in every direction. The iterates follow the update rule

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - [\widetilde{\nabla^2 f(\mathbf{x}^{(t)})}]^{-1}\nabla f(\mathbf{x}^{(t)}). \tag{2.13}$$

where $\widetilde{\nabla^2 f(\mathbf{x}^{(t)})}$ is an appropriate estimate of $\nabla^2 f(\mathbf{x}^{(t)})$. For Ordinary Least Squares (OLS), we recall that the Hessian is constant over $\mathbf{x}$ so this amounts to finding a good estimate $\hat{\mathbf{H}}$ for $\mathbf{H} = \nabla^2 f(\mathbf{x})$. The iterates (2.13) can be performed in time $O(d^2 + nd)$ if $\hat{\mathbf{H}} \in \mathbb{R}^{d\times d}$ is explicitly materialised. In Chapter 5, we will investigate better ways to implement this step.

Similar to gradient descent, the approximate Newton method can be performed in the distributed setting. Now the central coordinator holds an estimate to the Hessian matrix as well as the estimated weights $\hat{\mathbf{x}} = \mathbf{x}^{(t)}$. Of course, this requires a slight increase in the size of communicated objects from $O(d)$ to

[4]For general convex functions, Newton's method is only locally convergent, so one is not necessarily guaranteed to descend towards the optimal solution. However, for the least squares problems we will consider, this will not be a problem as we will show that initialising at zero suffices for convergence.

Method	Iteration Cost		Number of iterations
	Space	Time	
Gradient Descent	$O(d)$	$O(nd)$	$O(\log(1/\varepsilon))$
Approximate Newton Method	$O(md)$	$O(md + nd)$	$O_m(\log(1/\varepsilon))$
Newton Method	$O(d^2)$	$O(d^3)$	1

Table 2.2: Comparison of the related iterative methods for ordinary least squares on full-rank data. The parameter m should be used to control the space and time usage per iteration and should be thought of as a tunable parameter that determines how well the Hessian is approximated. Consequently, different values of m will have an impact on how many iterations must be completed, represented by the $O_m(\cdot)$ notation. The role of m and the number of iterations will become clear in Chapter 5.

$O(d^2)$, however the hope is that many fewer communication rounds are necessary than gradient descent. More details on how we exploit the approximate Newton method are presented in Section 5.1.3.

In summary for OLS, the approximate Newton method seeks a tradeoff between gradient descent and the (exact) Newton method. Gradient descent has a cheap per-iteration cost as it needs only inner products to be computed for every step; however, many iterations are necessary. Newton's method is expensive for every iteration as a the $d \times d$ Hessian must be obtained, followed by a linear solve, yet few iterations are needed. For the particular case of OLS, exact Newton is equivalent to solving the normal equations. The approximate Newton method exploits spectral information of the data so that better steps can be taken than in gradient descent, meaning fewer iteration rounds are necessary. Also, a good approximate Newton scheme should efficiently find the estimate $\hat{\mathbf{H}}$ to yield computational gains over the exact Newton method. These relationships are summarised in Table 2.2.

2.4 Communication Complexity

We introduce the Index problem which is used in Chapters 3 and 4 to prove space lower bounds required to solve a problem up to certain approximation factors (e.g. a constant factor). The typical setup for Index is given in the following definition.

Definition 2.4.1. *Alice holds a vector* $\mathbf{a} \in \{0,1\}^N$ *and Bob holds an index* $i \in [N]$*. Bob is tasked with finding* $\mathbf{a}_i$ *following communication with Alice.*

Definition 2.4.1 is the *general* Index problem. Note that if there are no requirements on the direction or number of communication rounds, Bob can output $\mathbf{a}_i$ exactly in 2 rounds of communication by sending Alice i and Alice

returning $\lceil \log N \rceil$ bits for $\mathbf{a}_i$. However, we will focus on the *one-way communication complexity* model. In this setting, Alice communicates information to Bob so that Bob can output $\mathbf{a}_i$. Alice must send Bob $\Omega(N)$ (a linear number of bits in the size of Alice's vector $\mathbf{a}$) bits for him to solve the Index problem [KNR99]. We use this model because the size of the message Alice sends to Bob is the same as the space consumption of an algorithm that Bob can query.

Index has been used many times in proving lower bounds for classical streaming algorithms, such as frequency moment estimation [LW13] or finding the median of all items seen on a stream [GM09]. These ideas have been extended to matrix problems [Woo14a] and we will adapt this communication problem for our lower bounds. Unlike typical reductions to Index, we require nontrivial constructions from *coding theory* which will generate a sufficiently large instance to our input problem. Our lower bounds will typically make use of a *binary* code $\mathcal{C}$, comprised of a collection of *codewords*, which are vectors (or strings) of fixed length. Informally, we will sample a code $\mathcal{C} \subset \{0,1\}^d$ with certain properties, such as low weight and small number of collisions amongst $\mathbf{u}, \mathbf{u}' \in \mathcal{C}$ so that $|\mathcal{C}| = 2^{\Omega(d)}$. The input to the problem may then be either the codewords in the raw state, or some function of the codewords. Upon defining a suitable instance, we will argue that Bob can solve Index, and thus incur the associated lower bound.

2.4.1 Outlining Lower Bound Arguments

More concretely, let $\mathcal{C}$ be a binary code with $|\mathcal{C}| = N$ and suppose Alice holds a subset of words $T \subset \mathcal{C}$. Alice's input T encodes a *binary membership vector* $\mathbf{a} \in \{0,1\}^N$ such that $\mathbf{a}_i = 1$ if Alice holds the string $\mathbf{w}_i \in T$. Note here that we simply need a suitable enumeration over $\mathcal{C}$ to generate the indexing i of $\mathbf{a}$. Since we can take the canonical mapping over $[N]$ into binary, this is not problematic. We will abuse notation to refer to binary strings $i \in \mathcal{C}$; meanwhile when i is used as an index, it is assumed to be in binary representation. By this we mean that if the code is $\mathcal{C} = \{00, 01, 10, 11\}$ and Alice holds $T = \{00, 10\}$, then the binary representation of Alice's codewords are $00 \mapsto 0$ and $10 \mapsto 2$ so $\mathbf{a} = (1, 0, 1, 0)$. It is precisely this vector $\mathbf{a}$ whose indices Bob would like to learn using communication from Alice.

Roughly, our approach will be as follows. As above, Alice holds $T \subset \mathcal{C}$ and populates her membership vector $\mathbf{a}$. She will define an instance $\mathbf{A}$ for the given problem P based on $\phi(\mathbf{x})$ for every codeword $\mathbf{x}$ she holds in T. Here, ϕ is the identity operator if no transformation is made (Chapter 5, Theorem 4.5.2), or a particular function we may define if we need something more sophisticated. In the simple case that ϕ is the identity map, then $\mathbf{A} \in \{0,1\}^{N \times d}$ is simply a matrix whose rows are the codewords from Alice's held set T. However, for

the more complex constructions of ϕ we will use in Chapter 3, $\mathbf{A}$ may have many more rows than N and we will have to address concerns regarding the size of $\mathbf{A}$, ensuring that it does not grow unboundedly large.

Bob's Approach

Independent of any information Alice has, Bob is also given a test vector $\mathbf{y} \in \mathcal{C}$ so that $\mathbf{y}$ is guaranteed to share properties of the code that we have determined a priori. We will then hypothesise that there exists an algorithm **ALG** for the problem **P** with a certain approximation guarantees. For example, **ALG** returns a constant factor approximation to **P**. Bob will query **ALG** using the input $\mathbf{A}$ and his task is to distinguish between two cases: (i) $\mathbf{y} \in T$; (ii) $\mathbf{y} \in \mathcal{C} \setminus T$. These two cases are sufficient for Bob, because if he can tell which of (exactly one) (i) or (ii) is true, then he can return the value of Alice's membership vector $\mathbf{a}$. Note again, that Bob doesn't need to know the underlying indexing scheme that Alice has used to populate $\mathbf{a}$, all he needs is that if $\mathbf{y} \in T$ then Alice will have $\mathbf{a}_y = 1$, that is, case (i) is true. Otherwise, case (ii) is true and Bob knows that $\mathbf{a}_y = 0$. Hence, Bob is able to solve the Index problem.

Although this appears simple, the trick is in how we design both $\mathcal{C}$ and ϕ so both the code and the test instance $\mathbf{A}$ have certain properties we can reason against in establishing the performance of **ALG**. The second difficulty is in determining the separation between the two cases (i) and (ii) and how that interacts with the approximation factor guarantee of the algorithm. In particular, if we assume **ALG** returns a constant factor approximation for **P**, then we need to be able to show a constant factor gap between some output corresponding to case (i) and case (ii) which will involve some combinatorial calculations. At this point, we can be satisfied Bob can determine which case he is in, thus solving Index and incurring the space bound of $\Omega(N)$ which is the size of Alice's membership vector. Since $N = |\mathcal{C}|$ and $|\mathcal{C}| = 2^{\Omega(d)}$, the space cost for an algorithm with guarantees of **ALG** will be $2^{\Omega(d)}$.

Chapter 3

Projected Frequency Estimation

In this chapter, we are interested in "projected frequency estimation" which can be understood as follows. Given an $n \times d$ dimensional dataset $\mathbf{A}$, a projection query specifies a subset $S \subseteq [d]$ of columns which yields a new $n \times |S|$ array. We present a theoretical study of the space complexity for computing data analysis functions over such subspaces, including heavy hitters and norms, when the subspaces are revealed only after observing the data. We show that this important class of problems is typically hard: for many problems, we show $2^{\Omega(d)}$ lower bounds. Our results are based on careful constructions of instances using coding theory and novel combinatorial reductions that exhibit such space-approximation tradeoffs. However, we present upper bounds which demonstrate space dependency better than 2^d. That is, for $c, c' \in (0, 1)$ and a parameter $N = 2^d$ an N^c-approximation can be obtained in space $\min(N^{c'}, n)$, showing that it is possible to improve on the naïve approach of keeping information for all 2^d subsets of d columns. More concretely, the accuracy-space tradeoff is controlled by a tunable parameter $\alpha \in (0, 1/2)$. We have $c' = H(1/2 - \alpha)$ where $H(t) \leq 1$ is the binary entropy function.[1] For the error, we can think of $c \approx |1 - p|\alpha$ when operating in ℓ_p, which is less than 1 for appropriate α.

The computation model outlined above has not been previously studied so before moving on to the technical details, we will show a high level illustrative example. This example is intended only to present the main setup and motivation for our chapter, enabling the reader to more easily conceptualise the mathematical detail given at a later stage. Our aim in this example is to show how the model is motivated and also how a user could use the model in practice.

[1] See Figure 3.4

Preface: Practical Motivation

Suppose that an online advertising broker has deployed a new set of adverts whose performance they wish to monitor. We can imagine that they have a set of D **adverts** a_j that may or may not be presented to their n **users** u_i. The presentation of advert a_j to user u_i can be represented as simple binary `Yes`, `No` values. Following this, the users experience a set of **events** that we will assume for simplicity are binary. The events may be relatively simple, for example, suppose the broker also monitors whether

- users **interact** with the advert in full. Traditionally, users might have interacted with adverts through simple clicks, but nowadays there are a huge number of possible formats such as click-through ads, product trailers or banner adverts on YouTube videos, Spotify adverts between different audio media, Instagram product adverts and many more. The `Yes`/`No` variable now represents whether a click-ad was clicked or a video advert was watched in full, an entire Instagram product advert has been swiped and so on. A user u_i's interaction with advert a_j is denoted by $e_j \in \{\texttt{Yes}, \texttt{No}\}$. We make the distinction of interacting with the advert in full to prevent enforcing too much redundancy in the available patterns as this allows each $(a_j, e_j) \in \{(\texttt{Yes}, \texttt{Yes}), (\texttt{Yes}, \texttt{No}), (\texttt{No}, \texttt{No})\}$ but clearly (a_j, e_j) cannot be $(\texttt{No}, \texttt{Yes})$ as the user must at least observe the advert a_j before being capable of interacting with it.

- users give **feedback** on the quality or usefulness of the advert a_j which is represented by f_j. For our example we assume the feedback is initially neutral/negative (`No`) or the user gives positive feedback (`Yes`).

For user u_i, we generate the entire length $d = 3D$ **user pattern** which is the binary string describing their behaviour over each of the adverts and its events. An example user pattern is illustrated in the following table:

	Adverts and Events						
Users	a_1	e_1	f_1	$\ldots$	a_D	e_D	f_D
u_i	`Yes`	`Yes`	`No`		`Yes`	`No`	`Yes`

This setup leads to a sequence of `Yes`, `No` bitstrings for every user u_i which generates the input array describing the user-level behaviour across all of the adverts. Over all of the users in the company's database, the behaviour is represented as a array $\mathbf{A} \in \{0, 1\}^{n \times d}$ under the mapping $0 \mapsto \texttt{No}$ and $1 \mapsto \texttt{Yes}$ with every row $\mathbf{A}_i$ representing user u_i's behaviour across every advert-event tuple. At this point, having curated their user data, the company can begin to ask analytical questions about the behaviour.

The Broker's Analysis

Suppose that the broker wishes to answer to the following question.

Question 3.0.1. *How many users were presented with adverts a_1 and a_2, interacted with each of them, and gave positive feedback to both?*

Such behaviour is described by the bitstring

$$(a_1, e_1, f_1, a_2, e_2, f_2) = \mathbf{1}_{1\times 6}.$$

Under this model, one could pass through the array $\mathbf{A}$ and count the number of times $\mathbf{1}_{1\times 6}$ is present amongst the columns

$$S = (a_1, e_1, f_1, a_2, e_2, f_2).$$

Of course, this appears to be a simple counting problem. However, the computational and statistical challenges of counting the number of distinct items from large-scale data *in sublinear space* inspired the (standard) streaming model [AMS99b].

Alternatively, the broker might be interested in slightly different queries such as

Question 3.0.2. *How many different behaviours were observed on adverts a_1 and a_2?*

Although, questions 3.0.1 and 3.0.2 are related, there is a subtle difference in that 3.0.1 asks for the frequency of item $\mathbf{1}_{1\times 6}$ while 3.0.2 asks how many distinct user patterns from $\{0,1\}^{|S|}$ are observed on columns S. Indeed, if the broker knows the columns of interest before generating or observing $\mathbf{A}$, then they could simply maintain these statistics as the data is observed. On the other hand, it is plausible that the user-base or number of advert-event tuples is so large that generating this data even for a small timeframe (for instance, the time window across the halftime interval at the Super Bowl) yields an extremely large array $\mathbf{A}$, so large that taking another pass over it is not practical. In such a situation, the columns of interest, S, are decided *after* observing the data. As a result, the analyst can no longer simply scan through $\mathbf{A}$ and count the patterns as they are seen. Consequently, a new approach is needed to deal with this scenario in which we face both large-scale and high-dimensional data.

Extending the example, suppose the company has 4 users in its database with the user patterns described in the following table.
If this were the input data, then the answer to Question 3.0.1 would be 1 as only u_1 has a user pattern of $\mathbf{1}_{1\times 6}$. Question 3.0.2 has an answer of 4 as each user $u_1, \ldots, u_4$ has a unique user pattern.

	Adverts and Events					
Users	a_1	e_1	f_1	a_2	e_2	f_2
u_1	Yes	Yes	Yes	Yes	Yes	Yes
u_2	Yes	No	No	No	No	No
u_3	No	No	No	Yes	Yes	Yes
u_4	Yes	Yes	No	Yes	No	Yes

In a nutshell, the broker is interested in estimating the frequency of certain patterns after the data has been observed. The data is so large that it cannot be accessed in full so some technique of summarisation is necessary in order to approximately answer queries. These queries are roughly of the form *how many different behaviours were experienced on a chosen advert-event columnset?* Or similarly, *how many times was a given advert-event behaviour observed?* This general setup motivates our subsequent work.

3.1 Introduction

In many data analysis scenarios, datasets of interest are of moderate to high dimension, but many of these dimensions are spurious or irrelevant. Thus, we are interested in subspaces, corresponding to the data projected on a particular subset of dimensions. Within each subspace, we are concerned with computing statistics, such as norms, measures of variation, or finding common patterns. Such calculations are the basis of subsequent analysis, such as regression and clustering. In this chapter, we introduce and formalize novel problems related to functions of the frequency in such projected subspaces. Already, special cases such as subspace projected distinct elements have begun to generate interest, e.g., in Vu's work [Vu18], and as an open problem in sublinear algorithms [Sublinear Open Problems: 94].

In more detail, we consider the original data to be represented by a (usually binary) array with n rows of d dimensions. A subspace is defined by a set $S \subseteq [d]$ of columns, which defines a new array with n rows and $|S|$ dimensions. Our goal is to understand the complexity of answering queries, such as which rows occur most frequently in the projected data, computing frequency moments over the rows, and so on. If S is provided prior to seeing the data, then the projection can be performed online, and so many of these tasks reduce to previously studied questions. Hence, we focus on the case when S is decided *after* the data is seen. In particular, we may wish to try out many different choices of S to explore the structure of the subspaces of the data. Our model is given in detail in Chapter 3.2.

For further motivation, we outline some specific areas where such problems arise.

- **Bias and Diversity.** A growing concern in data analysis and machine learning is whether outcomes are 'fair' to different subgroups within the population, or whether they reinforce existing disparities. A starting point for this is to quantify the level of bias within the data when different features are considered. That is, we want to know whether certain combinations of attribute values are over-represented in the data (heavy hitters), and how many different combinations of values are represented in the data (captured by measures like F_0). We would like to be able to answer such queries accurately for many different (typically overlapping) subsets of dimensions.

- **Privacy and Linkability.** When sharing datasets, we seek assurance that they are not vulnerable to attacks that exploit structure in the data to re-identify individuals. An attempt to quantify this risk is given in recent work [CDP+19], which asks how many distinct values occur in the data for each partial identifier, specified as a subset of dimensions. This prior work considered the case where the target dimensions are known in advance, but more generally we would like to compute such measures for arbitrary subsets, based on frequency moments and sampling techniques.

- **Clustering and Frequency Analysis.** In the area of clustering, the notion of subspaces has been studied under a number of interpretations. The common theme is that the data may look unclustered in the original space due to spurious dimensions inflating the distance between points that are otherwise close. Many papers addressed this as a search problem: to search through exponentially many subspaces to find those in which the data is well-clustered. See the survey by Parsons, Haque and Liu [PHL04]. In our setting, the problem would be to estimate various measures of density or clusteredness for a given subspace. A related problem is to find subspaces (or "subcubes" in database terminology) that have high frequency. Prior work proceeded under strong statistical independence assumptions between feature dimensions dimensions, for example, that the distribution can be modeled accurately with a (Naïve) Bayesian model [KMVX18].

3.2 Preliminaries and Definitions

For a positive integer Q, let $[Q] = \{0, 1, \ldots, Q-1\}$, and $\mathbf{A} \in [Q]^{n \times d}$ be the input data. The objective is to keep a summary of $\mathbf{A}$ which is used to estimate the solution to a problem $\mathbf{P}$ upon receiving a column subset query $S \subseteq [d]$. Problems $\mathbf{P}$ of interest are described in Section 3.2.1. Define the restriction of $\mathbf{A}$ to the columns indexed by S as $\mathbf{A}^S$ whose rows A_i^S, $1 \leq i \leq n$, are vectors

over $[Q]^{|S|}$. We use the Minkowski norm $\|\mathbf{X}\|_p = (\sum_{i,j} |\mathbf{X}_{ij}|^p)^{1/p}$ to denote the entrywise-ℓ_p norm for vectors ($j = 1$) and matrices ($j > 1$).

Computational Model. We operate in the projected summary model of Definition 2.1.2. First, the data $\mathbf{A}$ is received under the assumption that it is too large to hold entirely in memory so can be modeled as a stream of data. Our lower bounds are not strongly dependent on the order in which the data is presented. After observing $\mathbf{A}$, a *column query* S is presented. The frequency vector over $\mathbf{A}$ induced by S is $f = f(\mathbf{A}, S)$ whose entries $f_i(\mathbf{A}, S)$ denote the frequency of Q-ary word $\mathbf{w}_i \in [Q]^{|S|}$. We study functions of the frequency vector $f = f(\mathbf{A}, S)$ after the observation of $\mathbf{A}$ and receiving column query S. The task is, during the observation phase, to design a summary of $\mathbf{A}$ which approximates statistics of $\mathbf{A}^S$, the restriction of $\mathbf{A}$ to its projected subspace S. Approximations of $\mathbf{A}^S$ are accessed through the frequency vector $f(\mathbf{A}, S)$. Note that functions (e.g., norms) are taken over $f(\mathbf{A}, S)$ as opposed to the raw vector inputs from the column projection.

Remark 3.2.1 (Indexing Q-ary words into f)**.** *Recall that the frequency vector $f(\mathbf{A}, S)$ has length $Q^{|S|}$ with each entry f_i counting the occurrences of word $\mathbf{w}_i \in [Q]^{|S|}$. To clearly distinguish between the (scalar) index i of f and the input vectors $\mathbf{w}_i$ whose frequency is measured by f_i we introduce the **index function** $e(\mathbf{w}_i) = i$. We may think of $e(\cdot)$ as simply the canonical mapping from $[Q]^{|S|}$ into $\{0, 1, 2, \ldots, Q^S - 1\}$, but other suitable bijections may be used.*

Example 3.2.1. *Suppose $Q = 2$ and $\mathbf{A} \in \{0,1\}^{5\times 3}$ with column indices $\{1, 2, 3\}$ given below. If $S = \{1, 2\}$, then using the canonical mapping from $\{0,1\}^{|S|}$ into $\{0, 1, 2, 3\}$ (e.g $e(00) = 0, e(01) = 1, \ldots e(11) = 3$) we obtain $\mathbf{A}^S$ and hence $f(\mathbf{A}, S) = (1, 1, 0, 3)$ as 00 occurs once, 01 occurs once, and 11 occurs 3 times while 10 never occurs.*

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \qquad \longrightarrow \qquad \mathbf{A}^S = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

The vector $f = f(\mathbf{A}, S)$ is then the frequency vector over which we seek to compute statistical queries such as $\|f\|_0$. In this example, $\|f\|_0 = 3$ (there are three distinct rows in $\mathbf{A}^S$), while $\|f\|_1 = 5$ is independent of the choice of S.

3.2.1 Problem Definitions

The problems that we consider are column-projected forms of common streaming problems ([KNW10, BCI+17, BGL+18]). Here, we refer to these problems

as "projected frequency estimation problems" over the input A. We define

$$f_i(\mathbf{A}, S) = |\{j : \mathbf{A}_j^S = \mathbf{w}_i, j \in [n]\}| \tag{3.1}$$

$$F_p(\mathbf{A}, S) = \sum_{i \in \{0,1\}^{|S|}} f_i(\mathbf{A}, S)^p. \tag{3.2}$$

The quantity $f_i(\mathbf{A}, S)$ should be thought of as counting how many times the pattern (at location) i (under the indexing from $e(\cdot)$) occurs on $\mathbf{A}^S$. Hence, the function $F_p(\mathbf{A}, S)$ then evaluates frequency moments over all of the entries in f.

- F_p **estimation**: Given a column query S, the F_p estimation problem is to approximate the quantity $F_p(\mathbf{A}, S) = \|f(\mathbf{A}, S)\|_p^p$ under some measure of approximation to be specified later (e.g., up to a constant factor). Of particular interest to us is (projected) $F_0(\mathbf{A}, S)$ estimation, which counts the number of distinct row patterns in $\mathbf{A}^S$.

- ℓ_p**-heavy hitters**: The query is specified by a column query $S \subseteq [d]$, a choice of metric/norm $\ell_p, p > 0$ and accuracy parameter $\phi \in (0, 1)$. The task is then to identify all patterns $\mathbf{w}_i$ observed on $\mathbf{A}^S$ for which $f_i(\mathbf{A}, S) \geq \phi \|f(\mathbf{A}, S)\|_p$. Such values $\mathbf{w}_i$ (or equivalently i) are called *ϕ-ℓ_p-heavy hitters*, or simply *ℓ_p-heavy hitters* when ϕ is fixed. We will consider a multiplicative approximation based on a parameter $c > 1$, where we require that all ϕ-ℓ_p heavy hitters are reported, and no items with weight less than $(\phi/c) \cdot \|f(\mathbf{A}, S)\|_p$ are included.

- ℓ_p**-frequency estimation**: A related problem is to allow the frequency $f_i(\mathbf{A}, S)$ to be estimated accurately, with error as a fraction of $F_p(\mathbf{A}, S)^{1/p} = \|f(\mathbf{A}, S)\|_p$, which we refer to as *$\ell_p$ frequency estimation*. Specifically, for a given $\mathbf{w}_i$, return an estimate $\hat{f}_i$ which satisfies $|\hat{f}_i(\mathbf{A}, S) - f_i(\mathbf{A}, S)| \leq \phi \|f(\mathbf{A}, S)\|_p$.

- ℓ_p **sampling**: The goal of this sampling problem is to sample patterns $\mathbf{w}_i$ according to the distribution $p_i \in (1 \pm \varepsilon) \frac{f_i^p(\mathbf{A}, S)}{\|f(\mathbf{A}, S)\|_p^p} + \Delta$ where $\Delta = 1/\operatorname{poly}(nd)$, and return a $(1 \pm \varepsilon')$-approximation to the probability p_i of the item $\mathbf{w}_i$ returned.

Casting our mind back to the advertising example given at the start of the chapter, we can begin to see how the functions described above have applications. For example, after the frequency vector $f(\mathbf{A}, S)$ is built, the ℓ_p-heavy hitters problem is asking for which advert-event user patterns occur most frequently on the projection S. Similarly, Question 3.0.1 is asking for the frequency of a fixed user pattern; if this is relaxed to estimating the frequency of the pattern, then it is an instance of ℓ_p frequency estimation. Question 3.0.2

asks for all patterns observed on a particular $\mathbf{A}^S$ which is an example of projected F_0 estimation. Finally, if the analyst wanted to return a random set of users or user patterns that represents the general population-level behaviour, then they could instantiate this as an instance of projected ℓ_p sampling.

When clear, we may drop the dependence upon S in the notation and write f_i and F_p instead. We will use $\tilde{O}$ and $\tilde{\Omega}$ notation to supress factors that are polylogarithmic in the leading term. For example, lower bounds stated as $\tilde{\Omega}(2^d)$ suppress terms polynomial in d.

3.2.2 Related Work

The model we study is reminscent of, but distinct from, some related formulations. In the problem of cascaded aggregates [JW09], we imagine the starting data as an array, and apply a first operator (denoted Q) on each row to obtain a vector, on which we apply a second operator P. One can think of P as a function such as F_0 which returns the number of nonzeros in every row and populates a new vector with this value in the corresponding location. That is, update $r_i = Q(\mathbf{A}_i)$ for every input row $\mathbf{A}_i$. Following this first aggregation, a second operator P is applied which is evaluated over the row-wise frequency vector r. This is written as $P(r) = P(Q(\mathbf{A}))$. Our problems can be understood as special cases of cascaded aggregates where Q is a project-then-concatenate operator, to obtain a vector whose indices correspond to the concatenation of the projection of a row.

Another example of a cascaded aggregate is a so-called correlated aggregate [TW12], but this was only studied in the context of two dimensions. In this setting, the data is viewed as a 2-dimensional stream (x_i, y_i) for $1 \leq i \leq n$. Then one wishes to evaluate a function f over all values of x_i provided that $y_i \leq c$ for a constant c. Note that this differs from our model as the goal is to compute a function over only a subset of the stream based upon the condition $y_i \leq c$ being satisfied. In contrast, our aim is to compute a function over *all* of the items in the stream which can vary depending on which columns are included in the query. To the best of our knowledge, our projection-based definitions have not been previously studied under the banner of cascaded aggregates.

Other work includes results on provisioning queries for analytics [AKLT16], but the way these statistics are defined is different from our formulation. In that setting there are different scenarios ("hypotheticals") that may or may not be turned on: this corresponds to "what-if" analysis whereby a query is roughly "how many items are observed if a given set of columns is present (turned on)?" The number of distinct elements for the query is the union of the number of distinct elements across scenarios. In our setting, we concatenate the distinct

items into a vector such as r above, and count the number of distinct vectors in r. Note that in the hypotheticals setting in the binary case, each column only has 2 distinct values, 0 and 1, and thus the union also only has 2 distinct values. However, we can obtain up to 2^d distinct vectors. Consequently, Assadi et al. are able to achieve poly (d/ε) space for counting distinct elements, whereas we show a $2^{\Omega(d)}$ lower bound. Moreover, they achieve a $2^{\Omega(d)}$ lower bound for counting (i.e., F_1), whereas we achieve a constant upper bound. These disparities highlight the differences in our models.

More recently, the notion of "subset norms" was introduced by Braverman, Krauthgamer and Yang [BKY18]. This problem considers an input that defines a vector $\mathbf{v}$, where the objective is to take a subset s of entries of $\mathbf{v}$ and compute the norm of $\mathbf{v}$ only over the specified subset of entries. Results are parameterized by the "heavy hitter dimension", which is a measure of complexity over the set system from which s can be drawn. While sharing some properties with our scenario, the results for this model are quite different. In particular, in [BKY18] a trivial upper bound follows by maintaining the vector $\mathbf{v}$ explicitly, of dimension n. Meanwhile, many of our results show lower bounds that are exponential in the dimensionality, as $2^{\Omega(d)}$, though we also obtain non-trivial upper bounds.

3.3 Contributions

The main challenge here is that the column query S is revealed *after* observing the data; consequently, applying a known algorithm to just the columns S as the data arrives is not possible. For example, consider the exemplar problem of counting the number of distinct rows under the projection S, i.e., the projected F_0 problem. Recall that $\mathbf{A}_i^S$ denotes the i-th row of array $\mathbf{A}^S$. Then the task is to count the number of distinct rows observed in $\mathbf{A}^S$, i.e.,

$$F_0(\mathbf{A}, S) = |\{\mathbf{A}_j^S : j \in [n]\}| = \|f(\mathbf{A}, S)\|_0.$$

Observe that $F_0(\mathbf{A}, S)$ can vary widely over different choices of S. For example, even for a binary input $\mathbf{A} \in \{0,1\}^{n \times d}$, $F_0(\mathbf{A}, S)$ can be as large as 2^d when S consists of all columns from a highly diverse dataset, and as small as 1 or 2 when S is a single column or when S selects homogeneous columns (e.g., the columns in S are all zeros).

3.3.1 Summary of Results

Our main focus, in common with prior work on streaming algorithms, is on space complexity. For the above problems we obtain the following results:

- In Section 3.4 we show that projected F_0 estimation requires $2^{\Omega(d)}$ space for a constant factor approximation, demonstrating the essential hardness of these problems. Nevertheless, we obtain a tradeoff in terms of upper bounds described below.

- Section 3.5 presents results for ℓ_p frequency estimation, ℓ_p heavy hitters, F_p estimation, and ℓ_p sampling. We show a space upper bound of $O(\varepsilon^{-2}\log(1/\delta))$ for ℓ_p frequency estimation when $0 < p < 1$ and complement this result with lower bounds for heavy hitters when $p > 1$, F_p estimation and ℓ_p sampling for all $p \neq 1$, showing that these problems require $2^{\Omega(d)}$ bits of space.

- In Section 3.6 we show upper bounds for F_0 and F_p estimation which improve on the exhaustive approach of keeping summaries of all 2^d subsets of columns, by showing that we can obtain coarse approximate answers with a smaller subset of materialized answers. Specifically, for parameters $N = 2^d$ and $\alpha \in (0, 1)$ we can obtain an N^α approximation in $\min\left(N^{H(1/2-\alpha)}, n\right)$ space. Since the binary entropy function $H(x) < 1$, this bound is better than the trivial 2^d bound.

These bounds show that there is no possibility of "super efficient" solutions that use space less than exponential in d. Nevertheless, we demonstrate some solutions whose dependence is still exponential but weaker than a naïve 2^d. Thinking of $N = 2^d$, the above upper and lower bounds imply the actual complexity is a nontrivial polynomial function of N.

The bounds also show novel dichotomies that are not present in comparable problems without projection. In particular, we show that (projected) ℓ_p sampling is difficult for $p \neq 1$ while (projected) ℓ_p-heavy hitters has a small space algorithm for $0 < p < 1$. This differs from the standard streaming model in which the (classical) ℓ_p heavy hitters problem has a small space solution for $p \leq 2$ without projection [LNNT16], and (classical) ℓ_p sampling can be performed efficiently for $p \leq 2$ [JW18]. Our lower bounds are built on amplifying the frequency of target codewords for a carefully chosen test word.

Note that there are trivial naïve solutions which simply retain the entire input and so answer the query exactly on the query S: to do so takes $\Theta(nd)$ space, noting that n may be exponential in d. Alternatively, if we know $t = |S|$ then we may enumerate all $\binom{d}{t}$ subsets of $[d]$ with size t and maintain (approximate) summaries for each choice of S. However, this will entail a cost of at least $\Omega(d^t)$ and as such does not give a major reduction in cost.

3.3.2 Coding Theory Definitions

Our lower bounds will typically make use of a *binary* code $\mathcal{C}$, constituted of a collection of *codewords*, which are vectors (or strings) of fixed length. We write $\mathcal{B}(l,k)$ to denote all binary strings of length l and (Hamming) weight k. We first consider the dense, low-distance family of codes $\mathcal{C} = \mathcal{B}(d,k)$ but will later use more sophisticated randomly sampled codes. When $k < d/2$, we have $\binom{d}{k} \geq (d/k)^k$ and when $k = d/2$, we have $\binom{d}{d/2} \geq 2^d/\sqrt{2d}$. A trivial but crucial property of $\mathcal{B}(d,k)$ is that any two codewords from this set can have intersecting 1s in at most $k-1$ positions.

We define the *support of a string* $\mathbf{x}$ as $\text{supp}(\mathbf{x}) = \{i : \mathbf{x}_i \neq 0\}$, the set of locations where $\mathbf{x}$ is non-zero. We define *child words* to be the set of new codewords obtained from $\mathcal{C}$ by generating all Q-ary words $\mathbf{z}$ with $\text{supp}(\mathbf{z}) \subseteq \text{supp}(\mathbf{x})$ for some $\mathbf{x} \in \mathcal{C}$, and construct them with the star operator defined next. The purpose of star^Q is to amplify the incidence and frequency of strings in order to construct a hard instance for the lower bound.

Definition 3.3.1 (star^Q operation, child words)**.** *Let d be the length of a binary word, k be a weight parameter, and suppose $y \in \mathcal{B}(d,k)$. Let $M = \text{supp}(\mathbf{x})$. We define the function $\mathsf{star}^Q(\mathbf{x})$ to be the operation which lifts a binary word $\mathbf{x}$ to a larger alphabet by generating all the words over alphabet $[Q]$ on M. Formally,*

$$\mathsf{star}^Q(\mathbf{x} \in \{0,1\}^d) = \{\mathbf{z} : \mathbf{z} \in [Q]^d, \text{supp}(\mathbf{z}) \subseteq \text{supp}(\mathbf{x})\}$$

Since the alphabet size Q is often fixed when using this operation, when clear we will drop the superscript and abuse notation by writing $\mathsf{star}(\mathbf{x})$. Elements of the set $\mathsf{star}^Q(\mathbf{x})$ are referred to as child words of $\mathbf{x}$.

Example 3.3.1 (Action of $\mathsf{star}^Q(\cdot)$)**.** *Suppose the string length is $d = 4$ and the weight is $k = 2$. Consider $\mathcal{C} = \mathcal{B}(4,2)$ which is the set of all binary strings of length 4 and weight 2. If we take $\mathbf{w} = (0,0,1,1)$ and $Q = 3$ we generate the following child words of $\mathbf{w}$:*

$$\mathcal{C} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \longrightarrow \mathsf{star}^Q(\mathbf{w}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 2 & 2 \end{bmatrix}.$$

For any $y \in \mathcal{B}(d,k)$, there are Q^k words generated by $\mathsf{star}^Q(\mathbf{x})$. When $\mathsf{star}(\cdot)$

is applied to all vectors of a set U then we write $\mathsf{star}(U) = \cup_{\mathbf{u}\in U}\mathsf{star}(\mathbf{u})$. For example, if $\mathbf{x} \in \{0,1\}^d$ and $Q = 2$, then $\mathsf{star}^Q(\mathbf{x})$ is simply all possible binary words of length d whose support is contained in $\mathrm{supp}(\mathbf{x})$. For the projected F_0 problem, the code $\mathcal{C} = \mathcal{B}(d,k)$ is sufficient. However, for our subsequent results, we need a randomly chosen code whose existence is demonstrated in Lemma 3.3.1. The proof follows from a Chernoff bound.

Lemma 3.3.1. *Fix $\varepsilon, \gamma \in (0,1)$ and let $\mathcal{C} \subseteq \mathcal{B}(d, \varepsilon d)$ be such that for any two distinct $\mathbf{x}, \mathbf{y} \in \mathcal{C}$ we have $|\mathbf{x} \cap \mathbf{y}| \leq (\varepsilon^2 + \gamma)d$. With probability at least $1 - \exp(-2d\gamma^2)$ there exists such a code $\mathcal{C}$ with size $2^{O(\gamma^2 d)}$ instantiated by sampling sufficiently many words i.i.d. at random from $\mathcal{B}(d, \varepsilon d)$.*

Proof. Let Z be the random variable for the number of 1s in common between $\mathbf{x}$ and $\mathbf{y}$ sampled uniformly at random. Then the expectation of Z is $\mathbb{E}[Z] = \frac{(\varepsilon d)^2}{d} = \varepsilon^2 d$ and although the coordinates of $\mathbf{x}, \mathbf{y}$ are not independent, they are negatively correlated. We claim that $\mathbf{z} = (Z_1, Z_2, \ldots, Z_d)$ with $Z_i = X_i Y_i$ also inherits this negative correlation over its coordinates. We have $\mathbb{E}(Z_i Z_j) = \mathbb{E}(X_i Y_i X_j Y_j)$. By reordering and using the independence of X_i and Y_i we have $\mathbb{E}(Z_i Z_j) = \mathbb{E}(X_i X_j)\mathbb{E}(Y_i Y_j)$ upon which we may use the negative correlation amongst the $X_\cdot$ and $Y_\cdot$ and reorder to obtain $\mathbb{E}(Z_i Z_j) \leq \mathbb{E}(Z_i)\mathbb{E}(Z_j)$. Hence the $(Z_1, Z_2, \ldots, Z_d)$ are negatively correlated and such random variables obey Chernoff bounds (see Section 1.10.2 of [Doe20] for self-contained details). Our aim is to show that the number of 1s in common between $\mathbf{x}$ and $\mathbf{y}$ can be at most γd more than its expectation. Then, via an additive Chernoff-Hoeffding bound:

$$\mathbb{P}(Z - \mathbb{E}(Z) \geq \gamma d) \leq \exp(-2d\gamma^2).$$

This is the probability that any two codewords $\mathbf{x}$ and $\mathbf{y}$ are not too similar, so by taking a union bound over the $\Theta(|\mathcal{C}|^2)$ pairs of codewords, the size of the code is $|\mathcal{C}| = \exp(d\gamma^2) = 2^{\gamma^2 d/\ln 2}$. □

3.3.3 Overview of Lower Bound Constructions

Our lower bounds rely upon non-standard reductions to the Index problem using codes $\mathcal{C}$ defined in Section 3.3.2. These reductions are more involved than is typically found as we need to combine the combinatorial properties of $\mathcal{C}$ along with the $\mathsf{star}(\cdot)$ operation on Alice's input. In particular, the interplay between $\mathcal{C}$ and $\mathsf{star}(\cdot)$ must be understood over the column query S given by Bob, which again relies on properties of $\mathcal{C}$ used to define the input.

Recall that the typical reduction from Index is as follows: Alice holds a vector $\mathbf{a} \in \{0,1\}^N$, Bob holds an index $i \in [N]$ and he is tasked with finding $\mathbf{a}_i$ following one-way communication from Alice. The randomized communication complexity of Index is $\Omega(N)$ [KNR99]. We adapt this setup for our family

of problems, following an approach that has been used to prove many space lower bounds for streaming algorithms. Here the gadgets we need to reduce from the Index problem are highly non-trivial. The technical details of the lower bounds can be quite involved, so before progressing, we first present a high-level schematic of the different steps in the lower bound arguments.

Instantiating The Lower Bound Argument: Figure 3.1

First we choose a binary code $\mathcal{C}$ (usually independently at random as in Lemma 3.3.1) with certain properties such as a specific weight and a bounded number of 1s in common locations with other words in the code. In the communication setting, Alice holds a subset $T \subseteq \mathcal{C}$ while Bob holds a codeword $\mathbf{y} \in \mathcal{C}$ and is tasked with determining whether or not $\mathbf{y} \in T$. The corresponding bitstring for the Index problem that Alice holds is $\mathbf{a} \in \{0,1\}^{|\mathcal{C}|}$ which has $\mathbf{a}_j = 1$ for every element $\mathbf{w}_j \in T$ using the index function to map strings to indices (Remark 3.2.1) in $\mathbf{a}$. In comparison to the standard setting mentioned above, Alice's vector is of length $N = |\mathcal{C}|$ which our constructions establish is exponentially large in d.

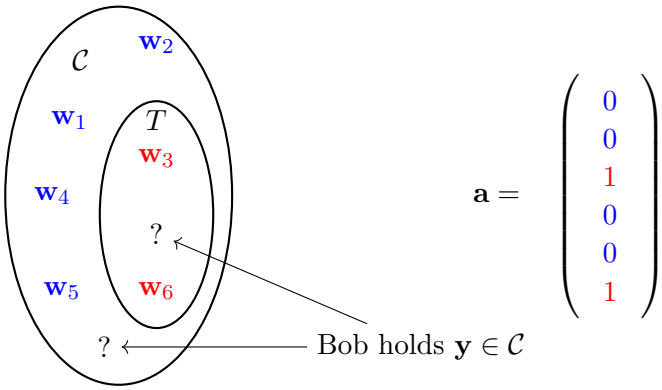


Figure 3.1: Alice holds a subset T of the code $\mathcal{C} = \{\mathbf{w}_1, \ldots, \mathbf{w}_6\}$, using that to populate her membership vector $\mathbf{a}$. Bob queries algorithm $\mathcal{A}$ with his test vector $\mathbf{y}$ and a column set S. The algorithm returns `estimate` which Bob compares to some threshold `thld`. If `estimate` $\geq$ `thld` then Bob knows $\mathbf{y} \in T$ and can return 1 as $\mathbf{a}_{e(\mathbf{w})} = 1$ for all $\mathbf{w} \in T$. Otherwise, Bob reports 0 as Alice's vector $\mathbf{a}_{e(\mathbf{w})} = 0$ for all $\mathbf{w} \in \mathcal{C} \setminus T$.

Bob can also access the index function $e(\mathbf{y})$ which returns the index or location that $\mathbf{y}$ is enumerated in $\mathcal{C}$. His task is to answer the question *is* $\mathbf{y} \in T$ *or is* $\mathbf{y} \in \mathcal{C} \setminus T$*?* This would enable Bob to return 1 if $\mathbf{y} \in T$, or 0 otherwise, which is the value of $\mathbf{a}_{e(\mathbf{y})}$, meaning that Bob can solve the Index problem.

Generating A Hard Instance Using $\mathsf{star}^Q(T)$: Figure 3.2

We use the $\mathsf{star}^Q(T)$ operator (Definition 3.3.1 and example 3.3.1) to map strings into an input $\mathbf{A}$ for each of the problems (i.e., a collection of rows

$$\mathbf{w} \xrightarrow{\mathsf{star}^Q(\mathbf{w})} \begin{pmatrix} \{\mathsf{star}^Q(\mathbf{w})\}_1 \\ \{\mathsf{star}^Q(\mathbf{w})\}_2 \\ \{\mathsf{star}^Q(\mathbf{w})\}_3 \\ \vdots \\ \{\mathsf{star}^Q(\mathbf{w})\}_m \end{pmatrix}$$

Figure 3.2: Alice maps her word $\mathbf{w} \in T$ to a larger set of *child words* $\mathsf{star}^Q(\mathbf{w})$. For a code $\mathcal{C}$ with weight k, the number of words in $\mathsf{star}^Q(\mathbf{w})$ is Q^k. Alice will concatenate $\mathsf{star}^Q(\mathbf{w})$ for every $\mathbf{w} \in T$ to generate the input data $\mathbf{A}$.

of datapoints). The $\mathsf{star}^Q(T)$ function is necessary to 'diversify' the possible representations of a particular word. Roughly speaking, we would like a frequency-type function to be 'large' if $\mathbf{y} \in T$ and 'small' otherwise (or vice versa). Using $\mathsf{star}^Q(T)$ boosts the frequency of certain patterns in such a way that we can control to reason in which of the two cases Bob is in. Alice evaluates $\mathsf{star}^Q(\mathbf{w})$ for every $\mathbf{w} \in T$ that she holds.

Example 3.3.2. *Recall Example 3.3.1 with $\mathcal{C} = \mathcal{B}(4,2)$ and $Q = 3$. Suppose Alice's set T is $\{\mathbf{w}_1, \mathbf{w}_2\} = \{0011, 0101\}$. Then the associated array which is used as input to a later algorithm is (absence of ellipsis marks all zeros column):*

$$\mathbf{A} = \begin{bmatrix} - & \mathit{star}^3(\mathbf{w}_1) & - \\ - & \mathit{star}^3(\mathbf{w}_2) & - \end{bmatrix} \longrightarrow \mathbf{A} = \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ & & \vdots & \vdots \\ 0 & 0 & 2 & 2 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ & \vdots & & \vdots \\ 0 & 2 & 0 & 2 \end{array}\right] \in [3]^{2 \cdot 3^2 \times 4}.$$

Bob's Approach

Suppose that for a chosen problem we claim algorithm $\mathcal{A}$ achieves a certain approximation factor, for example, a constant factor approximation. Bob holds the vector $\mathbf{y}$ and chooses a column query S based upon $\mathbf{y}$; typically, this will be $\mathrm{supp}(\mathbf{y})$. Algorithm $\mathcal{A}$ is executed with the input array $\mathbf{A}$, column query S and returns an estimate `estimate`. Bob compares `estimate` to a threshold value `thld` which is a function of the input parameters. Should `estimate` be larger than the threshold, Bob is assured that $\mathbf{y}$ was present on the input $\mathbf{A}$ so can happily return 1, otherwise returning 0.

Example 3.3.3. *We illustrate how Bob would use algorithm $\mathcal{A}$ for the projected F_0 problem. Suppose as in Examples 3.3.1 and 3.3.2 that $\mathcal{C} = \mathcal{B}(4,2), Q = 3$ & $T = \{\mathbf{w}_1, \mathbf{w}_2\}$ is Alice's set. Consider $\mathbf{y} \in T$ and assume $\mathbf{y} = \mathbf{w}_1$ so $\mathbf{y} = (0011), \mathrm{supp}(\mathbf{y}) = \{3,4\}$ which defines $\mathbf{A}^S$ as in (3.4). Bob queries $\mathcal{A}$ over the stream of length-2 strings in $\mathbf{A}^S$ used to build the frequency vector $f(\mathbf{A}, S)$. Bob uses the index function $e(00) = 0, e(01) = 1, \ldots, e(22) = 8$ so $\mathbf{A}^S$ implicitly defines the following stream and frequency vector*

$$\begin{aligned} \mathsf{stream}(\mathbf{A}, S) &= (0,1,2,3,4,5,6,7,8,0,1,2,0,1,2,0,1,2) \\ f(\mathbf{A}, S) &= (4,4,4,1,1,1,1,1,1). \end{aligned}$$

On the other hand, if $\mathbf{y} = (0110) \notin T$, then Bob's column query is $S' = \{2,3\}$, yielding $\mathbf{A}^{S'}$ as shown in (3.4). Now the patterns that Bob observes define the stream and frequency vector in (3.3). We recognise that $\|f(\mathbf{A}, S)\|_0 = 9$ whereas $\|f(\mathbf{A}, S')\|_0 = 3$.

$$\begin{aligned} \mathsf{stream}(\mathbf{A}, S') &= (0,0,0,1,1,1,2,2,2,0,0,0,1,1,1,2,2,2) \\ f(\mathbf{A}, S') &= (6,6,6,0,0,0,0,0,0). \end{aligned} \tag{3.3}$$

$$\mathbf{A}^S = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 2 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 2 & 0 \\ 2 & 1 \\ 2 & 2 \\ 0 & 0 \\ 0 & 1 \\ 0 & 2 \\ 0 & 0 \\ 0 & 1 \\ 0 & 2 \\ 0 & 0 \\ 0 & 1 \\ 0 & 2 \end{bmatrix} \qquad \mathbf{A}^{S'} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 2 \\ 0 & 2 \\ 0 & 2 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 2 \\ 0 & 2 \\ 0 & 2 \end{bmatrix}. \tag{3.4}$$

The gap between the two cases $\mathbf{y} \in T$ & $\mathbf{y} \notin T$ is controlled through the weight k of $\mathcal{C}$ along with the child word alphabet Q. The main property we use is that the number of patterns observed after projection is proportional to the number of 1s located in the same place between $\mathbf{y}$ & members of Alice's set T.

3.4 Lower Bounds for F_0

In this section, we focus on the F_0 (distinct counting) projected frequency problem. The main result in this section is a strong lower bound for the problem, which is exponential in the domain size d. We use codes $\mathcal{C} = \mathcal{B}(d, k)$ as defined in Section 3.3.2. We generate a hard instance and reduce to Index which gives an information-theoretic lower bound on the necessary space.

Theorem 3.4.1 and its corollaries follow the skeleton argument outlined in Section 3.3.3. First, Alice populates her vector $\mathbf{a}$ based upon her held set $T \subset \mathcal{C}$. Alice's input is expanded into the highly diverse set of child words $\mathsf{star}^Q(T)$ with every word from this set representing a row of the input array $\mathbf{A}$. Bob takes a test vector $\mathbf{y} \in \mathcal{C}$ and will query the projected algorithm with $S = \text{supp}(\mathbf{y})$. We will argue that if Alice holds $\mathbf{y}$, then she must have included $\mathsf{star}^Q(\mathbf{y})$ into $\mathbf{A}$, hence Bob observes at least all of the Q^k different patterns over $[Q]^k$ so the estimate is 'large'. On the other hand, we also show that if Alice does not hold $\mathbf{y}$ the algorithm returns an estimate for the projected F_0 problem which is 'small', being bounded above by kQ^{k-1}. This separation allows Bob to solve Index, assuming the existence of an appropriate approximation algorithm.

Theorem 3.4.1. *Let $Q \geq 2$ be the target alphabet size and $k < d/2$ be a fixed query size with $Q > k$. Any algorithm achieving an approximation factor of $|Q|/k$ for the projected F_0 problem requires space $2^{\Omega(d)}$.*

Proof. Fix the code $\mathcal{C} = \mathcal{B}(d, k)$, recalling that any $\mathbf{x} \in \mathcal{C}$ has Hamming weight k, and for distinct $\mathbf{x}, \mathbf{y} \in \mathcal{C}$ at most $k - 1$ bits are shared in common. We will use these facts to obtain the approximation factor.

We will reduce from the Index problem in communication complexity as follows. Alice has a set of (binary) codewords $T \subseteq \mathcal{C}$ and initializes the input array $\mathbf{A}$ for the algorithm with all strings from the set $\mathsf{star}^Q(T)$. Bob has a vector $\mathbf{y} \in \mathcal{C}$ and wants to know if $\mathbf{y} \in T$ or not. Let $S = \text{supp}(\mathbf{y})$ so that $|S| = k$ and Bob queries the F_0 algorithm on columns of $\mathbf{A}$ restricted to S. First suppose that $\mathbf{y} \in T$. Then Alice holds $\mathbf{y}$ so $\mathsf{star}(\mathbf{y})$ is included in $\mathbf{A}$ and there must be at least Q^k patterns observed. Conversely, if $\mathbf{y} \notin T$, then Alice does not include $\mathbf{y}$ in $\mathbf{A}$. However, by the construction of $\mathcal{C}$, $\mathbf{y}$ shares at most $(k - 1)$ 1s with any distinct $\mathbf{y}' \in \mathcal{C}$. Thus, the number of patterns observed on the columns corresponding to S is at most $\binom{k}{k-1}Q^{k-1} = kQ^{k-1}$.

We observe that if we can distinguish the case of kQ^{k-1} from Q^k, then we could correctly answer the Index instance, i.e., if we can achieve an approximation factor of Δ such that:

$$\Delta = \frac{Q^k}{kQ^{k-1}} = \frac{Q}{k}. \tag{3.5}$$

Any protocol for Index requires communication proportional to the length

of Alice's input vector $\mathbf{a}$, which translates into a space lower bound for our problem. Alice's set $T \subset \mathcal{C}$ defines an input vector for the Index problem built using a characteristic vector over all words in $\mathcal{C}$, denoted by $\mathbf{a} \in \{0,1\}^{|\mathcal{C}|}$, as follows. Under a suitable enumeration of $\mathcal{C} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{|\mathcal{C}|}\}$, Alice's vector is encoded via $\mathbf{a}_i = 1$ if and only if Alice holds the binary word $\mathbf{w}_i \in T$. From the separation shown earlier, Bob can determine if Alice holds a word in T, thus solving Index and incurring the lower bound. Hence, space proportional to $|\mathcal{C}| = \binom{d}{k}$ is necessary. We use the standard relation $\binom{d}{k} \geq (d/k)^k$ and choose $k = ad/2$ for a constant $a \in [0,1)$ from which we obtain $|\mathcal{C}| \geq 2^{ad/2}$ to achieve the stated approximation guarantee. □

Setting $k = ad/2$ allows us to vary the query size and directly understand how this affects the size of the code necessary for the lower bound. For a query of size k, the input array $\mathbf{A}$ to the projected F_0 problem is an array whose rows are words contained in $\mathsf{star}^Q(T)$, hence $\mathbf{A}$ has size $|T|Q^k \times d$. Theorem 3.4.1 is for $k < d/2$. When $k = d/2$ we can use the tighter bound for the central binomial term on the sum of the binomial coefficients and obtain the following stronger bounds. The subsequent results use the same encoding as in Theorem 3.4.1. However, at certain points of the calculations the parameter setttings are slightly altered to obtain different guarantees.

Corollary 3.4.1. *Let $Q \geq d/2$ be an alphabet size and $d/2$ be the query size. There exists a choice of input data $\mathbf{A} \in [Q]^{n \times d}$ such that any algorithm achieving approximation factor $2Q/d$ for the projected F_0 problem on the query requires space $2^{\Omega(d)}$.*

Proof. Repeat the argument of Theorem 3.4.1 with $k = d/2$. The approximation factor from Equation (3.5) becomes: $\Delta = {}^{2Q}/_{d}$. The code size for Index is $|\mathcal{C}| \geq 2^d/\sqrt{2d}$. Note that $|\mathcal{C}|$ is $2^{\Omega(d)}$ as $\frac{1}{2}\log_2(d)$ can always be bounded above by a linear function of d. The instance is an array whose rows are the $Q^{d/2}$ child words in $\mathsf{star}^Q(T)$. Hence, the size of the instance to the F_0 algorithm is bounded above by $|T|Q^{d/2} \times d$. □

Corollary 3.4.2 follows from Corollary 3.4.1 by setting $Q = d$.

Corollary 3.4.2. *A 2-factor approximation to the projected F_0 problem on a query of size $d/2$ needs space $2^{\Omega(d)}$ with an instance $\mathbf{A}$ whose size is $|T|Q^{d/2} \times d$.*

Theorem 3.4.1 and its corollaries suffice to obtain space bounds over all choices of Q. However, Q could potentially grow to be very large, which may be unsatisfying. As a result, we will argue how the error varies for fixed Q. To do so, we map Q down to a smaller alphabet of size q and use this code to define the communication problem from which the lower bound will follow. The cost of this is that the instance is a logarithmic factor larger in the dimensionality.

Table 3.1: Comparison of parameter settings for projected F_0 lower bounds. Theorem 3.4.1 uses $\mathcal{C} = \mathcal{B}(d, k)$, corollaries use $\mathcal{C} = \mathcal{B}(d, d/2)$. Alice's set $T \subset \mathcal{C}$ so is no larger than 2^d, so we can always upper bound the size of the instance required for the lower bounds.

	Instance $\mathbf{A}$ for F_0	Approx. Factor
Theorem 3.4.1	$\lvert T\rvert Q^k \times d$ over $[Q]$	Q/k
Corollary 3.4.1	$\lvert T\rvert Q^{d/2} \times d$ over $[Q]$	$2Q/d$
Corollary 3.4.2	$\lvert T\rvert Q^{d/2} \times d$ over $[d]$	2
Corollary 3.4.3	$\lvert T\rvert Q^{d/2} \times d\log_q Q$ over $[q]$	$2Q/d$

Corollary 3.4.3. *Let q be a target alphabet size such that $2 \leq q \leq Q$. Let $\alpha = Q\log_q(Q) \geq 1$ and $d' = d\log_q(Q)$. There exists a choice of input data $\mathbf{A} \in [q]^{n\times d'}$ for which any algorithm for the projected F_0 problem over queries of size $d/2$ that guarantees error $\tilde{O}(\alpha/d')$ requires space $2^{\Omega(d)}$.*

Proof. Fix the binary code $\mathcal{C} = \mathcal{B}(d, d/2)$, Alice's set $T \subset \mathcal{C}$ and generate all child words $\mathsf{star}^Q(T)$ over alphabet $[Q]$ to obtain the approximation factor $\Delta = 2Q/d$ as in Corollary 3.4.2. For every $\mathbf{w} \in T$ there are $Q^{d/2}$ child words so $\mathsf{star}^Q(T)$ has size $n = |T|Q^{d/2}$ words. Since Q can be arbitrarily large, we encode it via a mapping to a smaller alphabet but over a slightly larger dimension; specifically, use a function $[Q] \mapsto [q]^{\log_q(Q)}$ which generates q-ary strings for each symbol in $[Q]$. Hence, all of the stored strings in $\mathsf{star}^Q(T) \subset [Q]^d$ are equivalent to a collection, $\mathcal{C}_q$ over $[q]^{d\log_q(Q)}$. Although $|\mathsf{star}^Q(T)| = |\mathcal{C}_q|$, words in $\mathsf{star}^Q(T)$ are length d, while the equivalent word in $\mathcal{C}_q$ has length $d\log_q(Q)$. This multiset of words from $\mathcal{C}_q$ now defines the instance $\mathbf{A} \in [q]^{n\times d\log_q(Q)}$, each word being a row of $\mathbf{A}$. Taking $\alpha = Q\log_q(Q)$ and $d' = d\log_q(Q)$ results in an approximation factor of:

$$\Delta = \frac{2Q}{d} = \frac{2\alpha}{d'}. \tag{3.6}$$

Alice's input vector $\mathbf{a}$ is defined by the same code $\mathcal{C}$ and held set $T \subset \mathcal{C}$ as in Theorem 3.4.1 so we incur the same space bound. Likewise, Bob's test vector $\mathbf{y}$ and column query S also remain the same as in that theorem.

□

Corollary 3.4.3 says that the same accuracy guarantee as Corollary 3.4.1 can be given by reducing the arbitrarily large alphabet $[Q]$ to a smaller one over $[q]$. However, the price to pay for this is that the size of the instance $\mathbf{A}$ increases by a factor of $\log_q(Q)$ in the dimensionality. These various results are summarized in Table 3.1.

3.5 ℓ_p-Frequency Based Problems

In this section, we extend the techniques from the previous section to understand the complexity of projected frequency estimation problems related to the ℓ_p norms and F_p frequency moments (defined in Section 3.2.1). A number of our results are lower bounds, but we begin with a simple sampling-based upper bound to set the stage.

3.5.1 ℓ_p Frequency Estimation

We first focus on the projected frequency estimation problem showing that a simple algorithm keeping a uniform sample of the rows works for $p < 1$. The algorithm $\texttt{uSample}(\mathbf{A}, S, t, b)$ first builds a uniform sample of t rows (sampled with replacement at rate $\alpha = t/n$) from $\mathbf{A}$ and evaluates the absolute frequency of string b on the sample after projection onto S. Let g be the absolute frequency of pattern $\mathbf{b}$ on the subsample. To estimate the true frequency of $\mathbf{b}$ on the entire dataset from the subsample, we return an appropriately scaled estimator $\hat{f}_{e(\mathbf{b})} = g/\alpha$ which meets the required bounds given in Theorem 3.5.1, recalling the $e(\mathbf{b})$ is the index location associated with the string $\mathbf{b}$. The proof follows by a standard Chernoff bound argument.

Theorem 3.5.1. *Let $\mathbf{A} \in \{0,1\}^{n\times d}$ be the input data and let $S \subseteq [d]$ be a given column query. For a given string $\mathbf{b} \in \{0,1\}^{|S|}$, the absolute frequency of $\mathbf{b}$, $f_{e(\mathbf{b})}$, can be estimated up to $\varepsilon\|f\|_1$ additive error using a uniform sample of size $O(\varepsilon^{-2}\log(1/\delta))$ with probability at least $1-\delta$.*

Proof. Let $T = \{i \in [n] : \mathbf{A}_i^S = \mathbf{b}\}$ be the set of indices on which the projection onto query set S is equal to the given pattern $\mathbf{b}$. Sample t rows of $\mathbf{A}$ uniformly with replacement at a rate $q = t/n$. Let the (multi)-subset of indices of rows sampled be denoted by B and the array formed from the rows whose index is in B be denoted $\hat{\mathbf{A}}$. For every $i \in B$, define the indicator random variable X_i which is 1 if and only if the randomly sampled index i satisfies $\mathbf{A}_i^S = \mathbf{b}$, which occurs with probability $|T|/n$. Next, we define $\hat{T} = T \cap B$ so that $|\hat{T}| = \sum_{i=1}^{t} X_i$ and the estimator $Z = \frac{n}{t}|\hat{T}|$ has $\mathbb{E}(Z) = |T|$. Finally, apply an additive form of the Chernoff bound:

$$\begin{aligned}
\mathbb{P}\left(|Z - \mathbb{E}(Z)| \geq \varepsilon n\right) &= \mathbb{P}\left(\left|\frac{n}{t}|\hat{T}| - |T|\right| \geq \varepsilon n\right) \\
&= \mathbb{P}\left(\left||\hat{T}| - \frac{t}{n}|T|\right| \geq \varepsilon t\right) \\
&\leq 2\exp\left(-\varepsilon^2 t\right).
\end{aligned}$$

Setting $\delta = 2\exp\left(-\varepsilon^2 t\right)$ allows us to choose $t = O(\varepsilon^{-2}\log(1/\delta))$, which is independent of n and d. The final bound comes from observing that $\|f\|_1 =$

$n, f_{e(b)} = |T|$ and $\hat{f}_{e(b)} = Z$. □

The same algorithm can be used to obtain bounds for all $0 < p < 1$. By noting that $\|f\|_1 \leq \|f\|_p$ for $0 < p < 1$ we can obtain the following corollary.

Corollary 3.5.1. *Let $\mathbf{A}, \mathbf{b}, S$ be as in Theorem 3.5.1. Let $0 < p < 1$. Then uniformly sampling $O(\varepsilon^{-2}\log(1/\delta))$ rows achieves $\left|\hat{f}_{e(\mathbf{b})} - f_{e(\mathbf{b})}\right| \leq \varepsilon\|f\|_p$ with probability at least $1 - \delta$.*

Both Theorem 3.5.1 and Corollary 3.5.1 are stated as if S is given. However, since the sampling did not rely on S in any way, we can sample complete rows of the input uniformly prior to receiving the query S, which is revealed after observing the data. The uniform sampling approach also allows us to identify the ℓ_p heavy hitters in small space: for each item included in the sample (when projected onto column set S), we use the sample to estimate its frequency, and declare those with high enough estimated frequency to be the heavy hitters. By contrast, for $p > 1$ we are able to obtain a $2^{\Omega(d)}$ space lower bound, given in the next section.

3.5.2 ℓ_p Heavy Hitters Lower Bound

Recall that the objective of (projected) ℓ_p heavy hitters is to find all those rows in $\mathbf{A}^S$ whose frequency is at least some fraction of the ℓ_p norm of the frequency distribution of this projection. For the lower bound we need a randomly sampled code as defined in Lemma 3.3.1. The lower bound argument follows a similar outline to that in Section 3.3.3 and for the projected F_0 problem. However, the key difference now is that Bob's query is on the *complement of the support* of his test vector $\mathbf{y}$ (i.e., $S = [d] \setminus \text{supp}(\mathbf{y})$) rather than $\text{supp}(y)$.

Akin to Theorem 3.4.1, we will create a reduction from the Index problem in communication complexity, and use its communication lower bound to argue a space lower bound for projected ℓ_p heavy hitters. The proof will generate an instance of ℓ_p heavy hitters based on encoding a collection of codewords, and consider in particular the status of the string corresponding to all zeroes. We will consider two cases: when Bob's query string is represented in Alice's set of codewords, then the all zeros string will be a heavy hitter (for a subset of columns determined by the query); and when Bob's string is not in the set, then the all zeros string will not be a heavy hitter. We begin by setting up the encoding of the input to the Index instance.

Theorem 3.5.2. *Let $\phi \in (0, 1)$ be a parameter and fix $p > 1$. Any algorithm which can obtain a constant factor approximation to the projected ℓ_p-heavy hitters problem requires space $2^{\Omega(d)}$.*

Proof. Fix $\varepsilon > 0$. Let $\mathcal{C} \subset \mathcal{B}(d, \varepsilon d)$ be a code whose words have weight εd and any two distinct words $\mathbf{x}, \mathbf{y}$ have at most $(\varepsilon^2 + \gamma)d$ ones in common. By Lemma 3.3.1 such a $\mathcal{C}$ exists and $|\mathcal{C}| = 2^{\Omega_\gamma(d)}$.

Suppose Alice holds a subset $T \subset \mathcal{C}$. Let $\mathbf{a} \in \{0,1\}^{|\mathcal{C}|}$ be the characteristic vector over all length-d binary strings for which $\mathbf{a}_{e(\mathbf{u})} = 1$ if and only if Alice holds $\mathbf{u} \in T$. Bob holds $\mathbf{y} \in \mathcal{C}$ and wants to determine if Alice holds $\mathbf{y} \in T$. Ascertaining whether or not Alice holds $\mathbf{y}$ would be sufficient for Bob to solve Index and incur the $\Omega(|\mathcal{C}|)$ lower bound. We will study the frequency of the vector $\mathbf{0}_S$ which is the all zero vector on columns S.

The input array, $\mathbf{A}$, for the ℓ_p-heavy hitters problem is constructed as follows.

1. Alice populates $\mathbf{A}$ with $2^{\varepsilon d}$ copies of the length-d all ones vector, $\mathbf{1}_d$
2. Next, Alice takes $Q = 2$ and inserts into $\mathbf{A}$ the collection $\mathsf{star}^Q(T)$, which is the expansion of her input strings in T to all child-words in binary. That is, for every $\mathbf{s} \in T$, Alice computes all binary strings $\mathbf{x}$ of length d with $\mathrm{supp}(\mathbf{x}) \subseteq \mathrm{supp}(\mathbf{s})$ and includes these in $\mathbf{A}$.

Case 1: $\mathbf{y} \in T$. If $\mathbf{y} \in T$, then we claim that $\mathbf{0}_S$ is a ϕ-ℓ_p heavy hitter for some constant ϕ, i.e., $f_{e(\mathbf{0}_S)} \geq \phi \|f\|_p$. We will manipulate the equivalent condition $f^p_{e(\mathbf{0}_S)} \geq \phi^p F_p$. Since $\mathbf{y} \in T$, the set $\mathsf{star}(\mathbf{y})$ is included in the table $\mathbf{A}$ as Alice inserted $\mathsf{star}(\mathbf{s})$ for every $\mathbf{s}$ that she holds. Consider any child word of $\mathbf{y}$, that is, a $\mathbf{w} \in \mathsf{star}(\mathbf{y})$. Since $\mathbf{y}$ is supported only on $[d] \setminus S$ and $\mathrm{supp}(\mathbf{w}) \subseteq \mathrm{supp}(\mathbf{y})$, every $\mathbf{w}_i = 0$ for $i \in S$. So $\mathbf{0}_S$ is observed once for every $\mathbf{w} \in \mathsf{star}(\mathbf{y})$ and there are $|\mathsf{star}(\mathbf{y})| = 2^{\varepsilon d}$ such $\mathbf{w}$. Hence, $\mathbf{0}_S$ occurs at least $2^{\varepsilon d}$ times.

Now that we have a lower bound on the frequency of $\mathbf{0}_S$, it remains to upper bound the F_p value when $\mathbf{y} \in T$ so that we are assured $\mathbf{0}_S$ will be a heavy hitter in this instance. The quantity we seek is the F_p value of all vectors in $\mathbf{A}^S$, written $F_p(\mathbf{A}, S)$; which we decompose into the contribution from $\mathbf{0}_S$ present due to $\mathbf{y}$ being in T, and two special cases from the block of $2^{\varepsilon d}$ all-ones rows and 'extra' copies of $\mathbf{0}_S$ which are contributed by vectors $\mathbf{y}' \neq \mathbf{y}$. We claim that this $F_p(\mathbf{A}, S)$ value is at most $|\mathcal{C}|^{1+p} 2^{\varepsilon d + (\varepsilon^2+\gamma)dp} + 3 \cdot 2^{\varepsilon p d}$.

First, let $\mathbf{y}' \in \mathcal{C}$ with $\mathbf{y}' \neq \mathbf{y}$ and consider prefixes $\mathbf{z}$ supported on S which can be generated by possible child words from $\mathsf{star}(\mathbf{y}')$. Since our code requires that $|\mathbf{y}' \cap \mathbf{y}| \leq (\varepsilon^2 + \gamma)d$, $\mathbf{y}'$ can have at most $(\varepsilon^2 + \gamma)d$ 1s located in $\bar{S} = [d] \setminus S$, and hence must have at least $(\varepsilon - \varepsilon^2 - \gamma)d$ 1s located in S. Since $|\mathsf{star}(\mathbf{y}')| = 2^{\varepsilon d}$, the number of copies of $\mathbf{z}$ inserted is at most $2^{\varepsilon d - (\varepsilon d - \varepsilon^2 d - \gamma d)} = 2^{\varepsilon^2 d + \gamma d}$. This occurs for every $\mathbf{y}' \in \mathcal{C}$ so the total number of occurrences of $\mathbf{z}$ is at most $|\mathcal{C}| 2^{(\varepsilon^2+\gamma)d}$. The contribution to F_p for this scenario is then $|\mathcal{C}|^p 2^{(\varepsilon^2+\gamma)dp}$. Observe that each codeword $\mathbf{y}'$ generates at most $2^{\varepsilon d}$ vectors under the $\mathsf{star}(\mathbf{y}')$ operator, so we have an upper bound of $|\mathcal{C}| 2^{\varepsilon d}$ such vectors generated, with a total contribution of $|\mathcal{C}|^{1+p} 2^{(\varepsilon^2 p + \varepsilon + \gamma p)d}$.

Next, we focus on the two special vectors to count which have a high contribution to the F_p value. Recall that Alice specifically included $\mathbf{1}_d$ into $\mathbf{A}$ $2^{\varepsilon d}$ times so the p-th powered frequency is exactly $2^{\varepsilon pd}$ for this term. From the above argument, $\mathbf{0}_S$ also has frequency $2^{\varepsilon d}$ from $\mathsf{star}(\mathbf{y})$. But $\mathbf{0}_S$ is also created at most $2^{(\varepsilon^2+\gamma)d}$ times from each $\mathbf{y}' \neq \mathbf{y}$ in T, giving an additional count of at most $|\mathcal{C}|2^{(\varepsilon^2+\gamma)d}$. Based on our choice of ε and γ, we can ensure that this is asymptotically smaller than $2^{\varepsilon d}$, and so the total contribution from these two special vectors is at most $3 \cdot 2^{\varepsilon d}$. So in total we achieve that F_p is at most $|\mathcal{C}|^{1+p}2^{\varepsilon d+(\varepsilon^2+\gamma)dp} + 3 \cdot 2^{\varepsilon pd}$, as claimed.

Then $\mathbf{0}_S$ meets the definition to be a ϕ-ℓ_p heavy hitter provided

$$2^{\varepsilon pd} > \phi^p(|\mathcal{C}|^{1+p}2^{\varepsilon d+(\varepsilon^2+\gamma)pd} + 3 \cdot 2^{\varepsilon pd}).$$

Assuming $p > 1$, and choosing ε sufficiently smaller than $(p-1)/p$ and γ sufficiently small, we have that

$$|\mathcal{C}|^{1+p}2^{\varepsilon d+(\varepsilon^2+\gamma)pd} \leq 2^{O(\gamma^2 d(1+p))+\varepsilon d+\varepsilon(p-1)d+\gamma pd} \leq 2^{\varepsilon pd}.$$

Hence, we require $2^{\varepsilon pd} > \phi^p O(2^{\varepsilon pd})$, i.e., $2^{\varepsilon d} > \phi O(2^{\varepsilon d})$, which is satisfied for a suitably small but constant ϕ.

Case 2: $y \notin T$. On the other hand, suppose that $\mathbf{y} \notin T$. Then the claim is that $\mathbf{0}_S$ is not a ϕ-ℓ_p-heavy hitter. Now the vector $\mathbf{0}_S$ does not occur with a high frequency because $\mathsf{star}(\mathbf{y})$ is not included in $\mathbf{A}$. However, certain child words in $\mathsf{star}(T)$ could also generate $\mathbf{0}_S$ when projected onto S and this is the contribution we need to upper bound. Again, any codeword $\mathbf{s} \in T$ has at least $(\varepsilon - \varepsilon^2 - \gamma)d$ 1s present on S. So for a particular $\mathbf{s} \in T$, $\mathbf{0}_S$ can occur $2^{\varepsilon^2 d+\gamma d}$ times. Taken over all $\mathbf{y}' \in \mathcal{C}$ for which Alice includes in $\mathbf{A}$, the frequency of $\mathbf{0}_S$ in this case is at most $|\mathcal{C}|2^{\varepsilon^2 d+\gamma d}$. Taking $\varepsilon < 1/3, \gamma < \varepsilon/3$ and using $|\mathcal{C}| = 2^{\gamma^2 d/\ln 2}$ (Lemma 3.3.1) we have $f_{e(\mathbf{0}_S)} \leq 2^{0.72\varepsilon d}$. Meanwhile, there are $2^{\varepsilon d}$ copies of the string $\mathbf{1}_d$ inserted into $\mathbf{A}$ meaning that $F_p(\mathbf{A}, S) \geq 2^{\varepsilon pd}$ and hence $F_p^{1/p}$ is strictly greater than $f_{e(\mathbf{0}_S)}$. Hence, $\mathbf{0}_S$ is *not* a ϕ-ℓ_p heavy hitter provided that $f_{e(\mathbf{0}_S)}/F_p^{1/p} = 2^{-0.28\varepsilon d}$ is strictly less than $\phi = 1/4$, this is satisfied for suitable ε and d.

Concluding the proof. Bob can use his test vector $\mathbf{y}$ and a query S with a constant factor approximation algorithm $\mathcal{A}$ for the ℓ_p-heavy hitters problem and distinguish between the two cases of Alice holding $\mathbf{y}$ or not based on whether $\mathbf{0}_S$ is reported. As a result, Bob can determine if $\mathbf{y} \in T$ and consequently solve Index, thus incurring the $\Omega(|\mathcal{C}|) = 2^{\Omega(d)}$ lower bound. □

The instance $\mathbf{A}$ is initialised with $2^{\varepsilon d}$ rows of the vector $\mathbf{1}_d$ and the child words $\mathsf{star}^Q(T)$. For any $\mathbf{t} \in \mathsf{star}^Q(T), |\mathsf{star}^Q(\mathbf{t})| = 2^{\varepsilon d}$ so the size of the instance A is $(|T| + 1)2^{\varepsilon d} \times d$.

3.5.3 F_p Estimation

The space complexity of approximating the frequency moments F_p has been widely studied since the pioneering work of Alon, Matias and Szegedy [AMS99a]. Here, we investigate their space complexity under projection. For $p = 1$, the frequency is always the number n of rows in the original instance irrespective of the column set S, so only one word of space is required. We therefore devote attention to $p \neq 1$.

Our approach again follows the general method shown in Section 3.3.3. The reduction to Index for Theorem 3.5.3 essentially reuses the argument from Theorem 3.5.2 for $p > 1$. However, for $p < 1$, we encode the problem slightly differently, closer to that in Theorem 3.4.1 as the projection column query reverts to $\text{supp}(\mathbf{y})$. Again, the reduction to Index relies on Bob determining whether or not Alice holds $\mathbf{y}$, which for F_p estimation amounts to Bob evaluating $F_p(\mathbf{A}, S)$ and comparing to a threshold value. Our first case is showing that if Bob's test vector $\mathbf{y}$ is not in Alice's set T, then the F_p value returned by an algorithm is at most $2^{(1-\alpha)\varepsilon d}$. We defer the proof of this specific calculation until Lemma 3.5.1 and take it as given for the proof of Theorem 3.5.3. On the other hand, if Alice had $\mathbf{y} \in T$, then we can show that the F_p value must be at least $2^{\varepsilon d}$ to obtain the constant factor gap.

Theorem 3.5.3. *Fix a real number $p > 0$ with $p \neq 1$. A constant factor approximation to the projected F_p estimation problem requires space $2^{\Omega(d)}$.*

Proof. For $p > 1$ we begin by noticing that in the proof for Theorem 3.5.2 one can also monitor the F_p value of the input to the problem rather than simply checking the heavy hitters. In particular, depending on whether or not Alice holds Bob's test word, $\mathbf{y}$, the projected F_p changes by more than a constant. Consequently, we invoke the same proof for F_p, $p > 1$ and obtain the same $2^{\Omega(d)}$ lower bound.

On the other hand, suppose that $p < 1$. We assume a code $\mathcal{C} \subset \mathcal{B}(d, \varepsilon d)$ with the property that any distinct $\mathbf{x}, \mathbf{x}' \in \mathcal{C}$ have $|\mathbf{x} \cap \mathbf{x}'| \leq cd$ for some small constant $c > \varepsilon^2$ (see Lemma 3.3.1). Again, Alice holds a subset $T \subseteq \mathcal{C}$ and inserts $\mathsf{star}(T)$ into the table for the problem $\mathbf{A}$. Throughout this proof we use a binary alphabet so suppress the Q notation from $\mathsf{star}^Q(\cdot)$. Bob holds a test vector $\mathbf{y} \in \mathcal{C}$ and is tasked with determining whether or not Alice holds $\mathbf{y} \in T$. We distinguish between the cases when Alice holds $\mathbf{y} \in T$ or not as follows. Bob uses $\mathbf{y}$ to determine the query column set $S = \text{supp}(\mathbf{y})$ and will compare against the returned frequency value from the algorithm.

Case 1: $\mathbf{y} \notin T$. Consider some $\mathbf{y}' \in \mathcal{C} \setminus \{\mathbf{y}\}$. Since $\mathbf{y}$ and $\mathbf{y}'$ are both codewords, they can have a 1 coincident in at most cd locations. So if Alice does not hold $\mathbf{y}$ then the codewords we need to consider are all binary words

in the code which have at most cd 1s in common with $\mathbf{y}$ on S. We denote this collection of words by M, i.e., the set of binary strings of length d that have at most cd locations set to 1. There are r such vectors, where r is defined by:

$$r \triangleq \sum_{i=0}^{cd} \binom{d}{i} \leq cd \cdot \binom{d}{cd} = O(d)2^{\Theta(cd)}.$$

The total count of all strings generated by Alice's encoding is at most $2^{\varepsilon d}|\mathcal{C}|$: each string in $\mathcal{C}$ generates $2^{\varepsilon d}$ subwords from the $\mathsf{star}(\cdot)$ operation. We now evaluate the ℓ_p-frequency of elements in the set M, denoted $F_p(M)$. For $p < 1$, the value $F_p(M)$ is maximized when every element of M has the same number of occurrences, $|\mathcal{C}|2^{\varepsilon d}/r$. As there are at most r members of M, we obtain $F_p(M) \leq |\mathcal{C}|^p 2^{\varepsilon dp} r^{1-p}$. Recalling the bounds on $|\mathcal{C}|$ and r, this is:

$$2^{cdp+\varepsilon dp+\Theta((1-p)cd)} \cdot O(d^{1-p}). \tag{3.7}$$

We can now choose c to be a small enough constant so that (3.7) is at most $2^{(1-\alpha)\varepsilon d}$ for a constant $\alpha > 0$. This is proved in Lemma 3.5.1, found in the subsequent subsection.

Case 2: $\mathbf{y} \in T$. Now consider the scenario when $\mathbf{y} \in T$ so that Alice has inserted $\mathsf{star}(\mathbf{y})$ into the table $\mathbf{A}$. Here, we can be sure that each of the $2^{\varepsilon d}$ strings in $\mathsf{star}(y)$ appears at least once over the column set S, and so the F_p value is at least $2^{\varepsilon d}1^p = 2^{\varepsilon d}$.

We observe that these two cases obtain the constant factor separation, as required. Then, Bob can use his test vector $\mathbf{y}$ and a query S with a constant factor approximation algorithm to the projected F_p-estimation problem and distinguish between the two cases of Alice holding $\mathbf{y}$ or not. Thus, Bob can determine if $\mathbf{y} \in T$ and consequently solve the Index problem, incurring the $\Omega(|\mathcal{C}|) = 2^{\Omega_c(d)}$ lower bound for a c arbitrarily small. □

Remark 3.5.1. *For $p > 1$ we adopt the same instance as in Theorem 3.5.2 so the instance is of size $(|T| + 1)2^{\varepsilon d} \times d$. On the other hand, for $0 < p < 1$, only the words in $\mathsf{star}^Q(T)$ are required so $\mathbf{A}$ has size $|T|2^{\varepsilon d} \times d$.*

Upper Bounding F_p in Equation (3.7)

A key step in the proof of Theorem 3.5.3 is that in Equation (3.7), the expression

$$2^{cdp+\varepsilon dp+\Theta((1-p)cd)} \cdot O(d^{1-p})$$

can be bounded by a manageable power of two. We formalise this in Lemma 3.5.1.

Lemma 3.5.1. *Under the same assumptions as in Theorem 3.5.3, there exists a small constant $c > 0$ which bounds (3.7) by at most $2^{(1-\alpha)\varepsilon d}$ for some $\alpha > 0$.*

Proof. Here we use base-2 logarithms and let $0 < c < 1$ be a small constant which we need to bound. Also, let $0 < p < 1$ be a given constant. Observe that the $O(d^{1-p})$ term only contributes positively in the exponent term of (3.7) so we can ignore it from the calculation. Write $2^{\Theta(cd(1-p))} = 2^{cd\alpha(1-p)}$ for $\alpha > 0$. This follows from:

$$\binom{d}{cd} \leq \left(\frac{ed}{cd}\right)^{cd} \leq 2^{(2+\log\frac{1}{c})cd} \tag{3.8}$$

so let $\alpha = 2 + \log\frac{1}{c}$. For clarity, we proceed by using the trivial identity $1-(1-\nu) = \nu$ and show that $1 - \nu > 0$ for ν a function of c, p, d. We need to ensure:

$$cpd + \varepsilon dp + \alpha cd(1-p) \leq (1-\alpha)\varepsilon d. \tag{3.9}$$

This amounts to showing that:

$$\nu \triangleq cp/\varepsilon + p + \alpha c(1-p)/\varepsilon \leq (1-\alpha)$$

Now, $\nu = p(c/\varepsilon + 1 - \alpha c/\varepsilon) + \alpha c/\varepsilon$ and we require $\nu < 1$. We may enforce the weaker property of $p(c/\varepsilon + 1 - \alpha/\varepsilon) < 1$ because $c > 0$ and for $c < 4$ we also have $\alpha > 0$ (inspection on Equation (3.8)) so $\alpha c/\varepsilon > 0$, and so can be omitted. Solving for c we obtain $c(1-\alpha) < \varepsilon(1/p - 1)$. Recalling the definition of α this becomes:

$$c(\log c - 1) < \varepsilon(1/p - 1) \tag{3.10}$$

from which positivity on c yields $c \log c < \varepsilon(1/p - 1)$. Hence, it is enough to use $c < \varepsilon(1/p - 1)$. □

3.5.4 ℓ_p-Sampling

In the projected ℓ_p-sampling problem, the goal is to sample a row in $\mathbf{A}^S$ proportional to the p-th power of its number of occurrences. One approach to the standard (non-projected) ℓ_p-sampling problem on a vector $\mathbf{x}$ is to subsample and find the ℓ_p-heavy hitters [LNNT16]. Consequently, if one can find ℓ_p-heavy hitters for a certain value of p, then one can perform ℓ_p-sampling in the same amount of space, up to polylogarithmic factors. Interestingly, for projected ℓ_p-sampling, this is not the case, and we show for every $p \neq 1$, there is a $2^{\Omega(d)}$ lower bound. This is despite the fact that we can estimate ℓ_p-frequencies efficiently for $0 < p < 1$, and hence find the heavy hitters (Section 3.5.1).

We again follow the now familiar framework of Section 3.3.3 and separate the two cases of $p > 1$ and $p < 1$. For $p > 1$ it is possible to reuse the argument for heavy hitters from Theorem 3.5.2 as was previously done in the projected F_p estimation lower bound. Recall that Theorem 3.5.2 showed that $\mathbf{0}_S$ was

a heavy hitter if Alice had $\mathbf{y} \in T$ and was not heavy if she did not hold $\mathbf{y}$. Similarly, the consequence of this is showing that $\mathbf{0}_S$ will be sampled 'often' if Alice holds $\mathbf{y}$, otherwise $\mathbf{0}_S$ is sampled rarely.

For $p < 1$ we follow a similar approach but rather than showing $\mathbf{0}_S$ is sampled often, Bob will query an ℓ_p sampler and check if the returned strings are 'often' in (some specially chosen subset of) $\mathsf{star}(\mathbf{y})$. If this is the case, then Alice holds $\mathbf{y}$. On the other hand, if Alice did not hold $\mathbf{y}$, then the returned strings from the ℓ_p sampler are almost never in the special set that Bob checks against.

Theorem 3.5.4. *Fix a real number $p > 0$ with $p \neq 1$, and let $\varepsilon \in (0, 1/2)$. Let $S \subseteq [d]$ be a column query and i be a pattern observed on the projected data $\mathbf{A}^S$. Any algorithm which returns a pattern i sampled from a distribution $(p_1, \ldots, p_n)$, where $p_i \in (1 \pm \varepsilon)\frac{f_{e(i)}^p}{\|f(\mathbf{A},S)\|_p^p} + \Delta$ together with a $(1 \pm \varepsilon')$-approximation to p_i, $\Delta = 1/\operatorname{poly}(nd)$ and $\varepsilon' > 0$ is a sufficiently small constant, requires $2^{\Omega(d)}$ bits of space.*

Proof. **Case 1: $p > 1$.** The proof of Theorem 3.5.2 argues that the vector $\mathbf{0}_S$ is a constant factor ℓ_p-heavy hitter for any $p > 1$ if and only if Bob's test vector $\mathbf{y}$ is in Alice's input set T, via a reduction from Index. That is, we argue that there are constants $C_1 > C_2$ for which if $\mathbf{y} \in T$, then $f_{e(\mathbf{0}_S)}^p \geq C_1 F_p$, while if $\mathbf{y} \notin T$, then $f_{e(\mathbf{0}_S)}^p < C_2 F_p$. Consequently, given an ℓ_p-sampler with the guarantees as described in the theorem statement, then the (empirical) probability of sampling the item $\mathbf{0}_S$ should allow us to distinguish the two cases. This holds even tolerating the $(1 + \varepsilon')$-approximation in sampling rate, for a sufficiently small constant ε'. In particular, if $\mathbf{y} \in T$, then we will indeed sample $\mathbf{0}_S$ with $\Omega(1)$ probability, which can be amplified by independent repetition; whereas, if $\mathbf{y} \notin T$, we do not expect to sample $\mathbf{0}_S$ more than a handful of times. Consequently, for $p > 1$, an ℓ_p-sampler can be used to solve the ℓ_p-heavy hitters problem with arbitrarily large constant probability, and thus requires $2^{\Omega(d)}$ space.

Case 2: $0 < p < 1$. We now turn to $0 < p < 1$. In the proof of Theorem 3.5.3, a reduction from Index is described where Alice holds the set T and Bob the string $\mathbf{y}$. Bob can generate the set $\mathsf{star}(\mathbf{y})$ of size $2^{\varepsilon d}$ which is all possible binary strings supported on the column query S. From this, Bob constructs the set $M' = \left\{\mathbf{z} \in \mathsf{star}(\mathbf{y}) : |\operatorname{supp}(\mathbf{z})| \geq \frac{\varepsilon d}{2}\right\}$. We observe that if $\mathbf{y} \in T$ then at least half of the strings in $\mathsf{star}(\mathbf{y})$ are supported on at least $\varepsilon d/2$ coordinates which implies $|M'| \geq 2^{\varepsilon d - 1}$. The total F_p in this case can be bounded by a contribution of $|M'|1^p + 2^{\varepsilon d}$. The first term arises from the $|M'|$ strings in M' with a frequency of 1, while the second term is shown in Case 1 of Theorem 3.5.3. Since $|M'| \leq 2^{\varepsilon d}$, we have that $F_p \leq 2^{\varepsilon d + 1}$ in this case. Consequently, the correct probability of ℓ_p-sampling returning a string in M' is at least $\frac{1}{4}$ for the

"ideal" case of $\varepsilon = 0, \Delta = 0$. Even allowing $\varepsilon < \frac{1}{2}$ and $\Delta = 1/\operatorname{poly}(nd)$, this probability is at least $1/10$.

Otherwise, if $\mathbf{y} \notin T$, we exploit that $\mathbf{y}' \neq \mathbf{y}$ can coincide in at most $cd = O(\varepsilon^2 d)$ coordinates and $|\operatorname{supp}(\mathbf{z})| \geq \varepsilon d/2 > cd$ for any $\mathbf{z} \in M'$. Hence, no $\mathbf{z} \in M'$ can occur in $\mathsf{star}(\mathbf{y}')$ for another $\mathbf{y}' \in \mathcal{C} \setminus \{\mathbf{y}\}$ on the column projection S. In this case, there should be zero probability of sampling a string in M' (neglecting the trivial additive probability Δ).

To summarize, in the case that $\mathbf{y} \in T$, by querying the projection $S = \operatorname{supp}(\mathbf{y})$ then a constant fraction of the F_p-mass is on the set M', whereas when $\mathbf{y} \notin T$, then there is zero F_p-mass on the set M'. Since Bob knows M', he can run an ℓ_p-sampler and check if the output is in the set M', and succeed with constant probability. It follows that Bob can solve the Index problem (amplifying success probability by independent repetitions if needed), and thus again the space required is $2^{\Omega(d)}$. □

Remark 3.5.2. *For $p > 1$ we again adopt the same instance as in Theorem 3.5.2 which has size $(|T| + 1)2^{\varepsilon d} \times d$. However, for $0 < p < 1$, we require the instance from Theorem 3.5.3 so $\mathbf{A}$ has size $|T|2^{\varepsilon d} \times d$.*

3.6 Projected Frequency Estimation via Set Rounding

Although our lower bounds rule out the possibility of computing constant factor approximations to projected frequency problems in sub-exponential space, it is still possible to compute non-trivial approximations using exponential space but still better than naïvely enumerating all column subsets of $[d]$. We design a class of algorithms that proceed by keeping appropriate sketch data structures for a "net" of subsets. The net has the property that for any query $S \subset [d]$ there is a $S' \subset [d]$ stored in the net which is not too different from S. We can then answer the query on S using the summary data structure computed for columnset S'. To formalize this approach we need some further definitions, the first of which conceptualizes the notion of a net over subsets. Secondly, we define an α-neighbour of a set S which intuitively, is a set in the net that differs from S in at most an αd fraction of the items.

Definition 3.6.1 (α-net of subsets)**.** *Let $\mathcal{P}([d])$ denote the power set of $[d]$. Fix a parameter $\alpha \in (0, 1/2)$. An α-net of $\mathcal{P}([d])$ is the set $\mathcal{N} = \{U \in \mathcal{P}([d]) : |U| \leq d/2 - \alpha d \text{ or } |U| \geq d/2 + \alpha d\}$ which contains all subsets $U \in \mathcal{P}([d])$ whose size is at most $d/2 - \alpha d$ or at least $d/2 + \alpha d$.*

Definition 3.6.2 (α-neighbour)**.** *Let $S \in \mathcal{P}([d])$ and $\mathcal{N}$ be an α-net of $\mathcal{P}([d])$. We say that a set U is an α-neighbour of S if $U \in \mathcal{N}$ and the symmetric*

difference between S and U is at most αd.

Example 3.6.1 (α-net and α-neighbours)**.** *Let $d = 6$ and $\alpha = 1/6$ so that the power set $\mathcal{P}([6])$ is of size 64. A $1/6$-net for $\mathcal{P}([6])$ has keeps all subsets of size at most $d/2 - 1/6 = 2$ and at least $d/2 + 1/6 = 4$ by Definition 3.6.1.*

In Figure 3.3, the power set and net are both illustrated. The index at every level represents the size of subsets from $[6]$. Hence, the net contains all subsets stored at levels $0, 1, 2, 4, 5, 6$. the size of the net is $\sum_{i=0}^{2} \binom{6}{i} + \sum_{i=4}^{6} \binom{6}{i} = 44$. This is to be compared to $|\mathcal{P}([d])| = 2^6$.

If the query is set $S = \{1, 2, 3\}$ then S is not in the net $\mathcal{N}$ because it has size 3, hence we will look for $1/6$-neighbours of S. A $1/6$ neighbour is any set $U \in \mathcal{N}$ that differs from S in at most 1 element. Then $U_1 = \{1, 2\}$ and $U_2 = \{1, 2, 3, 4\}$ are in $\mathcal{N}$ and have only one element different from S so are $\frac{1}{6}$ neighbours. On the other hand, $U_3 = \{1, 2, 4\}$ which, although differing in only one element from S, has size 3, so is not in $\mathcal{N}$ and thus cannot be a $1/6$ neighbour.

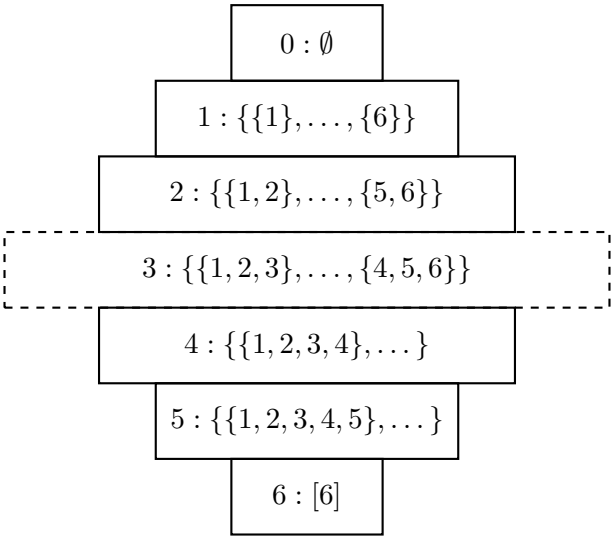


Figure 3.3: All subsets of [6] except those of size 3 (indicated by the dashed box) are included in the $1/6$ net $\mathcal{N}$. The index of the box indicates the size of subsets that are stored in that level with the ellipses marking that *all* subsets of that size are stored in the net. For example, both subsets $\{1, 2, 3, 4\}$ and $\{2, 3, 5, 6\}$ are included at the size 4 level of the net. However, any subset of [6] with size 3 is not included in the net.

Let $H(t) = -t \log_2(t) - (1-t) \log_2(1-t)$ denote the *binary entropy function* defined over $t \in [0, 1]$. We plot H over $t \in [0, 1/2]$ in Figure 3.4. Since H is symmetric about $t = 1/2$ and we will take $t = 1/2 - \alpha$ for $\alpha \in (0, 1/2)$ it suffices to consider only this domain.

Lemma 3.6.1. *Let $\mathcal{N}$ be an α-net for $\mathcal{P}([d])$. Then $|\mathcal{N}| \leq 2^{H(1/2-\alpha)\cdot d+1}$.*

Proof. The total number of subsets whose size is at most $d/2 - \alpha d$ is $\sum_{i \leq \alpha d} \binom{d}{i}$ and $\sum_{i \leq \alpha d} \binom{d}{i} \leq 2^{H(1/2-\alpha)d}$ [Gal14, Theorem 3.1]. By symmetry we obtain

the same bound for the number of subsets of size at least $d/2 + \alpha d$, yielding the claimed total. □

Understanding The Space Bound

Due to the binary entropy function, Lemma 3.6.1 appears a little terse. However, for any $t \in [0, 1/2], H(t) \in [0, 1]$, thus for $c = H(t)$, the bound on $\mathcal{N}$ from Lemma 3.6.1 becomes 2^{cd+1}. Since the exhaustive method retains a sketch for every one of the 2^d subsets in $\mathcal{P}([d])$, it is convenient to transform the bound so that the two quantities are easily comparable. That is, we want c so that $cd+1 < d$ which is satisfied provided $c < 1 - 1/d$ and as d grows, this condition only becomes easier to satisfy. Choose such a c so that $cd + 1 = \gamma d < d$. Now, our bound is effectively asking for a net $\mathcal{N}$ whose size is $2^{\gamma d}$. Clearly, this grows exponentially (as seen in Figure 3.5a with $|\mathcal{N}|$ plotted on a log scaled axis) but $\gamma < 1$ so $2^{\gamma d}$ is sublinear in 2^d.

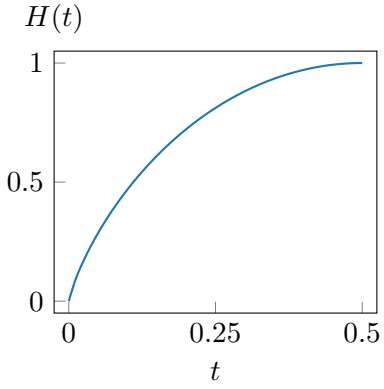
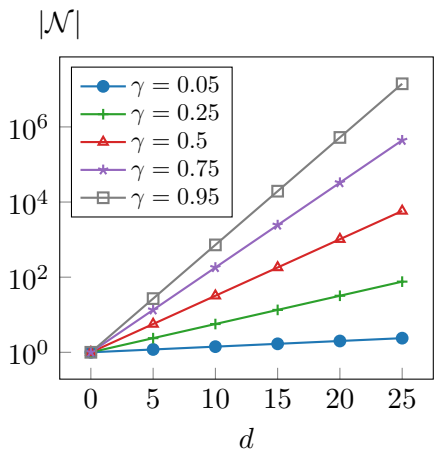
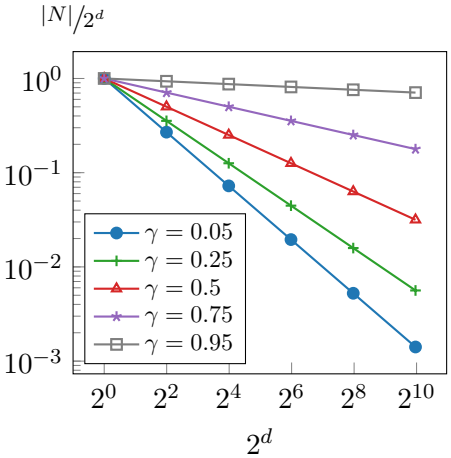


Figure 3.4: Binary entropy $H(t)$



(a) The number of sets stored in a net of size $|\mathcal{N}| = 2^{\gamma d}$ (log scale).



(b) Fraction of sketches stored by $\mathcal{N}$ vs all 2^d subsets of $[d]$ (log-log scale).

Figure 3.5: Behaviour of an α-net $\mathcal{N}$

We can understand the space saving offered by using the net through measuring the *fraction of sketches stored* in $\mathcal{N}$ versus storing a single sketch for every subset in $\mathcal{P}([d])$. That is, we measure $|\mathcal{N}|/2^d = 2^{(\gamma-1)d}$. In Figure 3.5b we plot straight lines of gradient $-(1-\gamma)$ on a log-log scale to illustrate how large the net is compared to 2^d. Unsurprisingly, as γ increases towards 1, we keep

a larger portion of $\mathcal{P}([d])$ so there is little saving in using the net. However, when γ is small, there is a substantial saving in using the net rather than storing a sketch for every member of $\mathcal{P}([d])$.

These illustrations are useful as they allow a practitioner to focus on the net size as a parameter of γ rather than tinkering with $H(1/2-\alpha)$. For instance, suppose that $d = 10$ and the user knew they could only store a sketch for at most 1% of all possible subsets of $[d]$ (i.e., roughly 10 subsets rather than 1024). Then Figure 3.5b can roughly inform the parameter choice for $\mathcal{N}$ as it ilustrates a $\gamma < 0.25$ will suffice.

Figures 3.5a and 3.5b give cause for cautious optimism. Although the size of the net is exponential in d, we can set this to be a small fraction of 2^d so that it is more manageable. However, it remains to understand how solving problems on the net can affect the quality of estimation and this forms the basis of the subsequent section.

3.6.1 From α-nets to Projections

Suppose that we are tasked with answering problem $P = P(\mathbf{A}, S)$ on a projection query S. We know that if S is known ahead of time then we can encode the input data $\mathbf{A} \in [Q]^{n\times d}$ on projection S as a standard stream over the alphabet $[Q]^{|S|}$. The use of α-nets allows us to sketch some of the input and use this to approximately answer a query. For a standard streaming problem, we will say that an algorithm yields a *β-approximation* to the true solution z^* if the returned estimate $z \in [z^*/\beta, \beta z^*]$. A sketch obtaining such approximation guarantees will be referred to as a *β approximate sketch*. We additionally need the following notion of error due to the distortion incurred when answering queries on elements of the α-net rather than the given query.

Definition 3.6.3 (Rounding distortion)**.** *Let $P = P(\mathbf{A}, S)$ be a projection query for the problem P on input $\mathbf{A} \in [Q]^{n\times d}$ with projection S. Let $\mathcal{N} \subset \mathcal{P}([d])$ be an α-net. The* rounding distortion *$r(\alpha, P)$ is the worst-case determinstic error incurred by solving $P(\mathbf{A}, S')$ rather than $P(\mathbf{A}, S)$ for an α-neighbour $S' \in \mathcal{N}$ of C so that $P(\mathbf{A}, S)/r(\alpha, P) \leq P(\mathbf{A}, S') \leq r(\alpha, P)P(\mathbf{A}, S)$.*

Definition 3.6.3 is easiest to conceptualize for the F_0 problem when $\mathbf{A} \in \{0,1\}^{n\times d}$. Specifically, $P = F_0$ and the task to solve is $P = F_0(A, S)$. For a given query S, with an α-neighbour S' in the net, the gap between the number of distinct items observed on S' at most doubles for each column in the set difference between S and S'. Since S' is an α-neighbour, we have the symemtric difference $|S' \,\Delta\, S| \leq \alpha d$ so the worst-case approximation factor in the number of distinct items observed over S' rather than S is $2^{\alpha d}$.

Example 3.6.2 (Projected F_0 by set rounding)**.** *Consider the input array $\mathbf{A} \in \{0,1\}^{8\times 6}$ as shown in (†). Assume we take a 1/6 net $\mathcal{N}$ for $\mathcal{P}([6])$. Suppose*

Algorithm 2: Projected frequency estimation by query rounding

Input: Data $\mathbf{A} \in \{0,1\}^{n \times d}$, parameter $\alpha \in (0, 1/2)$, frequency estimation problem P, query S revealed after $\mathbf{A}$

Function `ProjectedFreq(`$\mathbf{A}, \alpha, S$`):`
- Generate an α-net $\mathcal{N}$
- For every $U \in \mathcal{N}$ evaluate a β approximate sketch to estimate $P(\mathbf{A}, U)$
- Given a projection query S after observing $\mathbf{A}$:
- Obtain S', an α-neighbour to S in $\mathcal{N}$
- **return** $P(\mathbf{A}, S')$ *to* β *relative error*

the query set is $S = \{1, 2, 3\}$. *The symbol* $\star \in \{0, 1\}$ *is arbitrary and makes no difference to this example as the query set is on the first three columns.*

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 0 & 0 & 0 & 0 & \star & \star \\ 1 & 0 & 1 & 0 & \star & \star \\ 1 & 1 & 0 & 0 & \star & \star \\ 0 & 1 & 1 & 0 & \star & \star \\ 0 & 0 & 0 & 1 & \star & \star \\ 1 & 0 & 1 & 1 & \star & \star \\ 1 & 1 & 0 & 1 & \star & \star \\ 0 & 1 & 1 & 1 & \star & \star \end{bmatrix} \qquad \begin{bmatrix} \mathbf{A} \\ \mathbf{u} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 0 & 0 & 0 & 0 & \star & \star \\ 1 & 0 & 1 & 0 & \star & \star \\ 1 & 1 & 0 & 0 & \star & \star \\ 0 & 1 & 1 & 0 & \star & \star \\ 0 & 0 & 0 & 1 & \star & \star \\ 1 & 0 & 1 & 1 & \star & \star \\ 1 & 1 & 0 & 1 & \star & \star \\ 0 & 1 & 1 & 1 & \star & \star \\ \hline 1 & 1 & 1 & 1 & \star & \star \end{bmatrix}. \qquad (\dagger)$$

For the projected F_0 *problem* $F_0(\mathbf{A}, S) = 4$. *Suppose that we take two sets* $U_1 = \{2, 3\}, U_2$, *from the net* $\mathcal{N}$. *Then* $F_0(\mathbf{A}, U_1) = 4$ *while* $F_0(\mathbf{A}, U_2) = 8$, *hence we could always estimate* $4 \leq \hat{F}_0(\mathbf{A}, S) \leq 8$.

On the other hand, if the input were $\mathbf{B} = (\mathbf{A}^\top | \mathbf{u}^\top)^\top$ *as in* $(\dagger)$. *Then* $F_0(\mathbf{B}, S) = 5$ *and* $F_0(\mathbf{B}, U_2) = 9$ *both increase by only* 1 *compared to the corresponding quantities over* $\mathbf{A}$. *However,* $F_0(\mathbf{B}, U_1) = 4$ *is unchanged so we achieve* $4 \leq \hat{F}_0(\mathbf{A}, S) \leq 9$. *Although this would suggest that the bounds from rounding to projections stored in the net are pessimistic, the example of* $\mathbf{A}, S$ *and* U_1, U_2 *(rather than* $\mathbf{B}$*) suggests that the worst-case rounding bounds can be achieved and likely cannot be improved without further assumptions between the features, for example, perhaps a naïve Bayesian model as suggested in [KMVX18]. The worst case instance for achieving the distortion for projected* F_0 *is found by taking a full binary code of length* d *and ordering the columns so that a query of size* $k < d$ *can have its* F_0 *grow or reduce by a factor of* 2.

More generally, we can categorize the rounding distortion for other typical queries, as demonstrated in the following lemma. Note that if the query is contained in the α-net $\mathcal{N}$ then we will retain a sketch for that problem; hence

the distortion is only incurred for queries not contained in the net.

Lemma 3.6.2. *Fix $\alpha \in (0, 1/2)$, suppose $\mathbf{A} \in \{0,1\}^{n\times d}$ and $\mathcal{N}$ be an α-net. If S is a projection query for the following cases, the rounding distortion can be bounded as:*

1. *$P = F_0(\mathbf{A}, S)$ then $r(\alpha, F_0) = 2^{\alpha d}$*
2. *$P = F_p(\mathbf{A}, S), p > 1$ then $r(\alpha, F_p) = 2^{\alpha d(p-1)}$*
3. *$P = F_p(\mathbf{A}, S), p < 1$ then $r(\alpha, F_p) = 2^{\alpha d(1-p)}$*

Proof. Item (1) is an immediate consequence of the discussion preceding Example 3.6.2 so we focus on (2) and (3). Suppose $p \geq 1$. Let $f_S = f(\mathbf{A}, S)$ denote the frequency vector associated to the projection query S over domain $[2^{|S|}]$. First, consider a single index $j \in [2^{|S|}]$ with $(f_S)_j = x$. Let S' be an α-neighbour for S in $\mathcal{N}$, and without loss of generality, assume that $|S| < |S'|$. The task is to estimate $\|f_S\|_p^p = x^p$ from $\|f_{S'}\|_p^p$, where $f_{S'} = f(\mathbf{A}, S')$ is a frequency vector over the domain $[2^{|S'|}]$ which is a $|S' \setminus S|$ factor larger than the domain for f_S. However, observe that in $f_{S'}$, the value of x is spread across the at most $2^{\alpha d}$ entries that agree with j on columns S. The contribution to F_p from these entries is at most x^p (if the mass of x is mapped to a single entry). On the other hand, by Jensen's inequality, the contribution is at least $2^{\alpha d}(x/2^{\alpha d})^p = x^p/2^{\alpha d(p-1)}$. Hence, considering all entries j, we obtain $\|f_S\|_p^p/2^{\alpha d(p-1)} \leq \|f_{S'}\|_p^p \leq \|f_S\|_p^p$. In the case $|S| > |S'|$, essentially the same argument shows that $\|f_S\|_p^p \leq \|f_{S'}\|_p^p \leq \|f_S\|_p^p 2^{\alpha d(p-1)}$. Thus we obtain the rounding distortion of $2^{\alpha d(p-1)}$. For $p < 1$, we proceed as above, except by concavity, the ordering is reversed. □

Observe that the distortion reduces to 1 (no distortion) as we approach $p = 1$ from either side. This is intuitive, since the F_1 problem is simply to report the number of rows in the input, regardless of S, and so the problem becomes "easier" as we approach $p = 1$.

With these properties in hand, we can give a "meta-algorithm" as described in Algorithm 2. In Theorem 3.6.1 we can fully characterize the accuracy-space tradeoff for Algorithm 2 as a function of α and d.

Theorem 3.6.1. *Let $\mathbf{A} \in \{0,1\}^{n\times d}$ be the input data and $S \subseteq [d]$ be a projection query. Suppose $P = P(\mathbf{A}, S)$ is the projected frequency problem, $\alpha \in (0, 1/2)$ and $r(\alpha, P)$ is the rounding distortion. With probability at least $1 - \delta$ a $\beta \cdot r(\alpha, P)$ approximation can be obtained by keeping $\tilde{O}(2^{H(1/2-\alpha)d})$ β-approximate sketches.*

Proof. Let $\mathcal{N}$ be a α-net for $\mathcal{P}([d])$ and for every $U \in \mathcal{N}$ generate a sketch with accuracy parameter β for the problem P on the projection defined by

$U \subseteq [d]$. Either the projection $S \in \mathcal{N}$, in which case we can report a β factor approximation, or $S \notin \mathcal{N}$ in which case we take an α-neighbour, $S' \in \mathcal{N}$ and return the estimate z for $P(\mathbf{A}, S')$. The sketch ensures that the answer to $P(\mathbf{A}, S')$ is obtained with accuracy β, which by the rounding distortion is a $\beta \cdot r(\alpha, P)$ approximation. To obtain this guarantee we build one sketch for every $U \in \mathcal{N}$, for a total of $O(2^{H(1/2-\alpha)d})$ sketches (via Lemma 3.6.1). By setting the failure probabilty for each sketch as $\delta = 1/2^{\alpha d}$ and then taking a union bound over the α-net we achieve probability at least $1 - \delta$. □

Remark 3.6.1. *By taking $N = 2^d$ we achieve the presentation claimed at the start of this chapter. For $\alpha \in (0, 1/2)$, the space usage of our algorithm grows as $2^{H(1/2-\alpha)}$. Taking $c' = H(1/2 - \alpha)$ we obtain $N^{c'}$ space, as claimed. For the error, the overall approximation factor is $\beta r(\alpha, d) = \beta 2^{\alpha d q}$ by Lemma 3.6.2 with $q = |1 - p|$. Thus, the worst-case approximation factor Theorem 3.6.1 achieves is $N^c = N^{\alpha q \log_2(\beta)}$. Since β is the approximation factor from a standard streaming problem, we can think of this being a small constant, say 2 or 4, for example. This results in $c = \alpha \cdot \text{constant}|1 - p|$ which is less than 1 provided α is chosen correctly.*

Illustration of Bounds. First, observe that, irrespective of the problem P, the number of sketches needed is sublinear in 2^d. This is due to the fact that the entropy $H(1/2 - \alpha) < 1$ for $\alpha > 0$, so the size of the net $|\mathcal{N}| < 2^d$. For $0 \leq p \leq 2$, we have β-approximate sketches with $\beta = (1 + \varepsilon)$ whose size is $\tilde{O}(\varepsilon^{-2})$, which is constant for constant ε. For example, we obtain a $2^{\alpha d}$ approximation (ignoring small constant factors) for F_0 in space $O(2^{H(1/2-\alpha)}d)$, using for instance the $(1+\varepsilon)$-approximate sketch from [KNW10] which requires $O(\varepsilon^{-2} + \log n')$ bits for an input over domain $\{1, \ldots, n'\}$. Since $n' \leq 2^d$, and setting $\varepsilon = 1$, we obtain the approximation in space $O(d2^{H(1/2-\alpha)d})$. This is to be compared to the bounds in Section 3.4, where it is shown that (binary) instances of the projected F_0 problem require space $2^{\Omega(d)}$. These results show that the constant hidden by the $\Omega()$ notation is less than 1.

In Figure 3.6 we illustrate the general behavior of the bounds for $d = 20$. These plots differ from Figures 3.5a and 3.5b which only illustrates the space-saving properties of the net. The subsequent plots relate the space benefit to the approximation factor induced by the net. We plot the *relative space* by $2^{H(1/2-\alpha)}/2^d$ while varying α over $(0, 1/2)$ (plotted in the top pane). This shows the space reduction in using the α-net approach compared to naïvely storing all 2^d queries. The central pane shows how the *approximation factor* $2^{\alpha d}$ (on a log scale) varies with α. Although Theorem 3.6.1 shows that the approximation factor is $\beta r(\alpha, d)$, we can think of the superfluous terms as being small constants relative to $2^{\alpha d}$ so only plot this term for clarity. We plot the space-approximation tradeoff in the bottom pane and the approximation

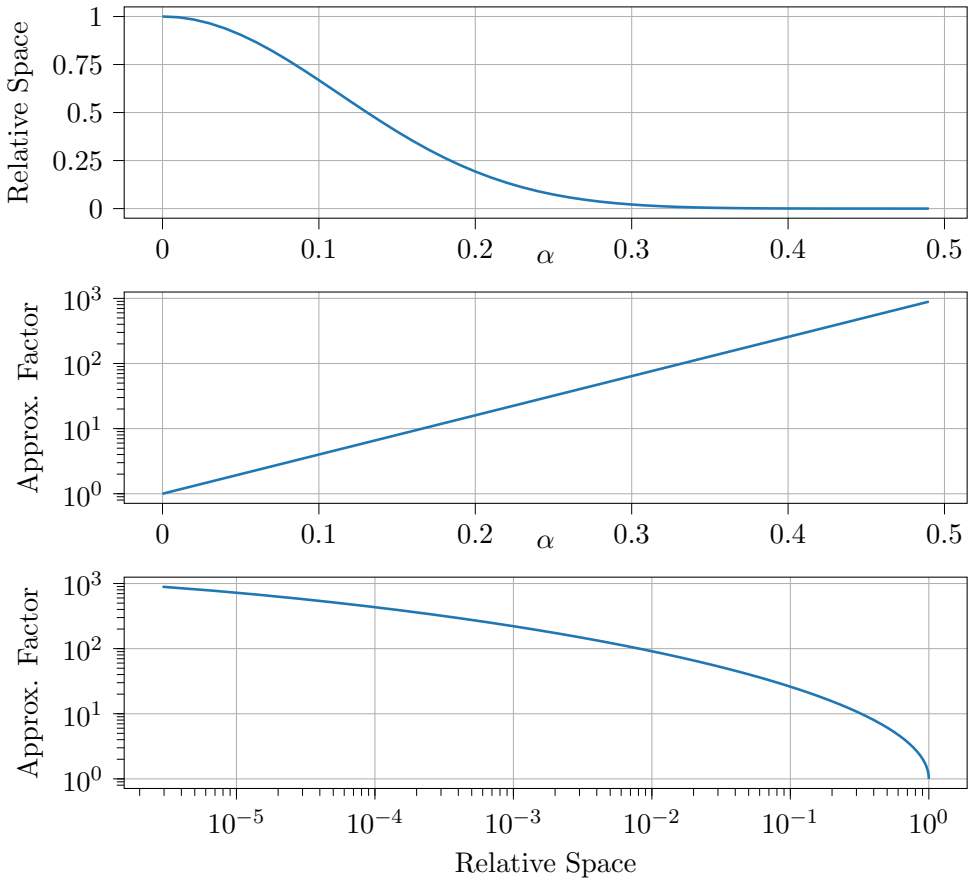


Figure 3.6: Space-approximation tradeoff for $d = 20$ as α is varied from 0 to 1/2. Relative space is $2^{H(1/2-\alpha)d}/2^d$ is the fraction of total subsets stored and Approximation Factor is $2^{\alpha d}$. As α approaches 0, the net stores increasingly more subsets from $\mathcal{P}([d])$ so the relative space approaches 1 which also explains why the approximation factor is smaller. When α approaches 1/2, many fewer subsets from $\mathcal{P}([d])$ are stored so the relative space is small and approaches 0; as fewer subsets are stored, the error induced by set rounding will be higher, as illustrated.

factor is again plotted on a $\log_{10}$-scale. This plot suggests that if we reduce the space by a factor of 10 (i.e., permit relative space 10^{-1}) then the approximation factor is on the order of 10s. Meanwhile, if we use relative space 10^{-3}, then we achieve a space-saving of a factor 1000 and the approximation remains on the order of hundreds: this is substantial as the number of summaries kept for the approximation is approximately $2^{10} = 1024 \ll 2^{20} \approx 10^6$. One can see this by observing $10^3 \approx 2^{10}$, so relative space 10^{-3} roughly corresponds to keeping 2^{10} subsets out of all possible 2^{20} sets in $\mathcal{P}([20])$.

3.7 Concluding Remarks

We have introduced the topic of projected frequency estimation, with the aim of abstracting a range of problems involving computing functions over projected subspaces of data. Our main results show that these problems are generally hard, in terms of the space requirements: in most cases, we require space which is exponential in the dimensionality d of the input. However, interestingly, the exact dependence is not as simple as 2^d: we show that coarse approximations can be obtained whose cost is substantially sublinear in 2^d. Letting $N = 2^d$, our upper and lower bounds establish that the space complexity for a number of problems here is polynomial in N, though substantially sublinear. And, in a few special cases (ℓ_p frequency estimation for $p \leq 1$), a sufficiently constant-sized sample suffices for accurate approximation of projected frequencies. It remains an intriguing open question to close the gaps between the upper and lower bounds, and to find the exact form of the polynomial dependence on N for these problems.

Chapter 4

Streaming Deterministic Summaries in ℓ_p Norms

4.1 Introduction

Prior work on approximate linear algebra has led to efficient distributed and streaming algorithms for problems such as approximate matrix multiplication, low rank approximation, and regression. Primarily, these problems have been studied in ℓ_2, the Euclidean norm, and rely on constructions outlined in Section 2.2. In this chapter, we study other ℓ_p norms, which are more robust for $p < 2$, and can be used to find outliers for $p > 2$. Unlike previous algorithms for such norms, we give algorithms that are deterministic, work simultaneously for every $p \geq 1$, including $p = \infty$, and (3) can be implemented in both distributed and streaming environments. We apply our results to ℓ_p-regression and entrywise ℓ_1-low rank approximation.

4.1.1 Background

Analysing large-scale, high volume data can be time-consuming and resource intensive. Core data analysis, such as robust instances of regression, involve convex optimization tasks over large matrices can be time-consuming and resource intensive as they may not naturally distribute or parallelize. In response to this, approximation algorithms have been proposed which follow a "sketch and solve" paradigm: produce a reduced size representation of the data, and solve a version of the problem on this summary [Woo14b]. It is then argued that the solution on the reduced data provides an approximation to the original problem on the original data. This paradigm is particularly attractive when the summarization can be computed efficiently on partial views of the full data, for example, when it can be computed incrementally as the data arrives (streaming model) or assembled from summarizations of disjoint partitions of

the data (distributed model) [Woo14b, ACH+12, FMS+06]. This template has been instantiated for a number of fundamental tasks in large-scale linear algebra such as matrix multiplication, low rank approximation, and regression.

Our understanding is well-established in the common case of the Euclidean norm, i.e., when distances are measured under the Minkowski p-norm for $p = 2$. Here, it suffices to choose a sketching matrix independent of the data—where each entry is i.i.d. Gaussian, Rademacher, or more efficient variants that we will introduce in Section 5.1.1. For other p values, less is known, but these are often needed to handle limitations of the 2-norm. For instance, $p = 1$ is widely used as it is robust with respect to the presence of outliers while $p > 2$ can be used to detect outlying observations. In addition, is argued in [CCDS20] that $p = 3$ is useful for topic modelling and the authors of this work obtain random coresets in a computation model similar to ours.

We continue the study of algorithms for ℓ_p norms on streaming and distributed data. A particular novelty of our results is that unlike previous distributed and streaming algorithms, they can all be implemented *deterministically*, i.e., our algorithms make no random choices. While in a number of settings randomised algorithms are highly beneficial, leading to massive computational savings, there are other applications which require extremely high reliability, for which one needs to obtain guaranteed performance across a large number of inputs. If one were to use a randomised algorithm, then it would need vanishingly small error probability; however, many celebrated algorithms in numerical linear algebra succeed with only constant probability. Another limitation of randomised algorithms was shown in [HW13]: if the input to a randomised sketch depends on the output of a preceding algorithm using the same sketch, then the randomised sketch can give an arbitrarily bad answer. Hence, such methods cannot handle adaptively chosen inputs. Thus, while randomised algorithms certainly have their place, the issues of high reliability and adaptivity motivate the development of deterministic methods for a number of other settings, for which algorithms are scarce.

Our techniques can be viewed as a conceptual generalization of Liberty's Frequent Directions (in the 2-norm) [Lib13], which progressively computes an SVD on subsequent blocks of the input. This line of work [Lib13, GP14, GLP16, GLPW16] is the notable exception in numerical linear algebra, as it provides deterministic methods, although all such methods and their guarantees are specific to the 2-norm. Our core algorithm is similar in nature, but we require a very different technical analysis to argue that the basis transformation computed preserves the shape in the target p-norm.

Our main application is to show how large-scale regression and low rank approximation problems can be solved approximately and deterministically in the sketch and solve paradigm. The core of the summary is to find a matrix

whose rows describe well the data. Absent of any space constraints, we could perhaps compute a basis for the input matrix, but on a data stream, this will use too much working space. Therefore, we use local views of the data and resort to summarising these local pieces which can then be combined in some way to say something informative about the global dataset. The two techniques we will use are to either exploit bases computed only over small parts of the data, or finding rows with of the original matrix which have high *leverage score*. In each of these two settings, the stored rows contain a lot of information about the shape of the data. In the Euclidean norm, leverage scores correspond directly to row norms of an orthonormal basis. This is less straightforward for other ℓ_p norms, where the scores correspond to the row norms of so-called ℓ_p-well-conditioned bases. Moreover, while leverage scores are often used for sampling in randomised algorithms, we use them here in the context of fully deterministic algorithms.

Our general technique for both of the two above setups is similar: we read in a block of input, compute a local summary and either prune the summary, or first merge it with another summary before pruning. Recall from Section 2.1 that we can relate the *distributed summary model* to the single-pass *row arrival model*. Given this relationship between the streaming and distributed models, our algorithms can be viewed as having data stored over multiple machines who each send 'important' rows of their data to a central coordinator in order to compute the approximation, or as a single-pass row arrival stream.

Our first exploration is that of *subspace summaries*; roughly speaking, this approach splits the input matrix into blocks which represent the leaves of a binary tree and reduces each block to a representative basis. These bases are then merged-and-reduced over the entire binary tree. For ease of understanding, it is easiest to conceptualise this result through the lens of the distributed summary model, although it can be implemented in a single pass. We apply this technique to ℓ_p-regression and entrywise ℓ_1-low rank approximation.

Our second approach shows how a superset of rows with high leverage scores can be found for arbitrary ℓ_p norms, based on only local information. This leads to efficient algorithms which identify rows with high (local) leverage scores within subsets of the data, and proceed hierarchically to collect a sufficient set of rows. These rows then allow us to solve the ℓ_∞ regression problem: essentially, we solve the regression problem corresponding to just the retained input rows. In particular, we approximate the ℓ_∞-regression problem with additive error in a stream. Note that the ℓ_∞ problem reduces to finding a ball of minimum radius which covers the data, and global solutions are slow due to the need to solve a linear program. Instead, we show that only a subset of the data needs to be retained in the streaming model to compute accurate approximations.

4.1.2 Summary of Contributions

All our algorithms are deterministic polynomial time, and use significantly sublinear memory or communication in streaming and distributed models, respectively. We consider tall and thin $n \times d$ matrices $\mathbf{A}$ for overconstrained regression so one should think of $n \gg d$. The overall space used is the dimensionality of some summary matrix of size $m \times d$. We will always inherit dimensionality d from the original data and the task is to keep m, the number of rows stored, as small as possible. Depending on the problem, we either store the only summary, or perhaps a constant number of small summaries that are then merged and reduced to achieve the claimed space bound.

Section 4.3.1 gives a method for computing an ℓ_p-subspace embedding of a data matrix in polynomial time. The space is $O(n^\gamma) \times d$ to obtain a summary of size $n^\gamma \times d$ which achieves $d^{O(1/\gamma)}$ distortion, for $\gamma \in (0, 1)$ a small constant. This result is then applied to ℓ_p-regression which is shown to have a poly (d) approximation factor with the same amount of space.

Section 4.3.3 describes a deterministic algorithm which gives a poly (k)-approximation to the optimal low rank approximation problem in entrywise ℓ_1-norm. It runs in polynomial time for constant k. This method builds on prior work by derandomising a subroutine from [SWZ17].

Section 4.4 presents an algorithm which returns rows of high 'importance' in a data matrix with additive error. This follows by storing a polynomial number (in d) of rows and using these to compute a well-conditioned basis. The key insight here is that rows of high norm in the full well-conditioned basis cannot have their norm decrease too much in a well-conditioned basis associated with a subblock; in fact they remain large up to a multiplicative poly(d) factor.

Section 4.5 describes an algorithm for computing an additive-error solution to the ℓ_∞-regression problem, and shows a corresponding lower bound, showing that relative error solutions in this norm are not possible in sublinear space, even for randomised algorithms.

Section 4.6 concludes with an empirical evaluation for the ℓ_∞ regression problem.

4.1.3 Comparison to Related Work

There is a rich literature on algorithms for numerical linear algebra in general p-norms; most of which are randomised with the notable exception of Frequent Directions. The key contributions of our work for each of the problems considered and its relation to prior work is as follows:

Subspace embedding, regression and ℓ_1-low rank approximation: various approaches using row-sampling [CP15, DDH$^+$08], and data oblivious methods such as low-distortion embeddings can solve regression in time propor-

tional to the sparsity of the input matrix [CDMI+16, MM13, SWZ17, WZ13]. However, despite the attractive running times and error guarantees of these works, they are all randomised and do not necessarily translate well to the streaming model of computation, possibly requiring a second pass of the data to perform the sampling or global data access to evaluate the sampling distribution (see Section 2.2.1). Obtaining coresets in ℓ_p on a data stream has also been studied in [CCDS20], but again, this approach is randomised and has more restrictive conditions on p; they assume $p \geq 2$, some of their algorithms only work for integer p, and different guarantees are shown between even and odd p. Our contribution here is a fully deterministic algorithm that works for all $p \geq 1$ in both streaming and distributed models. Randomised methods for ℓ_1 low rank approximation have also been developed in [SWZ17] and our result exploits a derandomised subroutine from this work to obtain a deterministic result which applies in both models.

Finding high leverage rows: our algorithm is a single pass streaming algorithm and uses small space. We show that the global property of ℓ_p-leverage scores can be understood by considering only local statistics. Frequent Directions is the only comparable result to ours and outputs a summary in the ℓ_2-norm whose rows are *not* the datapoints but rather weighted linear combinations of the rows. However, our method covers all $p \geq 1$ and stores the datapoints exactly. Theorem 4.4.1 is the key result and is later used to prove Theorem 4.5.1 and approximate the ℓ_∞-regression problem.

4.2 Preliminaries and Notation

We consider the task of finding deterministic summaries for an input matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$. These summaries are found by computing generalisations of orthonormal bases from the Euclidean norm into arbitrary ℓ_p. Our summaries are then used for the central tasks of low rank approximation, regression and evaluating the ℓ_p-leverage scores of a matrix. We assume that $n \gg d$ so $\operatorname{rank}(\mathbf{A}) \leq d$ and the regression problems are overconstrained. Without loss of generality we may assume that the columns of the input matrix are linearly independent so that $\operatorname{rank}(\mathbf{A}) = d$.

Our focus is on the cases $1 \leq p < 2$ and $p > 2$ because the deterministic $p = 2$ case is relatively straightforward. Indeed, for $p = 2$, $\mathbf{A}^\top \mathbf{A}$ can be maintained incrementally by adding the outer products of rows of $\mathbf{A}$ as they are presented. This streaming approach allows $\mathbf{x}^\top \mathbf{A}^\top \mathbf{A}\mathbf{x} = \|\mathbf{A}\mathbf{x}\|_2^2$ to be computed exactly for any vector $\mathbf{x}$ which results in an exact ℓ_2 subspace embedding using $O(d^2)$ space and $O(nd^{\omega-1})$ time ($\omega < 2.4$ is the matrix multiplication constant [Alm21]).

Throughout this chapter we rely heavily on the notion of a *well-conditioned*

basis for the column space of an input matrix, in the context of the *Minkowski p-norm* which is $\|\mathbf{M}\|_p = (\sum_{i,j} |\mathbf{M}_{ij}|^p)^{1/p}$ as defined in Definition 2.2.1. We adopt the convention that when $p = 1$ we take the dual norm as $q = \infty$.

Definition 4.2.1 (Well-conditioned basis [DDH+08])**.** *Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ have rank d. For $p \in [1, \infty)$ let $q = \frac{p}{p-1}$ be its dual norm. An $n \times d$ matrix $\mathbf{U}$ is an (α, β, p)-well-conditioned basis for $\mathbf{X}$ if the column span of $\mathbf{U}$ is equal to that of $\mathbf{X}$,*

1. *$\|\mathbf{U}\|_p \leq \alpha$*
2. *For all $\mathbf{z} \in \mathbb{R}^d$, $\|\mathbf{z}\|_q \leq \beta \|\mathbf{U}\mathbf{z}\|_p$*

and $\alpha, \beta, d^{O(1)}$ are independent of n.

Definition 4.2.1 may also be termed a *global well-conditioned basis* when the matrix in question relates to the space spanned by all of the data present in the input. That is, $\mathbf{X} = \mathbf{A}$. The reason for this distinction is that we will often compute *local well-conditioned bases* from blocks of the input $\mathbf{A}$. In this case, we will take $\mathbf{X} = \mathbf{P}\mathbf{A}$ where $\mathbf{P} \in \mathbb{R}^{n \times n}$ acts as a row selection matrix: $\mathbf{P}_{ii} = 1$ if and only if i is a row in the observed subset of input rows, all other $\mathbf{P}_{ij} = 0$.

Theorem 4.2.1 ([DDH+08])**.** *Let $\mathbf{A}$ be an $n \times d$ matrix of rank d, let $p \in [1, \infty)$ and let q be its dual norm. There exists an (α, β, p)-well-conditioned basis $\mathbf{U}$ for the column space of $\mathbf{A}$ such that:*

1. *if $p < 2$ then $\alpha = d^{\frac{1}{p} + \frac{1}{2}}$ and $\beta = 1$,*
2. *if $p = 2$ then $\alpha = \sqrt{d}$ and $\beta = 1$, and*
3. *if $p > 2$ then $\alpha = d^{\frac{1}{p} + \frac{1}{2}}$ and $\beta = d^{\frac{1}{p} - \frac{1}{2}}$.*

Moreover, $\mathbf{U}$ can be computed in deterministic time $O(nd^2 + nd^5 \log n)$ for $p \neq 2$ and $O(nd^2)$ if $p = 2$.

We freely use the fact that a well-conditioned basis $\mathbf{U} = \mathbf{A}\mathbf{R}$ can be efficiently computed for the given data matrix $\mathbf{A}$. Details for the computation can be found in [DDH+08] but roughly this is done by computing a change of basis $\mathbf{R}$ such that $\mathbf{U} = \mathbf{A}\mathbf{R}$ is well-conditioned. Similarly, as $\mathbf{R}$ can be inverted we have the relation that $\mathbf{U}\mathbf{R}^{-1} = \mathbf{A}$. Both methods are used so we adopt the convention that $\mathbf{U} = \mathbf{A}\mathbf{R}$ when writing a well-conditioned basis in terms of the input and $\mathbf{U}\mathbf{S} = \mathbf{A}$ for the input in terms of the basis.

Our algorithms in this chapter operate in the *row arrival model* (Definition 2.1.3) and the *distributed summary model* (Definition 2.1.4). In both settings an algorithm receives as input the matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$. For a problem

P, the algorithm must keep a subset of the rows of **A** and/or some function of the rows. Upon reading the full input, we use a black-box solver to compute an approximate solution to **P** with only the subset of rows stored. In both models we measure the following:

- *Summary size*: the number of rows retained in the summary. The overall space is then $\texttt{SummarySize} \times d$ so it will suffice to bound only the number of rows in the summary.
- *Update time*: the time taken to find the local summary on every batch-update of rows.
- *Query time*: the time taken to compute an approximation to **P** using the summary after observing the entire stream.

4.2.1 Convex Optimisation

For many of the problems we study, the optimal solution can be found by using a deterministic convex optimisation solver. However, the time cost for these algorithms can be polynomial in the input size. In Table 1.1 [Bub15], one can see that there are many different methods for solving convex problems which have benefits or downsides depending on the particular problem structure. We are not concerned with which solve method is used, however, what is of concern is that each of the deterministic methods requires multiple (sub)gradient steps. There are two problems here: firstly, any gradient over the full data requires $O(nd)$ just to compute a matrix-vector product; secondly, and more pertinently for our work, such a step requires *full access to the data* which is not possible without a further pass if the data is streaming.

Given that we are interested in *small space streaming algorithms*, calling a convex optimisation routine on the full input will not be possible, let alone scalable. As a result, our algorithm design circumvents the need to use a convex solver on large matrices by only calling the optimisation procedure on small summaries of the data. This will enable us to return an estimate to the orginal optimisation problem which is computed over a small summary of the original input problem which we then argue is a reasonable approximation to the original problem.

4.3 Obtaining & Applying ℓ_p Subspace Summaries

4.3.1 Relative Error ℓ_p Subspace Embeddings

The starting point of this section is to generalise the approach of Liberty's *Frequent Directions* algorithm, as introduced in Algorithm 1 Section 2.2.2, to arbitrary ℓ_p. The error guarantee for Frequent Directions is a happy

consequence of *specific* properties from the Euclidean norm. Such properties do not necessarily apply in arbitrary ℓ_p, for instance, one difficulty is that we cannot easily exploit the SVD of a matrix. This has substantial ramifications: we cannot then write the residual term $\|\mathbf{A}-\mathbf{A}_k\|_F^2$ or write $\|\mathbf{A}\|_F^2$ as sums over the singular values of $\mathbf{A}$. Both of these properties are crucial to the analysis of Frequent Directions so we cannot follow the analysis directly. As a consequence, our results here incur greater error than the corresponding results for Frequent Directions as the only tool we have is an ℓ_p well-conditioned basis. Nevertheless, we are able to obtain a sequence of local summaries that, when combined, do not incur too much error and are thus good surrogates for reconstructing the column space of $\mathbf{A}$ in ℓ_p.

Under the assumptions of the *distributed summary model*, we present an algorithm which computes an ℓ_p-subspace embedding with multiplicative relative-error to contrast the additive-error bound of Frequent Directions. By extension, this applies to both the distributed and streaming models of computation as described in Section 2.1. Two operations are needed for this model of computation: the merge and reduce steps.

- For the merge step, successive matrices (which may themselves be summaries, subsets of the input data, or both!) $\mathbf{S}$ are concatenated until a space bound is met.

- To reduce the input at each level a summary is computed by taking an input matrix $\mathbf{X}$ that may be a block of the raw input data (corresponding to a leaf node) *or* some subsequent summaries of the input data (corresponding to a node higher up the tree) and computing a well-conditioned basis $\mathbf{U}$ so that $\mathbf{X} = \mathbf{US}$. In particular, the summary is now the matrix $\mathbf{S}$ with $\mathbf{U}$ and $\mathbf{B}$ deleted.

A further reduce step takes as input this concatenated matrix and the process is repeated. Before proceeding with our algorithm, let us first recall Definition 2.2.2 regarding a relative error ℓ_p-subspace embedding. This will be the key tool that we exploit and it roughly amounts to finding a basis which doesn't distort matrix-vector products too much compared to that in the original data.

Definition 2.2.2. *Let $\mathbf{A} \in \mathbb{R}^{n\times d}, p > 0$ and $c_1, c_2 \geq 0$ be constants. A matrix $\mathbf{T} \in \mathbb{R}^{m\times d}$ is a (c_1, c_2) ℓ_p-subspace embedding for the column space of $\mathbf{A}$ if for all $\mathbf{x} \in \mathbb{R}^d$:*

$$c_1 \|\mathbf{Ax}\|_p^p \leq \|\mathbf{Tx}\|_p^p \leq c_2 \|\mathbf{Ax}\|_p^p .$$

Our goal in this section is to find a small space basis that approximately reconstructs the column space of $\mathbf{A}$. Of course, Theorem 4.2.1 ensures that

Algorithm 3: Streaming Deterministic ℓ_p Subspace Embedding

Input: $\mathbf{A} \in \mathbb{R}^{n \times d}, p > 0, \gamma < 1$
Output: Approximate ℓ_p Subspace Embedding of $\mathbf{A}$

```
1  Function ℓ_p-SubspaceEmbedding(A, p, γ)
2      Counters m, t ← 1
3      Summaries P^(t) ← EMPTY for all t
4      for m = 1 : n^(1−γ) do
5          A_[m] = US                          ▷ U an ℓ_p wcb for A
6          if num. rows(P^(t)) + d ≤ n^γ then
7              P^(t) ← [P^(t); S]
8          else
9              P^(t+1) ← S
10             t ← t + 1
11     Merge all P^(t): T = [P^(1); ...; P^(·)]
12     Reduce T by splitting into blocks of n^γ and repeating lines (2) -
       (12) with T in place of A.
13     return T
```

such an embedding exists if we are able to pay the high time cost of $\Omega(nd^2)$ by using Theorem 4.2.1. However, naïvely applying Theorem 4.2.1 would require access to all of the data which contravenes our assumption that the input is too large to hold in its entirety. Our task is to design a scalable alternative that incurs small distortion while using only small space.

Illustrating the Algorithm

Before formalising the analysis, let us first consider our algorithmic approach. Informally, our algorithm exploits a tree structure as follows: split input $\mathbf{A} \in \mathbb{R}^{n \times d}$ into $n^{1-\gamma}$ blocks of size n^γ, these form the leaves of the tree. For each block, a well-conditioned basis is computed and the change of basis matrix $\mathbf{S} \in \mathbb{R}^{d \times d}$ is stored and passed to the next level of the tree. This is repeated until the concatenation of all the $\mathbf{S}$ matrices would exceed n^γ. At this point, the concatenated $\mathbf{S}$ matrices form the parent node of the leaves in the tree and the process is repeated upon this node: this is the merge and reduce step of the algorithm. At every iteration of the merge-and-reduce steps it can be shown that a distortion of $1/d$ is introduced by using the summaries $\mathbf{S}$. However, this can be controlled across all of the $O(1/\gamma)$ levels in the tree to give a deterministic relative error ℓ_p subspace embedding which requires only sublinear space and little communication.

The pseudocode for the first level of the tree structure of the deterministic ℓ_p subspace embedding is given in Algorithm 3. We use the following notation: m is a counter to index the block of input currently held, denoted $\mathbf{A}_{[m]}$, and

ranges from 1 to $n^{1-\gamma}$ for the first level of the tree. Similarly, t indexes the current summary, $\mathbf{P}^{(t)}$ which are all initialized to be an empty matrix. We use the notation $[\mathbf{X}; \mathbf{Y}]$ to denote the row-wise concatenation of two matrices $\mathbf{X}$ and $\mathbf{Y}$ with equal column dimension.

Note that Algorithm 3 can be easily distributed as any block of sublinear size can be given to a compute node and then a small-space summary of that block is returned to continue the computation. In addition, the algorithm can be performed using sublinear space in the streaming model because at any one time a summary $\mathbf{T}$ of the input matrix whose size is at most $n^{\gamma} \times d$ can be computed and $\mathbf{T}$ is of size $d \times d$. Upon reading $\mathbf{A}_{[1]}$, a small space summary $\mathbf{P}^{(1)}$ is computed and stored with the algorithm proceeding to read in $\mathbf{A}_{[2]}$. Similarly, the summary $\mathbf{P}^{(2)}$ is computed and if $[\mathbf{P}^{(1)}; \mathbf{P}^{(2)}]$ does not exceed the storage bound, then the two summaries are *merged* (which in this context is simple row-wise concatentation) and this process is repeated until at some point the storage bound is met. Once the summary is large enough that it meets the storage bound, it is then *reduced* by performing the well-conditioned basis reduction (line (5)) and the reduced summary is stored with the algorithm continuing to read and summarize input until a corresponding block in the tree is obtained (or the blocks can be combined to terminate the algorithm).

Relative Error Embedding

Before presenting the main result, we show a simple bound on the distortion of a vector under the action of a well-conditioned basis. This is a consequence of the structural properties of well-conditioned bases as introduced in [DDH$^+$08].

Lemma 4.3.1. *Let $\mathbf{x} \in \mathbb{R}^d$ be arbitrary and suppose that $\mathbf{U}$ is an ℓ_p well-conditioned basis for the column span of an input matrix $\mathbf{A}$ with $p > 1$ and $p \neq 2$. Then*

$$\|\mathbf{x}\|_p \leq \|\mathbf{U}\mathbf{x}\|_p \leq d \|\mathbf{x}\|_p .$$

Proof. The main property we need is from [DDH$^+$08]:

$$\|\mathbf{x}\|_2 \leq \|\mathbf{U}\mathbf{x}\|_p \leq \sqrt{d} \|\mathbf{x}\|_2 . \tag{4.1}$$

Now, we split into two cases (i) $p < 2$ and (ii) $p > 2$. For the first case we have $\|\mathbf{x}\|_2 < \|\mathbf{x}\|_p$ so

$$\begin{aligned} \|\mathbf{U}\mathbf{x}\|_p &\leq \sqrt{d} \|\mathbf{x}\|_2 \\ &\leq \sqrt{d} \|\mathbf{x}\|_p . \end{aligned}$$

For the lower bound, we use Hölder's inequality to deduce that $\|\mathbf{x}\|_p / \sqrt{d} < \|\mathbf{x}\|_2$ so multiplying through by $\sqrt{d}$ achieves the stated result. When $p > 2$ we apply

a similar argument, using (4.1) for the lower bound and modifying Hölder's inequality to obtain:

$$\|\mathbf{x}\|_p \leq \|\mathbf{x}\|_2 \leq \|\mathbf{U}\mathbf{x}\|_p \leq \sqrt{d}\,\|\mathbf{x}\|_2 \leq d\,\|\mathbf{x}\|_p\,.$$

□

We are now in a position to prove the correctness of Algorithm 3.

Theorem 4.3.1. *Let $\mathbf{A} \in \mathbb{R}^{n\times d}$, $p \neq 2, \infty$ be fixed and fix a constant $\gamma \in (0,1)$. Then there exists a one-pass deterministic algorithm which constructs a $(1/d^{O(1/\gamma)}, 1)$ relative error ℓ_p-subspace embedding with $O(n^\gamma d^2 + n^\gamma d^5 \log n^\gamma)$ update time and $O(n^\gamma d)$ space in the distributed summary model of computation.*

Proof. Let $\mathbf{A} \in \mathbb{R}^{n\times d}$ and $\mathbf{B} \in \mathbb{R}^{n^\gamma \times d}$ be an arbitrary block of input from $\mathbf{A}$. We compute an ℓ_p well-conditioned basis for $\mathbf{B}$ in time poly $(n^\gamma d)$ by Theorem 4.2.1; so let $\mathbf{B} = \mathbf{U}\mathbf{S}$ for $\mathbf{U} \in \mathbb{R}^{n^\gamma \times d}$ and $\mathbf{S} \in \mathbb{R}^{d\times d}$ a change of basis matrix. Upon computing $\mathbf{U}$ and $\mathbf{S}$, Algorithm 3 subsequently deletes $\mathbf{U}$ and retains only $\mathbf{S}$. We apply Lemma 4.3.1 on the vector $\mathbf{S}\mathbf{x}$ which readily achieves:

$$\|\mathbf{S}\mathbf{x}\|_p \leq \|\mathbf{U}\mathbf{S}\mathbf{x}\|_p \leq d\,\|\mathbf{S}\mathbf{x}\|_p\,.$$

Recalling that $\mathbf{B} = \mathbf{U}\mathbf{S}$ and by rearranging, we have thus found $\mathbf{S}$ satisfying:

$$\frac{\|\mathbf{B}\mathbf{x}\|_p}{d} \leq \|\mathbf{S}\mathbf{x}\|_p \leq \|\mathbf{B}\mathbf{x}\|_p\,.$$

In particular, $\|\mathbf{S}\mathbf{x}\|_p$ agrees with $\|\mathbf{B}\mathbf{x}\|_p$ up to a distortion factor of d and is a $(1/d, 1)$ ℓ_p-subspace embedding for the column space of $\mathbf{B}$.

It remains to understand how this approach propagates over the entirety of $\mathbf{A}$. Algorithm 3 applies the *merge and reduce* framework. The matrix $\mathbf{A}$ is seen a row at a time and n^γ rows are stored which are used to construct a tree. So at every level a subspace embedding with distortion d is constructed. This error propagates through each of the $O(1/\gamma)$ levels in the tree so the overall distortion to construct the subspace embedding for $\mathbf{A}$ is $d^{O(1/\gamma)}$. The space bound is similar; we need $n^\gamma d$ storage per group so require $O(1/\gamma)n^\gamma d$ overall. The merge operation is the concatentation of successive change of basis matrices $\mathbf{S}$ while the reduce operation is the compression of a larger block into its representative basis matrix $\mathbf{S}$. □

4.3.2 Application: ℓ_p Regression

Subspace embeddings are useful as they roughly preserve the shape of the data while being relatively cheap to compute compared to solving a large-scale problem. While this is itself useful, their utility extends further and we show

how to use the approach of Section 4.3.1 to approximate crucial numerical linear algebra problems. The first application is to show how the subspace embedding of Theorem 4.3.1 can be used to achieve a deterministic relative-error approximate regression result. The proof proceeds similarly to Theorem 4.3.1 and relies upon analyzing the merge-and-reduce behaviour across all nodes of the tree.

ℓ_p-Regression Problem: Given matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ and target vector $\mathbf{b} \in \mathbb{R}^n$, find $\mathbf{x}^\star = \operatorname{argmin}_x f(\mathbf{x})$ where $f(\mathbf{x}) = \|\mathbf{A}\mathbf{x}-\mathbf{b}\|_p$. Solving ℓ_p regression for $p \neq 2$ is expensive as it relies on convex programming so we will relax the problem and settle for evaluating an estimate $\hat{\mathbf{x}}$ which approximates the *objective value* or *cost* of the regression problem as $f(\mathbf{x}^\star) \leq f(\hat{\mathbf{x}}) \leq \Delta f(\mathbf{x}^\star)$ for some $\Delta > 1$ which is to be understood through the method of approximation that we select. Of course, by optimality of $\mathbf{x}^\star$ we are guaranteed that *any* estimation $\hat{\mathbf{x}}$ satisfies $f(\mathbf{x}^\star) \leq f(\hat{\mathbf{x}})$, thus it suffices to focus on the upper bound and this is the focus of the subsequent theorem.

Theorem 4.3.2. *Let $\mathbf{A} \in \mathbb{R}^{n\times d}, \mathbf{b} \in \mathbb{R}^n$, fix $p \neq 2, \infty$ and a constant $\gamma > 0$. The* ℓ_p-regression problem *can be solved deterministically in the streaming and distributed models with a $(d+1)^{O(1/\gamma)} = \operatorname{poly}(d)$ relative error approximation factor. The algorithm's update time is $\operatorname{poly}(n^\gamma(d+1))$ and uses $O((1/\gamma)n^\gamma(d+1))$ working space. The query time is $\operatorname{poly}(n^\gamma)$ for the cost of convex optimization.*

Proof. The task is to minimise $\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_p$. Let $\mathbf{Z} = [\mathbf{A}, \mathbf{b}] \in \mathbb{R}^{n\times(d+1)}$ and compute a subspace embedding $\mathbf{S}$ for $\mathbf{Z}$ using Theorem 4.3.1. Note that $\mathbf{S}$ has $O((1/\gamma)n^\gamma)$ rows. Let $\Delta = (d+1)^{O(1/\gamma)}$, then for all $\mathbf{y} \in \mathbb{R}^{d+1}$ we have:

$$\frac{\|\mathbf{Z}\mathbf{y}\|_p}{\Delta} \leq \|\mathbf{S}\mathbf{y}\|_p \leq \|\mathbf{Z}\mathbf{y}\|_p . \tag{4.2}$$

Since this condition holds for all $\mathbf{y} \in \mathbb{R}^{d+1}$ it must hold, in particular, for vectors $\mathbf{y}' = (\mathbf{x}, -1)^\top$ where $\mathbf{x} \in \mathbb{R}^d$ is arbitrary. However, observe that:

$$\left\|\mathbf{Z}\mathbf{y}'\right\|_p = \left\|[\mathbf{A}, \mathbf{b}] \begin{bmatrix}\mathbf{x}\\ -1\end{bmatrix}\right\|_p = \|\mathbf{A}\mathbf{x}-\mathbf{b}\|_p . \tag{4.3}$$

Denote the first d columns of $\mathbf{S}$ by $\mathbf{S}_{[1:d]}$ and the last column by $\mathbf{S}_{[d+1]}$. Then

$$\left\|\mathbf{S}\mathbf{y}'\right\|_p = \left\|[\mathbf{S}_{[1:d]}, \mathbf{S}_{[d+1]}] \begin{bmatrix}\mathbf{x}\\ -1\end{bmatrix}\right\|_p = \|\mathbf{S}_{[1:d]}\mathbf{x} - \mathbf{S}_{[d+1]}\|_p. \tag{4.4}$$

Now we have transformed the subspace embedding relationship into an instance of regression. In particular, $\mathbf{S}_{[1:d]}$ has only $O((1/\gamma)n^\gamma)$ rows so is a smaller instance than the original problem. We now focus on the task

of finding $\min_{\mathbf{x}\in\mathbb{R}^d} \left\|\mathbf{S}_{[1:d]}\mathbf{x} - \mathbf{S}_{[d+1]}\right\|_p$. By using (4.2) with $\mathbf{y}'$ and utilising Equations (4.3) and (4.4) we have:

$$\frac{\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_p}{\Delta} \leq \left\|\mathbf{S}_{[1:d]}\mathbf{x} - \mathbf{S}_{[d+1]}\right\|_p \leq \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_p . \tag{4.5}$$

Convex optimisation can now be used to find $\min_{\mathbf{x}\in\mathbb{R}^d} \left\|\mathbf{S}_{[1:d]}\mathbf{x} - \mathbf{S}_{[d+1]}\right\|_p$. Let $\hat{\mathbf{x}} = \operatorname{argmin}_{\mathbf{x}\in\mathbb{R}^d} \left\|\mathbf{S}_{[1:d]}\mathbf{x} - \mathbf{S}_{[d+1]}\right\|_p$ which is output from the optimisation and let $\mathbf{x}^\star = \operatorname{argmin}_{\mathbf{x}\in\mathbb{R}^d} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_p$ be the optimal solution we would like to estimate. By optimality of $\hat{\mathbf{x}}$ in the small-space instance we have:

$$\left\|\mathbf{S}_{[1:d]}\hat{\mathbf{x}} - \mathbf{S}_{[d+1]}\right\|_p \leq \left\|\mathbf{S}_{[1:d]}\mathbf{x}^\star - \mathbf{S}_{[d+1]}\right\|_p . \tag{4.6}$$

However, combining Equation (4.6) with Equation (4.5) we see that:

$$\begin{aligned}\frac{\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_p}{\Delta} &\leq \left\|\mathbf{S}_{[1:d]}\hat{\mathbf{x}} - \mathbf{S}_{[d+1]}\right\|_p \\ &\leq \left\|\mathbf{S}_{[1:d]}\mathbf{x}^\star - \mathbf{S}_{[d+1]}\right\|_p \\ &\leq \|\mathbf{A}\mathbf{x}^\star - \mathbf{b}\|_p .\end{aligned}$$

Therefore, the ℓ_p-regression problem has been approximated with $\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_p \leq \Delta \|\mathbf{A}\mathbf{x}^\star - \mathbf{b}\|_p$ and $\Delta = \operatorname{poly}(d+1)$. The overall time complexity is the time taken to compute the subspace embedding, which is $\operatorname{poly}(nd)$ ($\operatorname{poly}(n^\gamma d)$ update time repeated over at least $n^{1-\gamma}$ leaves in the computation tree) by Theorem 4.3.1, and the time for the convex optimisation. However, the optimisation costs $\operatorname{poly}(O(1/\gamma)n^\gamma)$ [BBV04] which is subsumed by the dominant time cost for computing the embedding. Finally, the space cost is immediate from computing the subspace embedding in Theorem 4.3.1. □

4.3.3 Application: ℓ_1-Low Rank Approximation

One further application of our deterministic subspace embeddings is to approximately solve the ℓ_1-Low Rank Approximation Problem. The ℓ_1 version of low rank approximation is more robust than the standard Frobenius version in the presence of outliers and is useful if Gaussianity assumptions on the data do not apply. Unfortunately, it was shown in [GV18] that the solving ℓ_1-low rank approximation exactly is NP-hard. The first provable (randomised) approximation algorithms for this problem were given in [SWZ17] which improved over many prior heuristics. Our result is the first deterministic streaming algorithm for approximations to this problem.

ℓ_1-Low Rank Approximation Problem: Given matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$,

output a matrix $\mathbf{T}$ of rank k such that for constant k:

$$\|\mathbf{A} - \mathbf{T}\|_1 \leq \text{poly}(k) \min_{\mathbf{A}':\text{rank}(k)} \left\|\mathbf{A} - \mathbf{A}'\right\|_1 . \tag{4.7}$$

The key technique is similar to that of the previous section by using a tree structure with merge-and-reduce operations. For input $\mathbf{A} \in \mathbb{R}^{n\times d}$ and constant $\gamma > 0$ partition $\mathbf{A}$ into $n^{1-\gamma}$ groups of rows which form the leaves of the tree. The tree is defined as previously with the same 'merge' operation, but the 'reduce' step to summarize the data exploits a derandomisation (subroutine Algorithm 4) of [SWZ17] to compute an approximation to the optimal ℓ_1-low rank approximation. Once this is computed, k of the rows in the summary are kept for later merge steps.

This process is continued with the successive k rows from n^γ rows being 'merged' or added to the matrix until it has n^γ rows. The process is repeated across all of the groups in the level and again on the successive levels on the tree from which it can be shown that the error does not propagate too much over the tree, thus giving the desired result.

A Useful Subroutine

Our algorithm for low rank approximation will operate in a similar fashion as the subspace embedding algorithm from Section 4.3.1. In the same vein, we will need a subroutine that operates on smaller blocks at every level of the computation tree whose error we can bound. To this end, we introduce Algorithm 4 which is a derandomised version of an algorithm [SWZ17] which returns a low rank approximation to an input matrix. The derandomisation follows from generating and testing all possible combinations of the necessary matrices.

Lemma 4.3.2. *On an input of size $N\times D$, Algorithm 4 runs in time* $\text{poly}(ND)$.

Proof. Every matrix which is generated in Algorithm 4 has a number of nonzero entries bounded by $O(k\text{polylog}(k))$. We can test all of the matrices (line 6) in time proportional to the dimension of the matrix (N or D) with exponent $O(k\text{polylog}(k))$ resulting in time $\text{poly}(ND)$ overall, since k is constant. □

We need one further lemma which describes the approximation error induced by using well-conditioned bases to decompose a matrix.

Lemma 4.3.3. *Let $\mathbf{M} \in \mathbb{R}^{N\times D}$ have rank ρ and suppose $\mathbf{U} \in \mathbb{R}^{N\times\rho}$ is a well-conditioned basis for $\mathbf{M}$. Let $\mathbf{M} = \mathbf{US}$ for a change of basis $\mathbf{S} \in \mathbb{R}^{\rho\times D}$. Then for all $\mathbf{x} \in \mathbb{R}^D$:*

$$\frac{\|\mathbf{Sx}\|_1}{\text{poly}(D)} \leq \|\mathbf{Mx}\|_1 \leq \text{poly}(D) \|\mathbf{Sx}\|_1 .$$

Algorithm 4: Derandomised ℓ_1 low rank approximation of [SWZ17]

Input: $\mathbf{A} \in \mathbb{R}^{n\times d}, p > 0, \gamma < 1$
Output: ℓ_1 Low Rank Approximation of $\mathbf{A}$

1 **Function** `L1kRankApproximation`$(\mathbf{A}, n, d, k)$
2 $r = O(k \log k), m = O(r \log r), t_1 = O(r \log r), t_2 = O(m \log m)$
3 Generate all diagonal $\mathbf{R} \in \mathbb{R}^{d\times d}$ with only r 1s
4 Compute all possible sampling and rescaling matrices $\mathbf{D}, \mathbf{T}_1 \in \mathbb{R}^{n\times n}$ corresponding to Lewis Weights of $\mathbf{AR}$ whose entries are powers of 2 between 1 and $1/nd$. There are m and t_1 nonzero entries on the diagonal, respectively.
5 Compute all sampling and rescaling matrices $\mathbf{T}_2^\top \in \mathbb{R}^{d\times d}$ according to the Lewis weights of $(\mathbf{DA})^\top$ with t_2 nonzero entries, powers of 2 between 1 and $1/nd$ on the diagonal.
6 Evaluate $\|\mathbf{T}_1\mathbf{ARXYDAT}_2 - \mathbf{T}_1\mathbf{AT}_2\|_1$ for all choices of above matrices.
7 **return** $\mathbf{ARX}, \mathbf{YDA}$ *that minimise line 6*

Proof. For the left-hand side we can just calculate:

$$\begin{aligned}\|\mathbf{Sx}\|_1 &\leq D \cdot \|\mathbf{Sx}\|_\infty \\ &\leq D \cdot \text{poly}\,(D)\,\|\mathbf{USx}\|_1 \\ &= \text{poly}\,(D) \cdot \|\mathbf{Mx}\|_1 .\end{aligned}$$

The second inequality follows from Definition 4.2.1, property 2 of the well-conditioned basis $\mathbf{U}$ meanwhile the claimed result follows from observing:

$$\begin{aligned}\|\mathbf{Mx}\|_1 = \|\mathbf{USx}\|_1 &\leq \|\mathbf{U}\|_1 \|\mathbf{Sx}\|_\infty \\ &= \text{poly}(D)\,\|\mathbf{Sx}\|_1 .\end{aligned}$$

□

The Main Algorithm

We are now in a position to present our main algorithm which returns a (global) approximate solution to the ℓ_1 low rank approximation problem. This contrasts Algorithm 4 which will be used to find approximate local solutions that are collated to form the global solution. Correctness of this algorithm is established in Theorem 4.3.3. It is enough to show that for every level, the low rank approximation of each group is polynomially bounded by k in error. The result follows by reasoning how this error grows as we progress through the tree. Denote the jth block of $\mathbf{A}$ by $\mathbf{A}_{[j]}$. In Algorithm 5 we illustrate the first level of the tree, as was done for Algorithm 3.

Algorithm 5: Deterministic ℓ_1 low rank approximation on a stream

Input: $\mathbf{A} \in \mathbb{R}^{n \times d}, k, \gamma < 1$
Output: Estimated ℓ_1 low rank approximation for $\mathbf{A}$

Function `L1kRankApproximation`$(\mathbf{A}, k, \gamma)$
  Counters $m, t \leftarrow 1$
  Summaries $\mathbf{P}^{(t)} \leftarrow$ EMPTY for all t
  for $m = 1 : n^{1-\gamma}$ **do**
    $\mathbf{W}, \mathbf{V}^\top =$ `L1kRankApproximation`$(\mathbf{A})$ ▷ Algorithm 4
    $\mathbf{B} \leftarrow \mathbf{W}\mathbf{V}^\top$ ▷ nb. $\mathbf{B}$ is output in factored form from Algorithm 4
    Set $\mathbf{W} = \mathbf{U}\mathbf{S}$ for well-conditioned basis $\mathbf{U}$
    if *num. rows*$(\mathbf{P}^{(t)}) + d \leq n^\gamma$ **then**
      $\mathbf{P}^{(t+1)} \leftarrow [\mathbf{P}^{(t)}; \mathbf{S}\mathbf{V}^\top]$ ▷ Merge step
    else
      $\mathbf{P}^{(t+1)} \leftarrow \mathbf{S}\mathbf{V}^\top$
      $t \leftarrow t + 1$
  Merge all $\mathbf{P}^{(t)}$: $\mathbf{T} = [\mathbf{P}^{(1)}; \ldots; \mathbf{P}^{(\cdot)}]$
  Reduce $\mathbf{T}$ by splitting into blocks of n^γ and repeating lines (2) - (12) with $\mathbf{T}$ in place of $\mathbf{A}$.
  Set $\mathbf{P}$ to be matrix of final k rows
  Solve $\min_{\mathbf{Q}} \|\mathbf{Q}\mathbf{P} - \mathbf{A}\|_1$
  return $\mathbf{Q}\mathbf{P}$

Theorem 4.3.3. *Let $\mathbf{A} \in \mathbb{R}^{n \times d}$ be the given data matrix and k be the (constant) target rank. Let $\gamma > 0$ be an arbitrary (small) constant. Algorithm 5 is a deterministic & streaming algorithm which outputs a solution to the ℓ_1-Low Rank Approximation Problem with relative error* $\text{poly}(k)$*, update time* $\text{poly}(n, d)$*, space bounded by n^γpoly(d), and query time* $\text{poly}(n, d)$*.*

Proof. For every level in the tree we can take a group of rows, $\mathbf{C}$, and perform Algorithm 4. For every $\mathbf{C}$ used as input to Algorithm 4, a k-rank matrix $\mathbf{B}$ of dimensions $n^\gamma \times d$ is returned. In particular, $\mathbf{B}$ has the following property:

$$\|\mathbf{C} - \mathbf{B}\|_1 \leq \text{poly}(k) \min_{\mathbf{B}' : \text{rank}(\mathbf{B}') = k} \left\|\mathbf{C} - \mathbf{B}'\right\|_1.$$

Now factor $\mathbf{B}$ using a k rank decomposition. That is, set $\mathbf{B} = \mathbf{W}\mathbf{V}^\top$ where $\mathbf{W}$ has k columns and $\mathbf{V}^\top$ has k rows. Further decompose $\mathbf{W}$ as $\mathbf{W} = \mathbf{U}\mathbf{S}$ for a well-conditioned basis $\mathbf{U}$. Note that $\mathbf{W}$ is $n^\gamma \times k$ (and of rank k) by the rank decomposition so $\mathbf{U}$ is also $n^\gamma \times k$ and $\mathbf{S}$ is $k \times k$. The dimensions of these matrices ensure that individually they do not exceed the space budget from the theorem.

Apply Lemma 4.3.3 with $\mathbf{W}$ and k. Then we have for every $\mathbf{x} \in \mathbb{R}^k$ that $\|\mathbf{S}\mathbf{x}\|_1 = \text{poly}(k) \|\mathbf{W}\mathbf{x}\|_1$. Since $\mathbf{U}$ is n^γ by k and $k < \text{poly}(d)$, $\mathbf{U}$ remains

within the required space bound when we use it for the calculation. Now delete $\mathbf{U}$ and store $\mathbf{SV}^\top$. Note that each $\mathbf{SV}^\top$ is a matrix of k directions in $\mathbb{R}^d$. Pass $\mathbf{SV}^\top$ to the next level of the tree.

Merge the $\mathbf{SV}^\top$ for each group until we have a matrix of n^γ rows. Repeat the process over all $O(1/\gamma)$ levels in the tree. We require $n^\gamma d$ storage for every group so as we merge and pass $\mathbf{SV}^\top$ down the levels this combines to total space of $O((1/\gamma)n^\gamma \operatorname{poly}(d))$. This part of the algorithm is a repeated use of Algorithm 4 which is $\operatorname{poly}(n^\gamma d)$ by Lemma 4.3.2 and some further lower time cost manipulations. Repeating these steps over the entire tree gives $\operatorname{poly}(nd)$ as the overall time complexity.

When this is done over all levels we will again have k directions in $\mathbb{R}^d$. Let $\mathbf{P}$ be the matrix with these directions as rows. Then we claim that $\mathbf{P}$ can be used to construct our approximate ℓ_1 low rank approximation.

Claim 4.3.1. *Let* $\mathbf{P}$ *be as described above. Then there exists* $\mathbf{QP}$ *which is an* ℓ_1 *low rank approximation for* $\mathbf{A}$*:*

$$\min_{\mathbf{Q}} \|\mathbf{QP} - \mathbf{A}\|_1 \leq \operatorname{poly}(k) \left\|\mathbf{A} - \mathbf{A}'\right\|_1.$$

Proof. Each use of Algorithm 4 admits a $\operatorname{poly}(k)$ approximation at every level of the tree. Every time the well-conditioned basis $\mathbf{U}$ is constructed and then ignored we admit a further $\operatorname{poly}(k)$ error due to Definition 4.2.1, Property 1. The distortion is blown up by a factor of $\operatorname{poly}(k)$ every time we use Lemma 4.3.3 which is at every level in the tree. Hence, the total contribution of using Algorithm 4 is $\operatorname{poly}\left(k^{O(1/\gamma)}\right) = \operatorname{poly}(k)$ for constant γ. □

Claim 4.3.1 proves the approximation factor is $\operatorname{poly}(k)$ as required. By Lemma 4.3.2 we know that Algorithm 4 is $\operatorname{poly}(n^\gamma d)$ time. The most costly steps in Algorithm 5 are invocations of Algorithm 4 so combining this over the entire tree the overall time cost is $\operatorname{poly}(nd)$ as claimed, proving the theorem. □

4.4 Leverage Score Summaries in ℓ_p-Norm

Thus far, we have primarily been interested in using the change of basis matrices to construct a summary of input $\mathbf{A}$. Notably, this will return rows in the necessary subspace that approximately reconstruct the column space of $\mathbf{A}$. An alternative approach is to directly sample *observations* of the data rather than rows in the subspace. A benefit of this approach is that if the rows have semantic meaning, these relationships are not tampered with in the summarisation phase. Secondly, if data is sparse, then maintaining the rows of $\mathbf{A}$ directly preserves sparsity. In order to maintain such summaries, we will

need a slightly different approach although the well-conditioned basis remains our key tool.

Throughout this section we will repeatedly need the notion of *ℓ_p-leverage scores* in different contexts as we are concerned with finding rows of high leverage from a matrix with respect to various p-norms. These are ℓ_p analogues of the ℓ_2-leverage scores from Definition 2.2.2 which arise as a consequence of the ℓ_p-well conditioned bases that were introduced in Section 4.2. We conclude the section with an algorithm that returns rows of high leverage up to polynomial additive error. Before introducing these we will give a general definition to highlight the differences between the scores that are computed.

Definition 4.4.1. *Let $\mathbf{A} \in \mathbb{R}^{n \times d}$ be the input matrix and suppose that only a subset $I \subseteq [n]$ of the rows are stored. Let $\mathbf{P} \in \mathbb{R}^{n \times n}$ be a row selector matrix with $\mathbf{P}_{ii} = 1$ if and only if row $i \in I$, otherwise $\mathbf{P}_{ij} = 0$. Suppose that $\mathbf{R}$ is a change of basis matrix such that $\mathbf{PAR}$ is a well-conditioned basis for the column space of $\mathbf{PA}$. The ℓ_p-leverage scores of $\mathbf{PA}$ are defined as $w_i(\mathbf{PA}) = \|\mathbf{e}_i^T \mathbf{PAR}\|_p^p$.*

This definition is deliberately general so that we can make the following distinction between *global* and *local* leverage scores.

Definition 4.4.2 (Global ℓ_p-Leverage Scores)**.** *Suppose that $I = [n]$ so $\mathbf{P} = \mathbf{I}_n$ in Definition 4.4.1. Then the leverage scores $w_i(\mathbf{A})$ are called the* global ℓ_p-leverage scores.

Definition 4.4.3 (Local ℓ_p-Leverage Scores)**.** *Suppose that $I \subset [n]$ so $\mathbf{P} \neq \mathbf{I}_n$ in Definition 4.4.1 and $\mathbf{PA} = \mathbf{B}$ is a* block subset *of the input $\mathbf{A}$. Then the leverage scores $w_i(\mathbf{B})$ are called the* local ℓ_p-leverage scores of $\mathbf{A}$ (with respect to $\mathbf{B}$).

In the prose, we might abuse notation and refer to $\mathbf{B} = \mathbf{PA} \in \mathbb{R}^{n \times d}$ as the summary even though it is defined over n rows. Note that we easily recover the 'stored' summary by removing the rows of all zeros, which we will also refer to as $\mathbf{B} \in \mathbb{R}^{m \times d}$. This is fairly straightforward through compressing $\mathbf{P}$ into its implied representation over $\{0,1\}^{m \times n}$ by again removing any rows that are entirely zero. On the other hand, we will be precise which formulation is being used in the mathematics.

Clearly, the matrix $\mathbf{R}$ in Definitions 4.4.2 and 4.4.3 may change considerably under the row selection induced by $\mathbf{P}$. Note that w_i depends both on $\mathbf{A}$ and the choice of $\mathbf{R}$, but we suppress this dependence in our notation. Next we present some basic facts about the ℓ_p leverage scores.

Our first result has a similar flavour to many other results that use leverage scores in that we bound the total sum of all scores [CMM17, CMP16, CP15]. The purpose of this result is to understand the how many rows can have

'large' contribution to the sum of all leverage scores, and thus are important in composing this sum.

Lemma 4.4.1. *Let $\mathbf{A} \in \mathbb{R}^{n\times d}$ and $\mathbf{R}$ be a change of basis so that $\mathbf{AR}$ is an (α, β, p) well conditioned basis. Let $(w_i)_{i=1}^n$ be the* global *ℓ_p-leverage scores for $\mathbf{A}$ and let $\tau \in (0,1)$. Then:*

1. *there are at most $\text{poly}(d)/\tau$ rows i of $\mathbf{A}$ that have $w_i \geq \tau\|\mathbf{AR}\|_p^p$*
2. *Let $\mathbf{x} \in \mathbb{R}^d$ be arbitrary. If a row i contributes at least a τ fraction to $\|\mathbf{ARx}\|_p^p$, then $\tau \leq w_i\beta$.*

Proof. **Part 1.** Definition 4.2.1 states that $\sum_i w_i = \sum_i \left\|\mathbf{e}_i^\top \mathbf{AR}\right\|_p^p \leq \alpha^p$. Theorem 4.2.1 shows $\alpha = \text{poly}(d)$. Define $I = \{i \in [n] : w_i > \tau\|\mathbf{AR}\|_p^p\}$ to be the index set of all rows whose ℓ_p leverage exceeds a τ fraction of $\|\mathbf{AR}\|_p^p$, then: $\alpha^p \geq \sum_i w_i \geq \sum_{i\in I} w_i \geq |I| \cdot \tau\|\mathbf{AR}\|_p^p$. Hence, $|I| \leq \alpha^p/(\tau\|\mathbf{AR}\|_p^p)$ so $|I| \leq \text{poly}(d)/\tau$ meaning there are at most $\text{poly}(d)/\tau$ rows i for which $w_i \geq \tau\|\mathbf{AR}\|_p^p$.

Part 2. Definition 4.2.1 and Hölder's inequality show that for any vector $\mathbf{x}$ we have $|(\mathbf{ARx})_i|^p \leq \beta\|\mathbf{e}_i^\top \mathbf{AR}\|_p^p \cdot \|\mathbf{ARx}\|_p^p$. Then

$$\begin{aligned}\tau &\leq |\mathbf{e}_i^\top \mathbf{ARx}|^p / \|\mathbf{ARx}\|_p^p \\ &\leq \beta w_i.\end{aligned}$$

From this we deduce that if a row contributes at least a τ fraction of $\|\mathbf{ARx}\|_p^p$ then $\tau \leq w_i\beta$. That is, $\tau \leq w_i$ for $p \in [1,2]$ and $\tau \leq d^{1/2}w_i$ for $p \in (2,\infty)$ by using Theorem 4.2.1. □

4.4.1 Relating Local and Global Leverage Scores

Before delving into this section, it will be helpful to consider the following example. In Section 4.2, we saw that $\mathbf{A}^\top\mathbf{A}$ is an exact subspace embedding for $\mathbf{A}$ costing $O(d^2)$ space. Thus, this example is not intended to be formalised but will help communicate the ideas of our subsequent approach.

A Brief Aside on ℓ_2

Suppose that we are presented $\mathbf{A} \in \mathbb{R}^{n\times d}$ in the row-arrival model and tasked with finding all rows whose ℓ_2 leverage exceeds the threshold τ. If we take a block $\mathbf{B}$ of whose rows are a strict subset of those from $\mathbf{A}$, then we are guaranteed that under the *Löwner ordering*[1] over symmetric positive semidefinite matrices, $\mathbf{B}^\top\mathbf{B} \preceq \mathbf{A}^\top\mathbf{A}$.

[1] To be formally introduced in Section 2.2.2, Fact 6.2.1. The notation $\mathbf{B}^\top\mathbf{B} \preceq \mathbf{A}^\top\mathbf{A}$ means $\mathbf{x}^\top\mathbf{Bx} \leq \mathbf{x}^\top\mathbf{Ax}$ for all vectors $\mathbf{x}$.

Assuming for now that $\mathbf{B}$ is full rank, we may then deduce that

$$\left(\mathbf{A}^\top \mathbf{A}\right)^{-1} \preceq \left(\mathbf{B}^\top \mathbf{B}\right)^{-1}. \tag{4.8}$$

In ℓ_2, one nice formulation of the leverage scores of $\mathbf{A}$ is that for a row $\mathbf{A}_i \in \mathbb{R}^{1\times d}$, its leverage score is $w_i = \mathbf{A}_i \left(\mathbf{A}^\top \mathbf{A}\right)^{-1} \mathbf{A}_i^\top$. By writing the SVD of $\mathbf{A}$, it can be shown that this is analagous to our definition of ℓ_p leverage over the ℓ_2 well-conditioned basis $\mathbf{U}$ for $\mathbf{A}$. Another nice consequence of Equation (4.8) is that it preserves the ordering of matrix-vector products, for all $\mathbf{u} \in \mathbb{R}^d$:

$$\mathbf{u}^\top \left(\mathbf{A}^\top \mathbf{A}\right)^{-1} \mathbf{u} \preceq \mathbf{u}^\top \left(\mathbf{B}^\top \mathbf{B}\right)^{-1} \mathbf{u}. \tag{4.9}$$

Applying (4.9) with $\mathbf{u} = \mathbf{A}_i^\top$ we have the relation $w_i \leq \hat{w}_i$ between global leverage scores w_i computed over $\left(\mathbf{A}^\top \mathbf{A}\right)^{-1}$ and local leverage scores $\hat{w}_i$ computed with respect to $\left(\mathbf{B}^\top \mathbf{B}\right)^{-1}$. This shows that local leverage scores in ℓ_2 are lower bounded by the true global leverage scores, the latter being expensive to obtain as $O(nd^2)$ time for computing $\mathbf{A}^\top \mathbf{A}$. Similarly, it also shows that initialising $\mathbf{B}^\top \mathbf{B} = \mathbf{0}_{d\times d}$ and updating by the outer products of rows, the leverage scores of an individual row will always decrease as new rows are observed.

Consequently, we could buffer m rows of $\mathbf{A}$ at a time and update $\mathbf{A}^\top \mathbf{A}$ by outer products at every stage which would allow us to compute local leverage scores with respect to the matrix seen thus far. Because the leverage scores in ℓ_2 are non-increasing as new rows are added, we could prune out rows from the stored set whose leverage is too small following each batch update. Eventually, we would retain a set of rows that have leverage exceeding the threshold τ before the final buffer. Upon receiving the final block of rows, we would have exactly computed $\mathbf{A}^\top \mathbf{A}$ which would mean that any further leverage scores computed are *global* leverage scores. Therefore, we have gradually pruned out all small leverage score rows earlier on and are only left with those that exceeded the threshold throughout the stream. Notice that this is entirely a consequence of the observation that ℓ_2 leverage scores computed on local bases (or partial sums of outer products $\sum_{i=1}^n \mathbf{A}_i^\top \mathbf{A}_i$) are upper bounds on their global leverage score. Thus, once a local leverage score drops below the threshold τ, that row's contribution will remain less than the τ threshold as subsequent rows are added.

Of course, having argued that $\mathbf{A}^\top \mathbf{A}$ is an exact subspace embedding in ℓ_2, one perhaps would not want to perform the above algorithm, but it indeed provides insight into the approach we would like to take. Unfortunately, in ℓ_p the same niceties of ℓ_2 are not present; specifically, we show that local ℓ_p leverage scores *need not be upper bounds on global* ℓ_p leverage scores. This is because leverage scores are calculated from a well-conditioned basis for a

matrix which need not be a well-conditioned basis for a block. Thus, the above line of argument will not work exactly, but it will serve as our starting point.

Returning to ℓ_p

Our key theorem, Theorem 4.4.1, formalises exactly this behaviour; that there is an approach to finding rows whose leverage exceeds a threshold *globally* by repeatedly finding rows of high local leverage. Let $\mathbf{B} = \mathbf{PA} \in \mathbb{R}^{n \times d}$ be a subset of the rows of $\mathbf{A}$, clearly $\|\mathbf{B}\|_p^p \leq \|\mathbf{A}\|_p^p$ so

$$\frac{\left\|\mathbf{e}_i^\top \mathbf{A}\right\|_p^p}{\|\mathbf{A}\|_p^p} \leq \frac{\left\|\mathbf{e}_i^\top \mathbf{A}\right\|_p^p}{\|\mathbf{B}\|_p^p}. \tag{4.10}$$

That is, relative ℓ_p row norms of a submatrix are at least as large as the full relative ℓ_p norms. However, it is not guaranteed that this ordering holds for leverage scores, i.e., when $\mathbf{A}$ and $\mathbf{B}$ are replaced by a well-conditioned basis $\mathbf{U_A}$ and $\mathbf{U_B}$ for each. In spite of this, we are able to show that local ℓ_p leverage scores restricted to a coordinate subspace of a matrix basis do not decrease too much when compared to leverage scores in the original space. Consider row i of $\mathbf{A}$ with local leverage score $\hat{w}_i$ and global leverage score w_i. Then we show that $\hat{w}_i \geq w_i / \operatorname{poly}(d)$ (which contrasts the behaviour were we in ℓ_2 that would state $\hat{w}_i \geq w_i$). The proof relies heavily on properties of the well-conditioned basis and is presented in the following lemma 4.4.2.

Lemma 4.4.2. *Let $w_i = w_i(\mathbf{A})$ denote the i^{th}* global *leverage score of $\mathbf{A} \in \mathbb{R}^{n \times d}$. Let $\mathbf{PA}$ denote an arbitrary (strict) subset of the rows from $\mathbf{A}$ which contains row i and denote the local leverage scores of $\mathbf{A}$ with respect to $\mathbf{PA}$ by $\hat{w}_i = w_i(\mathbf{PA})$. Then $w_i / \operatorname{poly}(d) \leq \hat{w}_i$ and, in particular, $w_i/(d\alpha^p\beta) \leq \hat{w}_i$.*

Proof. Let $\mathbf{U} = \mathbf{AR}$ be a well-conditioned basis for $\operatorname{col}(\mathbf{A})$. Recall that $w_i = \left\|\mathbf{e}_i^\top \mathbf{AR}\right\|_p^p$. Then for some coordinate j we must have $|\mathbf{e}_i^\top \mathbf{ARe}_j|^p \geq w_i/d$. Taking $\mathbf{x} = \mathbf{e}_j$ we see that:

$$|(\mathbf{ARx})_i|^p \geq \frac{w_i}{d}. \tag{4.11}$$

However, Lemma 4.4.1 implies:

$$\|\mathbf{ARx}\|_p^p \leq \|\mathbf{AR}\|_p^p \leq \alpha^p \leq \operatorname{poly}(d). \tag{4.12}$$

Hence, we have shown there exists $\mathbf{y} = \mathbf{ARx} \in \operatorname{col}(\mathbf{A})$ such that $|\mathbf{y}_i|^p \geq w_i/d$ from Equation (4.11). Additionally, Equation (4.12) shows that $\|\mathbf{y}\|_p^p \leq \alpha^p$,

thus:

$$\begin{aligned}\frac{|\mathbf{y}_i|^p}{\|\mathbf{y}\|_p^p} &\geq \frac{|\mathbf{y}_i|^p}{\alpha^p} \\ &\geq \frac{w_i}{d\alpha^p} \\ &\geq \frac{w_i}{\text{poly}\,(d)}.\end{aligned}$$

Crucially,

$$w_i \leq \frac{|\mathbf{y}_i|^p d\alpha^p}{\|\mathbf{y}\|_p^p}. \tag{4.13}$$

Next we focus on the case when a strict subset of rows from $\mathbf{A}$ have been stored in some index set $J \subset [n]$ which is represented by a row-selector matrix $\mathbf{P}_{ij} = 1$ if $j = i$ *and* $j \in J$ but otherwise $\mathbf{P}_{ij} = 0$. Then $\mathbf{B} = \mathbf{PA} \in \mathbb{R}^{n \times d}$ whose rows are either exactly those from $\mathbf{A}$ or the all zeros row. Define $\hat{\mathbf{y}} = \mathbf{BRx}$ so that $\hat{\mathbf{y}}_{j'} = \mathbf{y}_{j'}$ if $j' \in J$, otherwise $\hat{\mathbf{y}}_{j'} = 0$. Thus, $\|\hat{\mathbf{y}}\|_p^p \leq \|\mathbf{y}\|_p^p$ and:

$$w_i \leq \frac{|\mathbf{y}_i|^p d\alpha^p}{\|\hat{\mathbf{y}}\|_p^p}. \tag{4.14}$$

Now, for $i \in J$ (i.e. those stored in $\mathbf{B}$) $|\hat{\mathbf{y}}_i|^p = |\mathbf{y}_i|^p$. So, *for such indices*, using Equations (4.13) and (4.14)

$$w_i \leq \frac{|\hat{\mathbf{y}}_i|^p d\alpha^p}{\|\hat{\mathbf{y}}\|_p^p}. \tag{4.15}$$

Although $\hat{\mathbf{y}}$ is simply the restriction of $\mathbf{y}$ to the coordinates of $\mathbf{P}$, it is not guaranteed that the change of basis matrix $\mathbf{R}$ makes $\mathbf{PAR}$ well-conditioned, as it does for $\mathbf{U} = \mathbf{AR}$. We will thus use the change of basis $\hat{\mathbf{R}}$ so that $(\mathbf{PA})\hat{\mathbf{R}}$ is well-conditioned. Then the local leverage scores for $i \in J$ are (again using the shorthand $\mathbf{B} = \mathbf{PA}$): $\hat{w}_i = \left\|\mathbf{e}_i^\top \mathbf{B}\hat{\mathbf{R}}\right\|_p^p$. We now claim that $|\hat{\mathbf{y}}_i|^p / \|\hat{\mathbf{y}}\|_p^p \leq \text{poly}\,(d)\,\hat{w}_i$. Indeed,

$$\begin{aligned}|\hat{\mathbf{y}}_i|^p &= |(\mathbf{B}\hat{\mathbf{R}}\hat{\mathbf{x}})_i|^p \\ &\leq \left\|\mathbf{e}_i^\top \mathbf{B}\hat{\mathbf{R}}\right\|_p^p \|\hat{\mathbf{x}}\|_q^p \quad \text{by Hölder's inequality} \\ &\leq \hat{w}_i \beta \|\mathbf{B}\hat{\mathbf{R}}\hat{\mathbf{x}}\|_p^p \quad \text{by Definition 4.2.1, property 2 on } \mathbf{B}\hat{\mathbf{R}} \\ &\leq \beta \hat{w}_i \|\hat{\mathbf{y}}\|_p^p.\end{aligned}$$

Finally, recalling Equation (4.15) we achieve:

$$\frac{w_i}{d\alpha^p} \leq \frac{|\hat{\mathbf{y}}_i|^p}{\|\hat{\mathbf{y}}\|_p^p} \leq \beta \hat{w}_i.$$

The above relation proves the latter claim of the lemma statement whereby

the general poly (d) is immediate from Theorem 4.2.1 which states β is at most poly (d), thus proving the result. □

Although Lemma 4.4.2 shows that local leverage scores can potentially drop in arbitrary ℓ_p norm we will provide an algorithm that finds all rows exceeding a global threshold by altering the local threshold. That is, to find all $w_i > \tau$ globally we find all local leverage scores exceeding an adjusted threshold $\hat{w}_i > \tau/\operatorname{poly}(d)$ to obtain a superset of all rows which exceed the global threshold. The price to pay for this is a poly (d) increase in space cost which, importantly, remains *sublinear in* n. Hence, we can gradually prune out rows of small leverage and keep only the most important rows of a matrix. This is formalised in the following lemma:

Lemma 4.4.3. *All global leverage scores exceeding threshold τ can be found by computing local leverage scores and increasing the space by a* poly (d) *factor.*

Proof. Lemma 4.4.1 shows that the space necessary to find all leverage scores exceeding τ in index set I is $|I| \leq \operatorname{poly}(d)/\tau$. We focus next on finding a superset of I by considering only local leverage scores which is made possible by Lemma 4.4.2. For the stored rows J used to construct $\mathbf{P}$ in Lemma 4.4.2 we have $w_i/d\alpha^p\beta \leq \hat{w}_i$. Hence, any $w_i > \tau$ results in $\hat{w}_i > \tau/d\alpha^p\beta$ for the local thresholding. To keep all such $w_i > \tau$, we must store all $\hat{w}_i > \tau/d\alpha^p\beta = \hat{\tau}$. Arguing similarly as in Lemma 4.4.1 it can be shown that for $J = \{k : \hat{w}_k > \hat{\tau}\}$ we have $\alpha^p \geq \hat{\tau}|J|$ so that $|J| \leq d\alpha^{2p}\beta/\tau$. Equivalently, $|J| \leq d\beta\alpha^p \cdot |I|$ which proves the claim as Theorem 4.2.1 states that all of the parameters are poly (d). □

By combining Lemmas 4.4.2 and 4.4.3 we can now present the main result for this section, Theorem 4.4.1. This theorem is proved by arguing the correctness of Algorithm 6 which reads $\mathbf{A}$ once only, row by row, and so operates in the row-arrival streaming model of computation as follows. Let $\mathbf{A}'$ be the submatrix of $\mathbf{A}$ induced by the b block of poly $(d)/\tau$ rows. Upon storing $\mathbf{A}'$, we compute $\mathbf{U}$, a local well-conditioned basis for $\mathbf{A}'$ and the local leverage scores with respect to $\mathbf{U}$, $\hat{w}_i(\mathbf{U})$ are calculated. Now, the local and global leverage scores can be related by Lemma 4.4.2 as $w_i/\operatorname{poly}(d) \leq \hat{w}_i$ so we can decide which rows to keep using an adjusted threshold. Any i for which the local leverage exceeds the adjusted threshold is kept in the sample and all other rows are deleted. The sample cannot be too large by properties of the well-conditioned basis and leverage scores so these kept rows can be appended to the next block which is read in before computing another well-conditioned basis and repeating in the same fashion. Our implementation is given in Algorithm 6.

Algorithm 6: Deterministically finding rows of high leverage on a stream

Input: $\mathbf{A} \in \mathbb{R}^{n \times d}, \tau \in (0,1)$
Output: Rows of high leverage

```
Function HighLeverageRows(X, τ)
    b ← poly(d)/τ
    A' ← first b rows of A
    U ← wcb(A')
    B ← LeverageScoreCheck(U, A', τ/poly(d))
    while rows of A unseen do
        A' ← next b rows of A
        U ← wcb([A'; B])
        B ← LeverageScoreCheck(U, [A'; B], τ/poly(d))
    return B
Function LeverageScoreCheck(X, W, τ)
    nb. X = wcb(W) and τ ∈ (0,1) is a threshold
    N ← Number of rows in X
    Y ← 0
    for i = 1 : N do
        if w_i(X) > τ then
            Y_i ← W_i
    return Nonzero rows of Y
```

Theorem 4.4.1. *Let $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $\tau > 0$ be a fixed constant. There is a single-pass streaming algorithm which returns a summary $\mathbf{B} \in \mathbb{R}^{b \times d}$ with $b \leq \text{poly}(d)/\tau$. Moreover, the summary $\mathbf{B}$ can be updated in time $O(bd^2 + bd^5 \log b)$.*

Proof. The algorithm initially reads in $b = \text{poly}(d)/\tau$ rows of $\mathbf{A}$ and inserts these to matrix $\mathbf{A}'$. A well-conditioned basis $\mathbf{U}$ for $\mathbf{A}'$ is then computed using Theorem 4.2.1 which incurs the associated $O(bd^2 + bd^5 \log b)$ time cost. The matrix $\mathbf{U}$ and $\mathbf{A}'$ are passed to subroutine `LeverageScoreCheck` along with the adjusted local threshold $\tau' = \tau/\text{poly}(d)$ whereby if a row i in $\mathbf{U}$ has local leverage exceeding τ' then row i of $\mathbf{A}'$ is kept. There are at most $\text{poly}(d)/\tau$ of these rows by Lemma 4.4.3. So on the first call to `LeverageScoreCheck` a matrix $\mathbf{B}$ is returned with rows whose ℓ_p local leverage satisfies $\hat{w}_i \geq w_i/\text{poly}(d)$ (where w_i is the global leverage score and $\hat{w}_i$ is the associated local leverage score) and only those exceeding $\tau/\text{poly}(d)$ are kept.

The algorithm proceeds by repeating this process on $\mathbf{A}'$ (the next set of b rows from $\mathbf{A}$) appended to the summary $\mathbf{B}$ containing high leverage rows from $\mathbf{A}$ already found from the previous block. Proceeding inductively, we see that when `LeverageScoreCheck` is called with matrix $[\mathbf{A}'; \mathbf{B}]$ then a well-conditioned basis $\mathbf{U}$ is computed. Again $[\mathbf{A}'; \mathbf{B}]_i$ is kept if and only if the local leverage score from $\mathbf{U}$, $\hat{w}_i(\mathbf{U}) > \tau$. This results in the improved summary $\mathbf{B}$

which contains high leverage rows from both the prior summary $\mathbf{B}$ and the newly appended rows $\mathbf{A}'$. Repeating over all blocks $\mathbf{B}$ in $\mathbf{A}$, only the rows of high leverage are kept. Any row of leverage smaller than $\tau/\text{poly}(d)$ is ignored so this is the additive error incurred. At any given time, the space usage is bounded above by $2\,\text{poly}\,(d)\,/\tau$ needed to store the new rows $\mathbf{A}'$ and the prior summary $\mathbf{B}$. The update time is that required to obtain a well-conditioned basis on a matrix of size at most $\square$

4.5 Application: ℓ_∞-Regression

Here we present a method for solving ℓ_∞-regression in a streaming fashion. Given input $\mathbf{A}$ and a target vector $\mathbf{b}$, it is possible to achieve additive approximation error of the form $\varepsilon\|\mathbf{b}\|_p$ for arbitrarily large p. This contrasts with both Theorems 4.3.1 and 4.3.2 which achieve a relative error $\text{poly}\,(d)$ approximation. Both of these theorems require that p is constant and not equal to the ∞-norm. This restriction is due to a lower bound for ℓ_∞- regression showing that it cannot be approximated with relative error in sublinear space. The key to proving Theorem 4.5.1 below is using Theorem 4.4.1 to find high leverage rows and arguing that these are sufficient to give the claimed error guarantee.

The ℓ_∞-regression problem, sometimes known as the *Chebyshev Approximation Problem* [BBV04], has been previously studied in the overdetermined case and can naturally be applied to curve-fitting under this norm. Solving ℓ_∞-regression requires solving a large linear program [Spo76, BBV04]. If the errors are known to be distributed uniformly across an interval then ℓ_∞-regression estimator is the maximum-likelihood parameter choice [Han78]. The same work argues that such uniform distributions on the errors often arise as round-off errors in industrial applications whereby the error is controlled or is small relative to the signal. There are further applications such as using ℓ_∞-regression to remove outliers prior to ℓ_2 regression in order to make the problem more robust [SSH$^+$14]. By applying ℓ_∞ regression on subsets of the data an approximation to the Least Median of Squares (another robust form of regression) can be found. We now define the problem and proceed to show that it is possible to compute an approximate solution with additive error in ℓ_p-norm for arbitrarily large p. The implementation is given in Algorithm 7 with correctness being established in Theorem 4.5.1.

Approximate ℓ_∞-Regression problem: Given data $\mathbf{A} \in \mathbb{R}^{n\times d}$, target vector $\mathbf{b} \in \mathbb{R}^n$, and error parameter $\varepsilon > 0$, compute an additive $\varepsilon\|\mathbf{b}\|_p$ error solution to:

$$\min_{\mathbf{x}\in\mathbb{R}^d} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_\infty = \min_{\mathbf{x}\in\mathbb{R}^d} \left[\max_i |(\mathbf{A}\mathbf{x})_i - \mathbf{b}_i|\right].$$

An ε *additive error solution* means that we find an estimate $\hat{\mathbf{x}}$ which

Algorithm 7: Deterministic Approximate ℓ_∞ Regression

Input: $\mathbf{A} \in \mathbb{R}^{n \times d}, \mathbf{b} \in \mathbb{R}^n, p \geq 1, \tau \in (0,1)$
Output: Smaller instance for the regression problem

1 **Function** `Approx`ℓ_∞`Regression`$(\mathbf{A}, \mathbf{b}, p, \tau)$
2 Use Algorithm 6 to obtain all rows $\mathbf{A}_i$ of leverage exceeding $\tau / \operatorname{poly}(d)$
3 **if** $\mathbf{A}_i$ *has large leverage score* **then**
4 store the pair $(\mathbf{A}_i, \mathbf{b}_i)$
5 **else if** $\mathbf{A}_i$ *has leverage less than* $\tau / \operatorname{poly}(d)$ and $|\mathbf{b}_i| \geq \tau \|\mathbf{b}\|_p$ **then**
6 store the pair $(\mathbf{0}_{1 \times d}, \mathbf{b}_i)$
7 Concatenate all stored $\mathbf{A}_i$ and $\mathbf{0}_{1 \times d}$ into summary $\mathbf{A}'$
8 Concatenate all stored $\mathbf{b}_i$ into new target vector $\mathbf{b}'$
9 Solve $\hat{f} = \min_{\mathbf{x}' \in \mathbb{R}^d} \|\mathbf{A}'\mathbf{x}' - \mathbf{b}'\|_\infty$
10 **return** $\hat{f}$

satisfies, compared to the optimal solution $\mathbf{x}^\star$:

$$\|\mathbf{A}\mathbf{x}^\star - \mathbf{b}\|_\infty \leq \|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_\infty \leq \|\mathbf{A}\mathbf{x}^\star - \mathbf{b}\|_\infty + \varepsilon \|\mathbf{b}\|_p.$$

Theorem 4.5.1. *Let $\mathbf{A} \in \mathbb{R}^{n \times d}, \mathbf{b} \in \mathbb{R}^n$ and fix constants $p \geq 1, \varepsilon > 0$ with $p \neq \infty$. There exists a one-pass deterministic streaming algorithm which solves the ℓ_∞-*regression problem *up to an additive $\varepsilon \|\mathbf{b}\|_p$ error in $d^{O(p)}/\varepsilon^{O(1)}$ space, $O(md^5 + md^2 \log m)$ update time and $T_{\mathsf{solve}}(m, d)$ query time to solve a linear program on an $m \times d$ input matrix.*

Proof. Algorithm 7 proceeds by finding all rows whose global leverage score exceeds a τ threshold. This is executed on the stream by Algorithm 6 which, by Lemma 4.4.3 incurs the associated $\operatorname{poly}(d)/\tau$ space cost. On these rows of high leverage, the pair $(\mathbf{A}_i, \mathbf{b}_i)$ is maintained exactly. Additionally, Algorithm 7 retains the pair $(\mathbf{0}_{1 \times d}, \mathbf{b}_i)$ if the leverage score is below the threshold but the target value is large. Since there can be at most $1/\tau$ such $\mathbf{b}_i$, this extra space is subsumed by the cost to obtain the summary of high leverage rows. Let $\mathbf{A}'$ denote the concatenated rows of $\mathbf{A}$ and copies of $\mathbf{0}_{1 \times d}$ if necessary, and similarly for $\mathbf{b}'$ as in lines 7 and 8 of Algorithm 7. The pair $(\mathbf{A}', \mathbf{b}')$ is then our reduced instance to the ℓ_∞ regression problem of $\operatorname{poly}(d)/\tau$ rows.

Let $\mathbf{R} \in \mathbb{R}^{d \times d}$ be a change of basis matrix so that $\mathbf{AR}$ is an ℓ_p well-conditioned basis. Note that $\mathbf{R}$ is not computed in Algorithm 7 but is used for convenience in our proof. Secondly, note that $\mathbf{R}$ makes $\mathbf{A}$ well-conditioned, not necessarily the summary matrix $\mathbf{A}'$. We will now focus on the task of solving $\min_{\mathbf{z}} \|\mathbf{A}'\mathbf{R}\mathbf{z} - \mathbf{b}'\|_\infty$.

Observe that any solution $\mathbf{z}$ must have $\|\mathbf{z}\|_p \leq \operatorname{poly}(d) \|\mathbf{b}\|_p$ as otherwise

$\mathbf{z} = \mathbf{0}_{d\times 1}$ is a better solution. This can be seen through:

$$\begin{aligned}\left\|\mathbf{A}'\mathbf{R}\mathbf{0}_{d\times 1} - \mathbf{b}'\right\|_\infty &= \left\|\mathbf{b}'\right\|_\infty \\ &= \|\mathbf{b}\|_\infty \\ &\geq \tau \|\mathbf{b}\|_p\end{aligned}$$

Now we evaluate how the summary contributes to the ℓ_∞ objective value. First focus on the rows with leverage score exceeding the $\tau/\operatorname{poly}(d)$ threshold. Such rows are stored exactly so we clearly have:

$$|(\mathbf{A}'\mathbf{R}\mathbf{z})_i - \mathbf{b}_i| = |(\mathbf{A}\mathbf{R}\mathbf{z})_i - \mathbf{b}_i|$$

and there is no change in the cost on this row between the original and small-instance regression problems.

On the other hand, suppose that $\mathbf{A}_i$ has leverage less than the $\tau/\operatorname{poly}(d)$ threshold. Then:

$$\begin{aligned}|\langle(\mathbf{A}\mathbf{R})_i, \mathbf{z}\rangle| &\leq \|(\mathbf{A}\mathbf{R})_i\|_\infty \|\mathbf{z}\|_1 \\ &\leq \|(\mathbf{A}\mathbf{R})_i\|_p \|\mathbf{z}\|_1 \\ &\leq \|(\mathbf{A}\mathbf{R})_i\|_p \cdot d\|\mathbf{z}\|_p \\ &\leq \frac{\tau}{\operatorname{poly}(d)} \cdot d \cdot \operatorname{poly}(d) \|\mathbf{b}\|_p \\ &\leq \varepsilon \|\mathbf{b}\|_p.\end{aligned}$$

The first and third inequalities are due to Hölder's inequality, while the final one follows by the assumption on $\|\mathbf{z}\|_p$. By an appropriate choice of the $\operatorname{poly}(d)$ factors scaling τ and choosing ε accordingly we see that $|\langle(\mathbf{A}\mathbf{R})_i, \mathbf{z}\rangle| \leq \varepsilon \|\mathbf{b}\|_p$ which obtains the ε guarantee. On such coordinates the ℓ_∞ cost is $|\mathbf{b}_i| \pm \varepsilon \|\mathbf{b}\|_p$ so by replacing the row with one which is all zero we still pay $|\mathbf{b}_i|$ which is within the $\tau \|\mathbf{b}\|_p$ had we included the row.

Let I denote the set of indices from $(\mathbf{A}, \mathbf{b})$ which define the reduced instance $(\mathbf{A}', \mathbf{b}')$. What we have shown is that if $i \in I$ and $\mathbf{A}_i$ has high leverage, then the contribution to the ℓ_∞ cost is no different in the reduced problem compared to the full problem. On the other hand, if $i \in I$ but $\mathbf{A}_i$ had small leverage, then the contribution to the ℓ_∞ cost is $|\mathbf{b}_i|$. However, on such rows, the optimal cost is at most $|\mathbf{b}_i| + \varepsilon \|\mathbf{b}\|_p$ so we remain within the claimed additve error on this row. Thus, taking the maximum over all $i \in I$, we either recover the cost exactly, or have an upper bound of the optimal cost plus $\varepsilon \|\mathbf{b}\|_p$, as claimed. The desired space bound is achieved through the relation between threshold τ, the $\operatorname{poly}(d)$ factors, and ε. □

Comments on Theorem 4.5.1: Linear Programming Time Cost

Running time analyses from the linear programming community tend to focus on the *iteration count* of a solver which may differ from worst-case run time analysis. Secondly, there are a variety of algorithms for linear programming that exploit structure in different ways to provide the most efficient solutions. Nevertheless, there are some observations we can make that highlight how reducing the linear program for ℓ_∞ regression from n to m constraints to yield efficiency gains. Standard interior point methods on an instance $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $\mathbf{b} \in \mathbb{R}^n$ require T iterations for an overall complexity of $O(\sqrt{d}T)$ [BBV04, LS14]. An extensive discussion comparing various methods for solving linear programs is given in [LS14] but we focus on the general picture in which the authors establish the theoretical best running time to $\tilde{O}(\sqrt{\text{rank}\,(\mathbf{A})}T)$, where the $\tilde{O}$ notation hides lower order $\text{poly}\,(\log(\cdot))$ terms. They argue that for both their method and the interior point methods, the cost of an iteration is at least $O(nd)$ so really this time complexity is $O(nd^{1.5}T)$ by recalling that $\mathsf{nnz}\,(\mathbf{A}) \geq n$ and $\text{rank}\,(\mathbf{A}) = d$. It is additionally argued in [LS14] that prior to their contribution, when $n = \tilde{\Omega}(d)$ the previous best running time was $O(n^{1.5}dT)$. Despite this relatively recent breakthrough, current practical implementations are typically based upon Karmarkar's interior barrier point method (requiring $O(dT)$ iterations) [Gurobi] or a variant with lower iteration count of $O(\sqrt{d}T)$ [Ren88]. For our approach, we assume access to a black box solver that with *query time* $T_{\text{solve}}(m, d)$ being the time taken to solve the linear program on a reduced instance size of $m \times d$. When n is large and many iterations are needed, the reduction to only m constraints can have a substantial benefit on the running time for the solve step. An example of this can be seen in Figure 4.4: the salient point being that a method keeping $m' > m$ constraints has higher time cost to solve the linear program.

Relative Error Lower Bound

Also, observe that Theorem 4.5.1 requires $p < \infty$. This restriction is necessary to forbid relative error with respect to the infinity norm. Indeed, p can be an arbitrarily large constant, but for $p = \infty$ we can look for rows above an $\varepsilon/\,\text{poly}\,(d)$ threshold in the case when $\mathbf{A}$ is an all-ones column n-vector (so an $n \times 1$ matrix). Then $\|\mathbf{Ax}\|_\infty = \|\mathbf{x}\|_\infty$ since $\mathbf{x}$ is a scalar. Also, $\mathbf{A}$ is a well-conditioned basis for its own column span but the number of rows of leverage exceeding $\varepsilon/\,\text{poly}\,(d) = \varepsilon$ is n for a small constant ε. This intuition allows us to prove the following theorem.

Theorem 4.5.2. *Any algorithm which outputs an $\varepsilon \|\mathbf{b}\|_\infty$ relative error solution to the ℓ_∞-regression problem requires $\min\left\{n, 2^{\Omega(d)}\right\}$ space.*

Proof. Let $\mathcal{C} \subset \{0,1\}^d$ be a set of $2^{\Omega(d)}$ strings in $\{0,1\}^d$ with each coordinate in a string uniformly sampled randomly from $\{0,1\}$. Let $\mathbf{z}, \mathbf{z}' \in \mathcal{C}$ and fix a constant $0 < c < 1$. By a Chernoff bound (see e.g. Lemma 3.3.1) it follows that there are at least cd coordinates in $[d]$ for which $\mathbf{z}_i = 0$ and $\mathbf{z}'_i = 1$ with probability $1 - 2^{-\Omega(d)}$. This implies for appropriate constants in the $\Omega(\cdot)$, by a union bound, all pairs of strings $\mathbf{z}, \mathbf{z}' \in \mathcal{C}$ have this property. Hence, such a $\mathcal{C}$ exists and we will fix this for the proof.

We will reduce the ℓ_∞ regression problem to that of Index (see Definition 2.4.1). Alice holds a subset $T \subset \mathcal{C}$ and denote $|T| = n - 1$ so that $T = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{n-1}\}$. Alice's bit string is a vector $\mathbf{a} \in \{0,1\}^{|C|}$ which has $\mathbf{a}_i = 1$ if and only if the string $i \in \mathcal{C}$ is in her held set T. Of course, this requires an enumeration or index scheme between the set of indices and strings, but since the canonical mapping will suffice, this is not problematic. Bob holds a test vector $\mathbf{y} \in \mathcal{C}$ and is tasked with determining whether $\mathbf{y} \in T$ or $\mathbf{y} \in \mathcal{C} \setminus T$. The vector that Bob will actually use to make a query will be $\bar{\mathbf{y}} = -\mathbf{y}$ and this is appended to the input as shown below. Let $J = \mathrm{supp}(\mathbf{y})$ and $J^c = [d] \setminus \mathrm{supp}(\mathbf{y})$.

The test instance is the following $(\mathbf{A}, \mathbf{b})$ pair:

$$\mathbf{A} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_{n-1} \\ \hline \bar{\mathbf{y}} \end{bmatrix} \in \mathbb{R}^{n \times d} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ \hline 1 \end{bmatrix} \in \mathbb{R}^n.$$

Note that the input matrix $\mathbf{A}$ is hidden from Bob as otherwise he could easily scan through the matrix and check if any of the rows are equal to the vector he holds, thus determining whether Alice holds $\mathbf{y}$.

We will use the function

$$g(\mathbf{u}) = \left| \sum_{j=1}^{d} \mathbf{u}_j \mathbf{x}_j - 1 \right|$$

which evaluates the ℓ_∞ cost of a vector (string) $\mathbf{u}$ when $\mathbf{x}$ is fixed and $\mathbf{b} = \mathbf{1}_{n \times 1}$ is given. There are two cases to consider (i) Bob's test vector $\mathbf{y} \in T$ and (ii) $\mathbf{y} \in \mathcal{C} \setminus T$.

Case (i). Suppose first that $\mathbf{y} \in T$ so that *both* $\mathbf{y}$ and $\bar{\mathbf{y}}$ are present in $\mathbf{A}$. This is because Alice will have inserted $\mathbf{y}$ into the table as she holds $\mathbf{y} \in T$,

and Bob inserted $\bar{\mathbf{y}}$ by our construction. Then:

$$g(\mathbf{y}) = \left|\sum_{j\in J} \mathbf{x}_j - 1\right| \quad \text{and} \quad g(\bar{\mathbf{y}}) = \left|1 + \sum_{j\in J} \mathbf{x}_j\right|.$$

Note that this implies the optimal cost for ℓ_∞ regression is at least 1 because $\sum_{j\in J} \mathbf{x}_j$ is either at least 0, in which case $g(\bar{\mathbf{y}}) \geq 1$ or $\sum_{j\in J} \mathbf{x}_j < 0$ resulting in $g(\mathbf{y}) \geq 1$. Any remaining rows $\mathbf{w}_i$ of $\mathbf{A}$ do not matter because either $g(\mathbf{w}_i) < 1$, in which case they are overruled by the max operation for the ℓ_∞ norm, or $g(\mathbf{w}_i) > 1$ and we will only need $g(\mathbf{w}_i) \geq 1$ anyway, which is satisfied by $g(\mathbf{y})$ or $g(\bar{\mathbf{y}})$.

Case (ii). Now suppose that $\mathbf{y} \in \mathcal{C} \setminus T$ so that Alice does not hold $\mathbf{y}$ and thus it is not present on the input matrix $\mathbf{A}$. In this case, we can lower bound the ℓ_∞ cost over $\mathbf{A}$. Recall that J^c is the set of indices where Bob's vector $\mathbf{y} = 0$. For $i \in J^c$ set $\mathbf{x}_i = 1/d$ and otherwise set $\mathbf{x}_i = -c/2d$. We will evaluate the cost of fixing such an $\mathbf{x}$ for the function g over all input rows of $\mathbf{A}$.

First consider

$$\begin{aligned} g(\bar{\mathbf{y}}) &= \left|\sum_{j\in J} \bar{\mathbf{y}}_j \mathbf{x}_j - 1\right| \\ &\leq \left|(-1)\left(\frac{-c}{2d}\right) cd - 1\right| \\ &\leq \left|1 - c^2/2\right|. \end{aligned}$$

That is, the ℓ_∞ cost of Bob's input $\bar{\mathbf{y}}$ is at most $\left|1 - c^2/2\right|$. On the other hand, for the remaining rows of $\mathbf{A}$ we have the vectors $\mathbf{w}_i$ which, by the construction of $\mathcal{C}$ have *at least* cd indices with $\mathbf{w}_{ij} = 0$ and $\mathbf{y}_j = 1$. Hence, the ℓ_∞ cost on such rows is:

$$\begin{aligned} g(\mathbf{w}_i) &= \left|\sum_{j\in[d]} \mathbf{w}_{ij}\mathbf{x}_j - 1\right| \\ &\leq |1 - c/2|. \end{aligned}$$

This follows since $|J^c| \geq cd$ so

$$\begin{aligned} \sum_{j\in[d]} \mathbf{w}_{ij}\mathbf{x}_j &\geq cd(1/d) - (d - cd)(c/2d) \\ &\geq c - c/2 \\ &\geq c/2. \end{aligned}$$

Finally, since $c < 1$ we have $|1 - c/2| \leq \left|1 - c^2/2\right|$ so the upper bound on

the ℓ_∞ cost is given by $g(\bar{\mathbf{y}})$. As the ℓ_∞ cost in Case (ii) is a constant factor less than the ℓ_∞ cost from Case (i), Bob can query a constant factor relative error approximation algorithm for the ℓ_∞ regression problem and determine whether his test vector $\mathbf{y} \in T$ or not. Consequently, Bob can solve Index: from the enumeration he knows the index of $\mathbf{y} \in \mathcal{C}$ and thus can return $\mathbf{a}_\mathbf{y} = 1$ if Alice holds $\mathbf{y} \in T$ or return 0 otherwise. Therefore, Bob must incur $\Omega(|\mathcal{C}|)$ communication, which by [KNR99] results in $\Omega(|\mathcal{C}|)$ space for a streaming algorithm. Since $|\mathcal{C}| = \min(n, 2^{\Omega(d)})$, this is the space bound we claimed.

□

4.6 Experiments

To validate our approach, we evaluate the use of high ℓ_p-leverage rows in order to approximate ℓ_∞-regression[2], focusing particularly on the cases using ℓ_1 and ℓ_2 well-conditioned bases. It is straightforward to model ℓ_∞-regression as a linear program in the offline setting. We use this to measure the accuracy of our algorithm. The implementation is carried out in the single pass streaming model with a fixed space constraint, m, and threshold, α^p/m for both conditioning methods to ensure the number of rows kept in the summary did not exceed m. Recall from Remark 2.1.2 that the single-pass row-arrival streaming implementation is equivalent to the distributed summary model with only one participant applying merge-and-reduce, so this experiment can also be seen as a distributed computation with the merge step being the appending of new rows and the reduce step being the thresholding in the new well-conditioned basis.

Methods. We analyse two instantiations of Algorithm 7 based on how we find a well-conditioned basis and repeat over 5 independent trials with random permutations of the data. Recall from Definition 4.2.1 that an (α, β, p) well-conditioned basis satisfies $\|\mathbf{U}\|_p \leq \alpha$ and for all $\mathbf{z} \in \mathbb{R}^d$, $\|\mathbf{z}\|_q \leq \beta \|\mathbf{U}\mathbf{z}\|_p$ with α and β small polynomials in d. The methods are as follows:

`SPC3`: We use an algorithm of [YMM13] to compute an ℓ_1-wcb. This method is randomised as it employs the Sparse Cauchy Transform[3] and is only an ℓ_1-well-conditioned basis with constant probability We also implemented a check condition which showed that almost always, roughly 99% of the time, the randomised construction `SPC3` would return a $(d^{2.5}, 1, 1)$-well-conditioned

[2]Code available at `https://github.com/c-dickens/stream-summaries-high-lev-rows`

[3]The Sparse Cauchy Transform is a sparsified version of the Cauchy Transform which is a matrix of rescaled Cauchy random variables. The dense version was first shown to provide an ℓ_1 subspace embedding in [SW11] with $O(d \log d)$ rows and distortion $O(d \log d)$. This was shown to have the optimal embedding dimension in [WW19]. To obtain the sparse version of [YMM13], we take a rescaled diagonal random Cauchy matrix and premultiply by a CountSketch.

basis. Thus, we bypassed this check in our experiment to ensure quick update times.

`Orth`: In addition, we also used an orthonormal basis using the QR decomposition which is an ℓ_2-wcb. This method is fully deterministic and outputs a $(\sqrt{d}, 1, 2)$-well- conditioned basis.

`Sample`: A sample of the data is chosen uniformly at random and the retained summary has size exactly m.

`Identity`: No conditioning is performed. For a block $\mathbf{B}$ of the input, the *surrogate* scores $w_i(\mathbf{B}) = \|\mathbf{e}_i^\top \mathbf{B}\|_2^2 / \|\mathbf{B}\|_F^2$ are used to determine which rows to keep. As the sum of these $w_i(\mathbf{B})$ is 1, we keep all rows which have $w_i(\mathbf{B}) > 2/m$. Since no more than $m/2$ of the rows can satisfy $w_i(\mathbf{B}) > 2/m$, the size of the stored subset of rows can be controlled and cannot grow too large.

Remark 4.6.1. *The `Identity` method keeps only the rows with high norm which contrasts our conditioning approach: if most of the mass of the block is concentrated on a few rows then these will appear heavy locally despite the possibility that they may correspond to previously seen or unimportant directions. In particular, if these heavy rows significantly outweigh the weight of some sparse directions in the data it is likely that the sparse directions will not be found at all. For instance, consider data $\mathbf{X} \in \mathbb{R}^{n \times d}$ which is then augmented by appending the identity (and zeros) so that these are the only vectors in the new directions. That is, set*

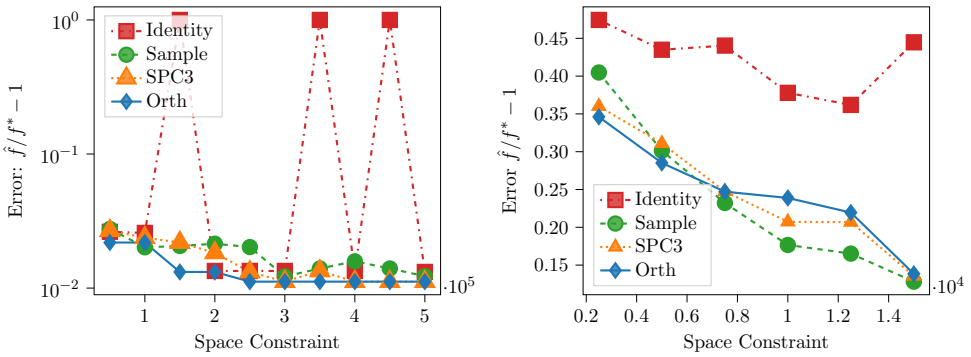
$$\mathbf{X}' = \left[\begin{array}{c|c} \mathbf{X} & \mathbf{0}_{n \times k} \\ \hline \mathbf{0}_{k \times d} & \mathbf{I}_{k \times k} \end{array}\right]$$

and then permute the rows of $\mathbf{X}'$. The appended sparse vectors from $[\mathbf{0}_{k \times d}, \mathbf{I}_{k \times k}]$ will have leverage of 1 as they are orthogonal to all other rows in $\mathbf{X}'$. Hence, they will be detected by the well-conditioned basis methods. However there is no guarantee that the `Identity` method will identify these directions if the entries in $\mathbf{X}$ significantly outweigh those in $\mathbf{I}_{k \times k}$. In addition, there is also no guarantee that using uniform sampling will identify these points, particularly when k is small compared to n and d. So while choosing to do no conditioning seems attractive, this example shows that doing so may not give any meaningful guarantees and hence we prefer the approach from Section 4.4. We compare only to these baselines as we are not aware of any other competing methods in the small memory regime for the ℓ_∞-regression problem.

Datasets. We tested the methods on a subset of the *US Census Data* containing 5 million rows and 11 columns[4] and *YearPredictionMSD*[5] which has roughly $500,000$ rows and 90 columns (although we downsample to a fixed set

[4] `http://www.census.gov/census2000/PUMS5.html`

[5] `https://archive.ics.uci.edu/ml/datasets/yearpredictionmsd`

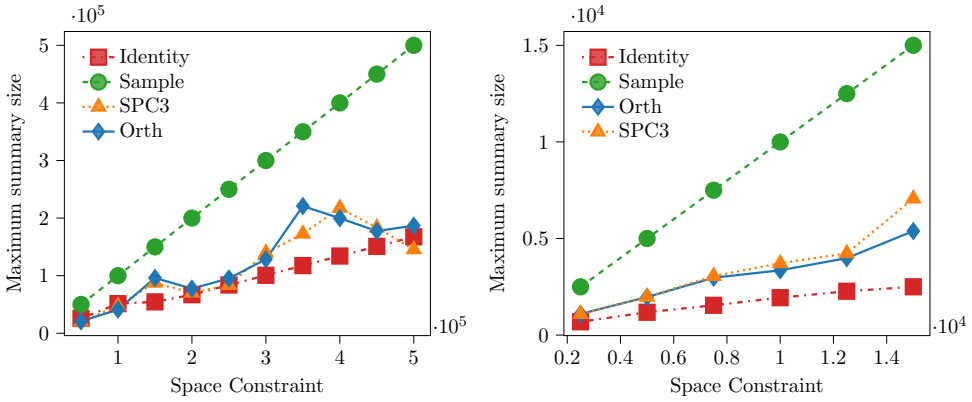


(a) U.S. Census (b) YearPredictionMSD

Figure 4.1: Error vs Space Constraint, which is the budget or largest number of rows that can be stored, not the summary size. Total input size is $5,000,000 \times 11$ for U.S. Census Data and approximately $50,000 \times 90$ for YearPredictionsMSD.

of $50,000$ observations so that the linear program for ℓ_∞ regression is tractable) For the census dataset, space constraints between 50,000 and 500,000 rows were tested which represents a range of $1-10\%$ of all available rows. For the YearPredictionsMSD data space budgets were tested between 2,500 and 15,000 representing $5-30\%$ of the rows due to downsampling. The general behaviour is roughly the same for both datasets. We vary the space constraint which is a budget on the total number of rows that can be stored, it is not necessarily the size of the summary, and is always less than the input size. The summary size is always upper bounded by the space constraint and varies depending on how the threshold is set.

Results on approximation error compared to storage. Let $f(\mathbf{x}) = \|\mathbf{A}\mathbf{x}-\mathbf{b}\|_\infty$ and denote the minimal value of the full regression obtained by the globally optimal solution $\mathbf{x}^\star$ as $f^* = f(\mathbf{x}^\star)$. Let the solution $\hat{\mathbf{x}}$ denote that found on the reduced instance after finding high leverage rows and write its associated objective value as $\hat{f} = f(\hat{\mathbf{x}})$. Hence, the approximation error is measured as $\hat{f}/f^* - 1$ (note that $\hat{f} \geq f^*$). An error closer to 0 demonstrates that $\hat{f}$ is roughly the same as f^* so the optimal value is well-approximated. Figures 4.1a and 4.1b show that on both datasets the `Identity` method consistently performs poorly while `Sample` achieves comparable accuracy to the conditioning methods. Despite the simplicity of uniform sampling to keep a summary, the succeeding sections discuss the increased time and space costs of using such a sample and show that doing so is not favourable. Thus, neither of the baseline methods output a summary which can be used to approximate the regression problem both *accurately* and *quickly*, hence justifying our use of leverage scores. Our conditioning methods perform particularly well in the *US Census Data* data (Figure 4.1a) with `Orth` appearing to give the most accurate summary and `SPC3` performing comparably well but with slightly more fluctuation: similar behaviour is observed in the *YearPredictionMSD* (Figure 4.1b) data too. The



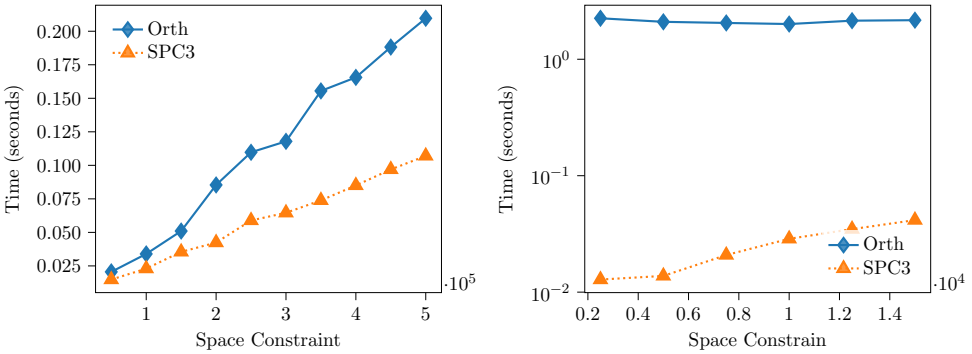
(a) U.S. Census (b) YearPredictionMSD

Figure 4.2: Maximum Summary Size vs Space Constraint

conditioning methods are also seen to be robust to the storage constraint, give accurate performance across both datasets using significantly less storage than sampling, and give a better estimate in general than doing no conditioning.

Results on Space Complexity. Recall that the space constraint is m rows and throughout the stream, after a local computation, the merge step concatenates more rows to the existing summary until the bound m is met, prior to computing the next reduction. During the initialization of the block $\mathbf{A}'$ by Algorithm 6, the number of stored rows is exactly m. However, we measure the maximum number of rows kept in a summary after every reduction step to understand how large the returned summary can grow. As seen in Figures 4.2a and 4.2b, `Identity` keeps the smallest summary but there is no reason to expect it has kept the most important rows. In contrast, if m is the bound on the summary size, then uniform sampling always returns a summary of size exactly m. However, we see that this is not optimal as both conditioning methods can return a set of rows which are pruned at every iteration to roughly half the size and contains only the most important rows in that block. Both conditioning methods exhibit similar behavior and are bounded between both `Sample` and `Identity` methods. Therefore, both of the conditioning methods respect the theoretical bound and, crucially, return a summary which is sublinear in the space constraint and hence a significantly smaller fraction of the input size.

Results on Time Complexity. There are three time costs measured which we will separate for ease of analysis. The first is the *update time* which measures how long it takes to find the local basis and prune out unimportant rows. Secondly, we measure the *query time* which is simply the time taken to call the black-box linear program solver on the reduced instance $(\mathbf{A}', \mathbf{b}')$. Finally, we evaluate the *total time* for computation which is a useful measure to understand how the (potentially significant) time cost for summarisation



(a) U.S. Census (b) YearPredictionMSD

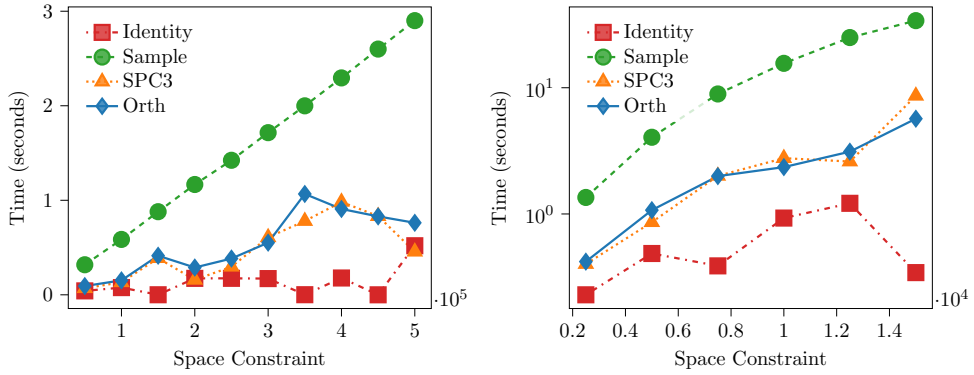
Figure 4.3: Update time for local basis

compares to solving the instance without summarisation. This should be roughly the sum of the query time and a multiple of update times depending on how many local summaries are computed.

1. Results on Time Complexity: Update Time The first is the update time taken to compute the local well-conditioned basis which is theoretically $O(md^2 + md^5 \log m)$ by Theorem 4.2.1. However, the two bases that we test are an orthonormal basis, computable in time $O(md^2)$ and the `SPC3` transform which takes time $O(\texttt{nnz}(\mathbf{B}) \log m)$ for a block $\mathbf{B}$ with m rows and $\texttt{nnz}(\mathbf{B})$ non-zero entries. Figure 4.3a demonstrates that `SPC3` is faster than `Orth` on this data in practice but this is a small absolute difference. On the other hand, Figure 4.3b shows a more dramatic separation between between the two methods which is likely due to the quadratic dependence on d for `Orth` compared to the linear dependence on d (through the $\texttt{nnz}(\mathbf{B})$ term) for `SPC3`.

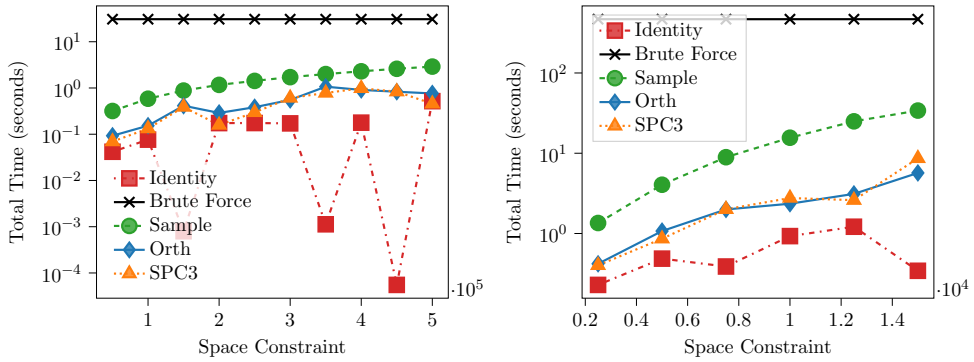
2. Results on Time Complexity: Query Time The query times for Census and Years data are presented in Figures 4.4a and 4.4b. We see that the time taken to solve the reduced instance is proportional to the summary size in all settings but the conditioning methods perform noticeably better due to the smaller summary size that is returned as discussed in the previous section. However, the disparity between `Sample` and the conditioning methods becomes significant as the space constraint grows. This is due to the increased size summary retained by sampling, further justifying our approach of pruning rows at every stage. While `Identity` appears to have fast query time, this is due to the summary being smaller (cf. Figures 4.2a and 4.2b). Unlike the update time, there is little difference between the behaviours exhibited on both datasets.

3. Results on Time Complexity: Total Time Both the update and query times have an impact on the total time. Although Figures 4.5a and 4.5b show noticeable differences in the time taken to obtain the basis, these time discrepancies becomes negligible over the entirety of the stream as seen in



(a) U.S. Census (b) YearPredictionMSD

Figure 4.4: Query time to solve optimisation



(a) U.S. Census (b) YearPredictionMSD

Figure 4.5: Total time cost

Figures 4.5a and 4.5b. Although it may seem that for smaller summaries more local bases need to be computed and this time could prohibitively increase over the stream, Figure 4.5a demonstrates that even using small blocks does not cause the *overall time* (to process the stream and produce an approximate query) to increase too much. Hence, an approximation can be obtained which is highly accurate, and in total time faster than the brute force solver.

Experimental Summary. While it might seem attractive not to perform any conditioning on the matrix and just pick heavy rows, our experiments show that this strategy is not effective in practice, and delivers poor accuracy. Although a simple sample of randomly chosen rows can be easily maintained, this appears less useful due to the increased time costs associated with larger summaries when conditioning methods output a similar estimate in less time over the entire stream. As the ℓ_∞-regression problems depend only on a few rows of the data there are cases when uniform sampling can perform well: if many of the critical rows look similar then there is a chance that uniform sampling will select some examples. In this case, the leverage of the important direction is divided across the repetitions, and so it is harder to ensure that desired direction is identified. Despite this potential drawback we have shown

that both `Orth` and `SPC3` can be used to find accurate summaries which perform robustly across each of the measures we have tested. It appears that `SPC3` performs comparably to `Orth`; both are relatively quick to compute and admit accurate summaries in similar space. In particular, both conditioning methods return summaries which are a fraction of the space budget and hence highly sublinear in the input size, which give accurate approximations and are robust to the concatenation of new rows. All of these factors make the conditioning method fast in practice to both find the important rows in the data and then compute the reduced regression problem with high accuracy.

Due to the problems in constructing summaries which can be used to solve regression quickly and accurately when using random sampling or no transformation, our methods are shown to be efficient and accurate alternatives. Our approach is vindicated both theoretically and practically: this is most clear in the U.S. Census dataset where small error can be achieved using a summary roughly 2% the size of the data. This also results in an overall speedup as solving the optimization on the reduced set is much faster than solving on the full problem. Such significant savings show that this general approach can be useful in large-scale applications.

4.7 Discussion

In this chapter we have presented the first algorithms for obtaining deterministic summaries in arbitrary ℓ_p for for subspace embedding, low rank approximation and finding high leverage rows. Additionally, we were able to give applications of the summaries and used them to approximate ℓ_p regression, including ℓ_∞ regression. Our results apply simultaneously in the streaming model and the distributed summary model so these algorithms are flexible enough for different modern computing environments.

Although these are the strengths of our methods, there are noticeable weaknesses. Unsurprisingly, the generality of our approach is both a blessing and a curse: we cover all p but may miss special structure that is more specific to certain p norms. One example where this has been seen is in [LWW20] in which the authors show dichotomies in the hardness of obtaining a (random) sketch for even $p \geq 2$ and odd $p \geq 1$. Another weakness of our results is that for ℓ_p leverage scores, we are not able to convert this into a statement about the *quality* of the summary in comparison to the input matrix $\mathbf{A}$. Part of the difficulty here is that we cannot exploit the SVD as has been done in [PKB14b, McC18] which is the typical approach for making quality-of-approximation arguments in ℓ_2.

Small-space summaries based on weighted samples of the input $\mathbf{A}$ have also been constructed using so-called Lewis weights which more coherently map

between ℓ_p and ℓ_2. It is possible that these could give better approximation guarantees than our results, yet the problem is that computing Lewis weights seems to require multiple accesses to the input matrix. Although there are fast algorithms for evaluating Lewis weights if all of the data is accessible, doing this in a memory-constrained or strict data availability model is not immediately clear and remains open for investigation.

Although we have applied Algorithm 6 to the ℓ_∞ regression problem, this is less widely-used compared to robust versions of regression such as ℓ_1 regression. Due to the similarities in our approaches, one might think that we can naturally use the same idea for ℓ_1 regression. However, the probem here seems to come from the crucial distinction between ℓ_1 and ℓ_∞: in ℓ_∞ we take the maximum function over all of the stored rows, arguing that the approximation on these rows is good. Since we only need one such row to attain the maximum, the error from any of the dropped rows is small. For ℓ_1, we take the sum over all of the stored rows. Again, this can be related back to the objective function to argue that the cost of these rows is approximately preserved, however, if I is the stored set of rows which has size $\text{poly}\,(d)$, there are then $n - |I| = O(n)$ rows not in the summary. Hence, to argue that the cost is well-preserved for the entire instance would incur $O(n)$ error terms, meaning that we would have to choose $\varepsilon = O(1/n)$. From a different perspective, in the proof of Theorem 4.5.1 we used an appropriate choice of polynomial factors so that $\varepsilon = \tau\,\text{poly}\,(d)$. Setting $\varepsilon = O(1/n)$ is equivalent to asking for $\tau = O(1/n)$ and thus returning a summary which is not sublinear in n as we desired.

Chapter 5

Iterative Sketching for Least Squares Problems

Thus far, our concern has been with *finding* different data summaries depending on the type of task one might have in mind for some later analysis. Chapter 3 was concerned with how to find summaries for a variety of data analysis tasks when the features being queried are not known in advance. Similarly, in Chapter 4, the task was roughly to find a subset (or coreset) of the input whose distances were approximately preserved under various ℓ_p norms. By contrast, this chapter is mostly concerned with how one might *use* established summarisation techniques to better scale a particular part of the Machine Learning (ML) pipeline. In particular, we focus on the task of training a regression model using sketches. We focus solely on least-squares problems which are ubiquitous in ML.

Chapter Outline and Contributions

The theme of this chapter is to study sparse sketches for training regression models.

- **Section 5.1** details the key techniques and ideas that are necessary for this chapter.
- **Section 5.2** shows a *hardness* result. Namely, that the CountSketch suffers from the same deficiency as optimal dense sketches for estimating regression weights in a "one-shot" sketching model.
- **Section 5.3** shows that CountSketch can be used in the the IHS model for fast training of regression models to *high-accuracy*.
- **Section 5.4** provides preliminary empirical evidence to support the *scalability* of our approach, suggesting that the Iterative Hessian Sketch

with sparse sketches should be considered alongside other "optimal" sketches.

5.1 Introduction

We focus on "standard" least-squares setup, assuming the setup for overconstrained ($n \gg d$) least squares with a full-rank dataset $\mathbf{A} \in \mathbb{R}^{n \times d}$ and target vector $\mathbf{y} \in \mathbb{R}^n$. The loss (or objective) function is $f(\mathbf{x}) = 1/2 \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2$ and the task is to minimise $f(\mathbf{x})$ over some set $\mathcal{K}$ that may be the entire domain $\mathcal{K} = \mathbb{R}^d$ or a closed convex cone in $\mathbb{R}^d$. Within this special family of convex constrained least squares problems are popular methods tools such as Ordinary Least Squares (OLS) and penalised regression: $\mathcal{K} = \{\mathbf{x} : \|\mathbf{x}\|_p \leq t, p = 1, 2\}$ as well as Elastic Net Regression with $\mathcal{K} = \{\mathbf{x} : \|\mathbf{x}\|_1 \leq t_1, \|\mathbf{x}\|_2 \leq t_2\}$ and Support Vector Machines. A typical subroutine in solving these problems is to obtain singular vectors of $\mathbf{A}$ or to construct matrix $\mathbf{A}^\top\mathbf{A}$ for the normal equations. However, this can be costly, needing $O(nd^2)$ time and $O(d^2)$ space to generate and store $\mathbf{A}^\top\mathbf{A}$.

For certain problems within this general framework one can convert the constrained problem to an unconstrained, but penalised, problem through the use of a regularisation term. Such examples are the commonly used ridge regression [HK70], lasso [Tib11] and elastic net regression [ZH05]. For the sake of clarity, we are interested in *computational issues* that arise by using such models, rather than exploring which models are suited for certain applications.

There are various approaches which exploit sketching for regression that can be interpreted as analogues of the standard optimisation routines introduced in Section 2.3. Each method has its merits and drawbacks depending on the exact goals needing to be satisfied. We will study two settings for sketched regression. The first is a one-shot model for which we provide a result showing suboptimal recovery of the weights. The second shows how we will use sketches in an iterative scheme to solve least squares problems. This result will apply in the *multi-round optimisation model* of computation (Definition 2.1.5) which permits us to view the data and compute simple functions (such as inner products or gradients), but does not permit expensive matrix computations. The sketches will be used to approximate these difficult parts of the process so that the entire iterative scheme can be efficiently implemented. Before detailing how the optimisation is performed with sketches, we first introduce the structural quantity which underpins the randomised approaches.

5.1.1 Constructing ℓ_2 Subspace Embeddings

The key tool that we use is a $(1 \pm \varepsilon)$ *subspace embedding* for ℓ_2 which asks for *all directions* to be preserved up to small error. The following definition is restated from Section 2.2.1.

Definition 2.2.1. *[ℓ_2 Subspace Embedding] Let $\mathbf{A} \in \mathbb{R}^{n \times d}$. A matrix $\mathbf{S} \in \mathbb{R}^{m \times n}$ is a $(1 \pm \varepsilon)$ subspace embedding for the column space of $\mathbf{A}$ if for all $\mathbf{x} \in \mathbb{R}^d$:*

$$\|\mathbf{SAx}\|_2^2 = (1 \pm \varepsilon) \|\mathbf{Ax}\|_2^2 .$$

We note that Definition 2.2.1 has various equivalent formulations that will each in turn be useful. The subspace embedding property is often proved by showing:

$$\max_{\mathbf{x} \in \mathbb{R}^d} \left| \|\mathbf{SAx}\|_2^2 - \|\mathbf{Ax}\|_2^2 \right| < \varepsilon \|\mathbf{Ax}\|_2^2 . \tag{5.1}$$

Taking $\mathbf{x}$ to be the right singular vectors of $\mathbf{A}$, the subspace embedding condition is also equivalent to the following singular value characterisation:

$$(1 - \varepsilon)\sigma_i^2(\mathbf{A}) \leq \sigma_i^2(\mathbf{SA}) \leq (1 + \varepsilon)\sigma_i^2(\mathbf{A}). \tag{5.2}$$

Alternatively, if we write $\mathbf{A} = \mathbf{U\Sigma V}^\top$ then we can restrict attention to unit vectors $\mathbf{z}$ such that

$$\max_{\mathbf{z} \in \mathbb{R}^d : \|\mathbf{z}\|_2 = 1} \left| \|\mathbf{SUz}\|_2^2 - \|\mathbf{z}\|_2^2 \right| < \varepsilon.$$

Since $\mathbf{U}$ has orthonormal columns the subspace embedding condition is now equivalent to the following spectral norm condition in (5.3)

$$\left\| \mathbf{U}^\top \mathbf{S}^\top \mathbf{SU} - \mathbf{I}_d \right\|_2 < \varepsilon. \tag{5.3}$$

Different random projections can be used to obtain subspace embeddings; if the distribution is decided ahead of seeing the data, these are referred to as *oblivious* subspace embeddings as they can be sampled prior to observing the data. Oblivious sketches have the following parameters we must understand:

1. *Projection dimension m* - how large must we set m to satisfy Definition 2.2.1?

2. *Sketch time T_{sketch}* - how long does it take to compute the linear transformation $\mathbf{SA}$?

We define the following common random linear projections $\mathbf{S} : \mathbb{R}^n \to \mathbb{R}^m$ which map large n-vectors (e.g., the columns of $\mathbf{A}$) down to more manageable m-vectors with m independent of n. These properties are summarised in Table 5.1.

Method	Projection Dimension: m	Sketch Time: T_{sketch}
Gaussian	$d + \log(1/\delta)/\varepsilon^2$	$O(nd^2)$
SRHT	$(d + \log(1/\varepsilon\delta)\log(d/\delta)/\varepsilon^2$	$O(nd\log d)$
SJLT	$d\log(d/\delta)/\varepsilon^2$	$s\,\mathsf{nnz}\,(\mathbf{A})$
CountSketch	$(d^2+d)/\varepsilon^2\delta$	$\mathsf{nnz}\,(\mathbf{A})$

Table 5.1: Time and space costs to obtain oblivious subspace embeddings. All sketches except the CountSketch achieve the optimal embedding dimension of $O(d\varepsilon^{-2}\log(d/\delta))$.

Gaussian Sketch. Sample a *standard Gaussian* matrix $\mathbf{G}$ whose entries are iid normal $\mathbf{G}_{ij} \overset{\text{iid}}{\sim} \mathcal{N}(0,1)$ and define the *Gaussian sketch* as $\mathbf{S} = \mathbf{G}/\sqrt{m}$. The Gaussian sketch requires explicit dense matrix-matrix multiplication so requires time $O(mnd)$ to obtain $\mathbf{SA}$. We set $m = O((d+\log(1/\delta))/\varepsilon^2)$ for a subspace embedding so that $T_{\mathsf{sketch}} = \tilde{O}(nd^2)$. This often renders the Gaussian sketch as computationally time consuming as solving the original problem, for instance, OLS needing $O(nd^2)$ time.

Subsampled Randomised Hadamard Transform (SRHT) [AC06]. Assume that n is a power of 2. Define $\mathbf{S}$ as the following decomposition of three matrices: $\mathbf{D}$ with $\mathbf{D}_{ii} \overset{\text{iid}}{\sim} \{\pm 1\}$ with probabilty 1/2; $\mathbf{H} = \mathbf{H}^{(n)}$ is the recursively defined Hadamard transform of size n as defined below, and $\mathbf{P}$ is a matrix which samples m rows uniformly at random. Finally, set the sketch $\mathbf{S} = 1/\sqrt{nm}\mathbf{PHD}$. Note that the requirement of n being a power of 2 is purely for convenience to best exploit the Hadamard and Fast Fourier Transform. If this is not the case we can either pad the input with zeros, or use less restrictive forms of the Fourier transform such as the Discrete Cosine Transform or Discrete Hartley Transform [AMT10]. These transforms exist for all n but could increase the sample size in $\mathbf{P}$ by a factor of 2 [AMT10].

We use the notion that the size n Hadamard matrix is

$$\mathbf{H}^{(n)} = \frac{1}{\sqrt{2}}\begin{pmatrix} \mathbf{H}^{(n-1)} & \mathbf{H}^{(n-1)} \\ \mathbf{H}^{(n-1)} & -\mathbf{H}^{(n-1)} \end{pmatrix}.$$

Using $\mathbf{H}^{(0)} = 1$ means $\mathbf{H}^{(1)} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ and so on. The sketch will then use $\mathbf{H} = \mathbf{H}^{(n)}$. Because of the recursive nature of $\mathbf{H}$, it can be applied to a vector in $\mathbb{R}^n$ in $O(n\log n)$ time; indeed we apply it to the vector $\mathbf{Dx}$ which takes $O(n)$ time to obtain as $\mathbf{D}$ is a diagonal matrix. Finally, $\mathbf{P}$ performs simple uniform sampling of m from n items so needs only $O(m)$ time. Overall, to obtain a subspace embedding, we must take $m = O(d\log(d/\delta)/\varepsilon^2)$ and applying $\mathbf{S}$ to $\mathbf{A}$ takes $O(nd\log n)$ which is almost linear in the input size.

CountSketch. Initialise $\mathbf{S} = \mathbf{0}_{m\times n}$. Let $h : [n] \to [m]$ be a pairwise independent hash function and $\sigma : [n] \to \{\pm 1\}$ be a 4-wise independent random sign function for every row. The hash functions h and σ are independently chosen. We can think of h as choosing a single bucket from $[m]$ for every row index in $[n]$ and allocating a random sign to that row index. In this setting the sketch is the matrix $\mathbf{S} = \mathbf{RD}$ with the columns $\mathbf{R}$ being randomly chosen canonical basis vectors and $\mathbf{D}_{ii} \overset{\text{iid}}{\sim} \{\pm 1\}$. The time to obtain $\mathbf{SA}$ is $T_{\mathsf{sketch}} = O(\mathsf{nnz}\,(\mathbf{A}))$ which can be a huge saving if the data is very sparse; for instance when $\mathsf{nnz}\,(A) \ll nd$. Since $\mathbf{S}$ need only be implicitly defined by the hash functions h, σ, CountSketch can be applied as the data is observed so is suited to common sparse data structures.

Remarks on CountSketch. The CountSketch was first shown to satisfy Definition 2.2.1 in [CW13], albeit with a super-quadratic dependence on d. It was later shown in [NN13, MM13] that the embedding dimension could be improved to $m = O(d^2/\varepsilon^2\delta)$ which is the construction that we use; note that [MM13, Theorem 1] explicitly obtain $m = (d^2 + d)/\varepsilon^2\delta$. This is to be compared with $m = \tilde{O}(d\log(d/\delta)/\varepsilon^2)$ of other random projection methods. From a worst-case perspective the CountSketch is suboptimal. The projection dimension m has quadratic dependence on d unlike the optimal embedding dimesion of $O(d\log d)$ which is nearly-linear. The reciprocal linear dependence on the failure probability δ is also exponentially worse than the $\log(1/\delta)$ of all other methods.

Very recent work [LLW21] has shown that if a sketching algorithm is told whether a row of $\mathbf{A}$ is "important" for composing its column space then the embedding dimension can be improved to the standard $m = O(d\log(d/\delta)/\varepsilon^2)$. The idea is that these important rows have high leverage score, and are "perfectly hashed" to avoid collisions. The analysis then follows that of [KN14] to analyse the sketch and obtain the stated embedding dimension. Although such sketches perform better, this result needs oracle knowledge of whether a row is high leverage so more investigation is needed to understand the wider practicality of this new approach.

Note that in the data stream literature, the CountSketch refers to a matrix with more than one nonzero in every column which is used to answer queries for frequency statistics [TZ12, CCFC02]. A median operation is used to answer such queries with high probability which does not translate to an embedding. Consequently, we will exclusively use CountSketch to refer to the matrix $\mathbf{S}$ above with *exactly* one nonzero in every column. Although this may appear pedantic, it allows for distinction with the following family of random projections.

Sparse Johnson-Lindenstrauss Transform (SJLT) [KN14].
The Sparse Johnson-Lindenstrauss Transform of [KN14] can be seen as a generalisation of the CountSketch with s nonzeros per column rather than only 1. Conditions were given in [CJN18] that a distribution of sparse matrices must satisfy to achieve the subspace embedding guarantee with asymptotically optimal projection dimension m. Two simple constructions of the s-sparse SJLT $\mathbf{S}$ are to first generate a matrix $\mathbf{S}'$ by either:

1. Concatenating s independently sampled CountSketch matrices of size $m/s \times n$; or

2. Sampling exactly s nonzeros per column at random without replacement.

The sketch we use is then $\mathbf{S} = \mathbf{S}'/\sqrt{s}$ to ensure that $\mathbf{S}^\top\mathbf{S}$ is an isometry in expectation.

A benefit of the SJLT is that unlike the CountSketch, the optimal embedding dimension of $m = O(d\log(d/\delta)/\varepsilon^2)$ can be obtained. However, the extra nonzeros incur an increase in the sketch time by a factor of s to $T_{\mathsf{sketch}} = O(s\,\mathsf{nnz}\,(\mathbf{A}))$. Note that we must take $s = \Omega(1/\varepsilon)$ nonzeros to achieve a $1 \pm \varepsilon$ subspace embedding. Thus there is a non-negligible tradeoff in the accuracy and the time taken to apply the sketch, in practice we see that this is mild but noticeable.

Table 5.1 summarises the sketch properties for subspace embeddings. For simplicity we have omitted small poly logarithmic factors that obtain the "exact" asymptotically optimal construction, but for the purpose of comparison with CountSketch, the bounds presented suffice.

5.1.2 The Sketch-and-Solve Model

The sketch-and-solve approach is the sketching analogue of a direct solve for minimising a least squares function $f(\mathbf{x})$. These algorithms operate in a "one-shot" setting when the data is viewed only once. We can operate in either of the entrywise or row-arrival models from Section 2.1.4.

The main idea is to first sample a subspace embedding $\mathbf{S} \in \mathbb{R}^{m\times n}$. The matrix $\mathbf{S}$ embeds the large data (and possibly the target vector) into a more manageable size so that a direct solve is tractable. Recall that optimising over $f(\mathbf{x})$ usually requires an expensive operation. For ordinary least squares, we saw in Section 2.3.1 that the *setup time* to generate $\mathbf{A}^\top\mathbf{A}$ has a large cost of $O(nd^2)$. However, the sketch dimension should be chosen so that $m \ll n$ and these expensive operations are cheap. One can view this as trying to speed up the slow part of the pipeline, the *setup time cost*, and then use a black box solver which only needs poly (d/ε) time.

There are two typical approaches, the *classical sketch* and the *Hessian sketch*. For simplicity, we assume that the problem is unconstrained (i.e. $\mathcal{K} = \mathbb{R}^d$) to avoid nuances regarding the cost of projecting onto a set. Both of the subsequent methods still operate if constraints on the solution are required [Woo14b]. For unconstrained least squares problems, the property that we need of the random projection $\mathbf{S}$ is that it is a $1 \pm \varepsilon$ *subspace embedding* for the column space of $\mathbf{A}$. We will use m to denote the 'small' sketch dimension and the size of m should be thought of as being between $\tilde{O}(d)$ and $\tilde{O}(d^2)$ for CountSketch; thus the poly (d) solve time behaves as $O(d^3)$ but could be up to $O(d^4)$ if the worst-case bound for CountSketch is used.

Classical Sketch [Sar06]. In the *classical sketching* setting, $\mathbf{S}$ is applied to both data and target vector so that $\mathbf{SA}$ and $\mathbf{Sy}$ are generated. The sketched objective function is then:

$$f_{\mathbf{S}}(\mathbf{x}) = \frac{1}{2} \|(\mathbf{SA})\mathbf{x} - \mathbf{Sy}\|_2^2 .$$

Specifically, we will optimise in the 'sketch space' described by $\mathbf{SA}$ and $\mathbf{Sy}$ to obtain some $\mathbf{x}^C = \operatorname{argmin}_{\mathcal{K}} f_{\mathbf{S}}(\mathbf{x})$. For least squares problems one can view this as solving the following $d \times d$ linear system

$$\mathbf{A}^\top \mathbf{S}^\top \mathbf{S} \mathbf{A} \mathbf{x}^C = \mathbf{A}^\top \mathbf{S}^\top \mathbf{S} \mathbf{y}. \tag{5.4}$$

If constraints are present then $\mathbf{x}^C$ can be projected onto the constraint set. Any direct solver can be used to solve (5.4) in $O(md^2)$ time by evaluating the SVD of $\mathbf{SA}$ in comparison to method (1) of the direct solvers from Section 2.3.1.

Typically, these approaches provide guarantees on the loss function so that

$$f(\mathbf{x}^C) = (1 \pm \varepsilon) f(\mathbf{x}^\star)$$

which should be understood as the $\mathbf{x}^C$ approximately preserving the orginal objective function. One way of seeing this is that $\left\|\mathbf{S}(\mathbf{A}\mathbf{x}^C - \mathbf{y})\right\|_2^2 \leq \|\mathbf{S}(\mathbf{A}\mathbf{x}^\star - \mathbf{y})\|_2^2$ by the optimality of $\mathbf{x}^C$ in the sketched problem. By the subspace embedding property, the right hand side is at most $(1+\varepsilon)\|\mathbf{A}\mathbf{x}^\star - \mathbf{y}\|_2^2$ and after exploiting normal equations and orthogonality, these ideas can be extended to show:

$$f(\mathbf{x}^\star) \leq f(\mathbf{x}^C) \leq (1+\varepsilon) f(\mathbf{x}^\star) \tag{5.5}$$

$$\left\|\mathbf{A}(\mathbf{x}^C - \mathbf{x}^\star)\right\|_2^2 \leq \varepsilon^2 f(\mathbf{x}^\star) \tag{5.6}$$

The details can be found in [Woo14b, Theorems 21,23].[1]

[1] Note that their ϵ is our ε^2 as they use a different error parameterisation with a sketch

Hessian Sketch [PW15]. An alternative approach is the *Hessian sketching* method for least squares problems. These methods apply the transform $\mathbf{S}$ only to the data $\mathbf{A}$ and leave the vector $\mathbf{A}^\top\mathbf{y}$ intact. One can see this from the following quadratic programming formulation of least squares regression:

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{2}\mathbf{x}^\top\mathbf{A}^\top\mathbf{A}\mathbf{x} - \mathbf{x}^\top\mathbf{A}^\top\mathbf{y}.$$

In this quadratic, we recognise the degree-2 term is the same operator as the covariance matrix which is the quantity we estimate under subspace embeddings (Section 5.1.1). Thus, Hessian sketch seeks to only sketch the quadratic term, leaving the linear term $\mathbf{x}^\top\mathbf{A}^\top\mathbf{y}$ intact, yielding the following solution:

$$\mathbf{x}^H = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{2}\mathbf{x}^\top\mathbf{A}^\top\mathbf{S}^\top\mathbf{S}\mathbf{A}\mathbf{x} - \mathbf{x}^\top\mathbf{A}^\top\mathbf{y}.$$

Hence, we have the augmented normal equations:

$$\mathbf{A}^\top\mathbf{S}^\top\mathbf{S}\mathbf{A}\mathbf{x}^H = \mathbf{A}^\top\mathbf{y}. \tag{5.7}$$

The Hessian Sketch method obtains a bound

$$\left\|\mathbf{A}(\mathbf{x}^H - \mathbf{x}^\star)\right\|_2 \leq \frac{\varepsilon}{1-\varepsilon}\left\|\mathbf{A}\mathbf{x}^\star\right\|_2 \tag{5.8}$$

which is argued through first-order optimality conditions [PW16]. This approach forms the basis of an iterative scheme that we will introduce in Section 5.1.3.

Computational Considerations

Asymptotically, the time and space cost of classical and Hessian sketching is no different. Both require storing the sketch $\mathbf{SA}$ of size $m \times d$ which takes time T_{sketch} depending on the family from which $\mathbf{S}$ is sampled. The solve time of $O(md^2)$ is $\tilde{O}(d^3)$ for all sketches except the CountSketch as $m = O(d\log(d/\delta)\varepsilon^2)$. If the CountSketch is used with a worst-case projection dimension of $m = O(d^2)$, the solve time theoretically could be $O(d^4)$. Overall, the time to approximate regression to ε accuracy for all sketches in the sketch-and-solve model is $T_{\mathsf{sketch}} + \text{poly}\left(d/\varepsilon^2\right)$.

Although this can be substantially faster than the $O(nd^2 + d^3)$ direct solver methods when $\mathbf{S}$ is appropriately chosen, we have an inverse quadratic relationship between the sketch dimension m and the accuracy ε. This is fine if ε is a small constant, say $[10^{-3}, 1/2]$. However, suppose that we require accuracy on the order of say $\varepsilon \approx 10^{-4}$. Such ε would incur $m = O(10^8)$ rows in

size of $O(1/\epsilon)$ for a subspace embedding, which is the same as our $O(1/\varepsilon^2)$.

the sketch. Clearly, this is not appropriate as the projection dimension should be small in comparison to n for scalability while retaining competitive accuracy to an exact solver.

5.1.3 Iterative Sketching Framework

As discussed in Section 5.1.2, it may be the case that significantly higher accuracy is needed, for example, on the order of 10^{-8} rather than 10^{-3}. To achieve such a guarantee, we relax the requirement of a single-pass algorithm in favour of an algorithm that can revisit the data. These are the so-called iterative approaches which use cheap estimates of solution vectors that aggregate to a high accuracy estimate of the optimal solution vector.

For the least squares objective, the approach follows a similar approach as for gradient descent and the approximate Newton Method from Section 2.3. We operate in the multi-round optimisation model from Definition 2.1.5. We introduce an approximation $\hat{\mathbf{H}} = \mathbf{A}^\top \mathbf{S}^\top \mathbf{S}\mathbf{A}$ to the Hessian $\mathbf{H} = \mathbf{A}^\top \mathbf{A}$. Informally, for OLS if we have

$$\begin{aligned}
\mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \hat{\mathbf{H}}^{-1} \nabla f(\mathbf{x}^{(t)}) \\
\left\| \mathbf{x}^{(t+1)} - \mathbf{x}^\star \right\|_2 &\leq \left\| \mathbf{I}_d - \hat{\mathbf{H}}^{-1} \mathbf{A}^\top \mathbf{A} \right\|_2 \left\| \mathbf{x}^{(t)} - \mathbf{x}^\star \right\|_2 \\
&\vdots \\
&\leq \left\| \mathbf{I}_d - \hat{\mathbf{H}}^{-1} \mathbf{A}^\top \mathbf{A} \right\|_2^{t+1} \left\| \mathbf{x}^\star \right\|_2 .
\end{aligned}$$

The error at every step can be controlled through matrix similarity on $\mathbf{I}_d - \hat{\mathbf{H}}^{-1}\mathbf{A}^\top\mathbf{A}$ by pre and post multiplying by $\hat{\mathbf{H}}^{1/2}, \hat{\mathbf{H}}^{-1/2}$, respectively. Hence,

$$\begin{aligned}
\left\| \mathbf{I}_d - \hat{\mathbf{H}}^{-1} \mathbf{A}^\top \mathbf{A} \right\|_2 &= \left\| \mathbf{I}_d - \hat{\mathbf{H}}^{-1/2} \mathbf{A}^\top \mathbf{A} \hat{\mathbf{H}}^{-1/2} \right\|_2 && (5.9) \\
&\leq \frac{\varepsilon}{1-\varepsilon}. && (5.10)
\end{aligned}$$

The inequality follows from the extremal values of the Rayleigh quotient:[2]

$$\begin{aligned}
\lambda_{\min}(\hat{\mathbf{H}}^{-1/2} \mathbf{A}^\top \mathbf{A} \hat{\mathbf{H}}^{-1/2}) &= \min_{\mathbf{u}} \frac{\mathbf{u}^\top \hat{\mathbf{H}}^{-1/2} \mathbf{A}^\top \mathbf{A} \hat{\mathbf{H}}^{-1/2} \mathbf{u}}{\mathbf{u}^\top \mathbf{u}} \\
\lambda_{\max}(\hat{\mathbf{H}}^{-1/2} \mathbf{A}^\top \mathbf{A} \hat{\mathbf{H}}^{-1/2}) &= \max_{\mathbf{u}} \frac{\mathbf{u}^\top \hat{\mathbf{H}}^{-1/2} \mathbf{A}^\top \mathbf{A} \hat{\mathbf{H}}^{-1/2} \mathbf{u}}{\mathbf{u}^\top \mathbf{u}}.
\end{aligned}$$

[2] If $\mathbf{M} \in \mathbb{R}^{d \times d}$ is a real symmetric matrix then the *Rayleigh quotient* is $R(\mathbf{M}, \mathbf{x}) = \mathbf{x}^\top \mathbf{M}\mathbf{x} / \mathbf{x}^\top \mathbf{x}$. The quotient is maximised (respectively, minimised) when $\mathbf{x}$ is the top (bottom) unit-length eigenvector $\mathbf{v}_1$ ($\mathbf{v}_d$) of $\mathbf{M}$ and achieves $R(\mathbf{M}, \mathbf{v}_1) = \lambda_1(\mathbf{M})$ (or $R(\mathbf{M}, \mathbf{v}_d) = \lambda_d(\mathbf{M})$). Applying this to the real symmetric matrix $\mathbf{M} = \mathbf{A}^\top \mathbf{A}$ shows that $\sigma^2_{\min}(\mathbf{A}) \leq \max_{\mathbf{x}} R(\mathbf{M}, \mathbf{x}) \leq \sigma^2_{\max}(\mathbf{A})$. The bounds are attained at by using the bottom and top unit length singular vectors, respectively.

For a unit vector $\mathbf{u}$ we can apply the change of variables $\mathbf{z} = \hat{\mathbf{H}}^{-1/2}\mathbf{u}$ to obtain

$$\mathbf{u}^\top \hat{\mathbf{H}}^{-1/2}\mathbf{A}^\top \mathbf{A}\hat{\mathbf{H}}^{-1/2}\mathbf{u} = \frac{\mathbf{z}^\top \mathbf{H}\mathbf{z}}{\mathbf{z}^\top \hat{\mathbf{H}}\mathbf{z}}$$

and $\mathbf{z}^\top \hat{\mathbf{H}}\mathbf{z} = (1 \pm \varepsilon)\mathbf{z}^\top \mathbf{H}\mathbf{z}$ due to the subspace embedding. Thus;

$$\lambda_{\min}(\hat{\mathbf{H}}^{-1/2}\mathbf{A}^\top \mathbf{A}\hat{\mathbf{H}}^{-1/2}) \geq \frac{1}{1+\varepsilon}$$
$$\lambda_{\max}(\hat{\mathbf{H}}^{-1/2}\mathbf{A}^\top \mathbf{A}\hat{\mathbf{H}}^{-1/2}) \leq \frac{1}{1-\varepsilon}$$

Ensuring Equation (5.9) is less than 1, is equivalent to:

$$\begin{aligned} & |1 - \lambda_{\max}(\hat{\mathbf{H}}^{-1/2}\mathbf{A}^\top \mathbf{A}\hat{\mathbf{H}}^{-1/2})| < 1 \\ \iff & \left|1 - \frac{1}{1-\varepsilon}\right| < 1 \\ \iff & \left|\frac{\varepsilon}{1-\varepsilon}\right| < 1 \end{aligned}$$

which occurs provided $\varepsilon \leq 1/2$, that is, we obtain no worse than a $1/2$-subspace embedding.

To achieve error of $\varepsilon_\star$ after T iterations set $\rho = \varepsilon/1-\varepsilon$, then

$$T = \frac{\log(1/\varepsilon_\star)}{\log(1/\rho)} \tag{5.11}$$

iterations are required. In comparison to gradient descent over strongly convex functions, we saw in (2.10), Section 2.3.2, that the number of iterations needed was

$$T_{\mathsf{GD}} = O\left(\frac{\log(1/\varepsilon_\star)}{\log(1/E)}\right).$$

However, as $E = 1 - \sigma^2_{\min}(\mathbf{A})/\sigma^2_{\max}(\mathbf{A})$, we see that $1/\log(1/E)$ is large when the ratio $\sigma^2_{\min}(\mathbf{A})/\sigma^2_{\max}(\mathbf{A})$ is small.[3] On the other hand, we can control the denominator $\log(1/\rho)$ of (5.11) as it is a direct consequence of an ε-subspace embedding. Hence, we may think of T as being a small constant number of steps that each require $O(d^3)$ time. Again, this should be contrasted with gradient descent which may need a very large number of steps that each require $O(d)$ space and $O(nd)$ update time.

Approximate Newton method moves the difficulty of the optimisation problem onto estimating the spectrum of the input data. Nevertheless, there is a $O(d^2)$ discrepancy between the two iteration time costs. Therefore, a tradeoff must be made for which computing *and applying* $\hat{\mathbf{H}}^{-1}$ is efficient, but retains enough spectral information about $\mathbf{A}$ so that $\hat{\mathbf{H}}^{-1}\nabla f(\mathbf{x})$ roughly acts

[3] This is known as an ill-conditioned matrix, when the largest singular value is much larger than the smallest singular value.

like $\mathbf{H}^{-1}\nabla f(\mathbf{x})$ to ensure the number of steps completed is small.

5.1.4 Comparison of Sketch-and-solve and Iterative Sketching Frameworks

Thus far we have been informal about how the two methods compare. We do so by studying how the approximate weights found using sketches compare to the "best" weights that could be found with no computational constraints. For such a task, the sketch-and-solve model appears attractive: it permits single-pass algorithms on either entrywise or row-arrival matrix streams with useful bounds on how the returned estimates compare to the optimal weights. However, the sketch-and-solve approximation bounds Equations (5.4) and (5.8) only provide weak constant factor guarantees on the behaviour of estimated weights. The Iterative Hessian Sketch framework is motivated by this weaknesses of the sketch-and-solve model.

If $\mathbf{x}^\star$ are the optimal weights for some optimisation problem then we want the estimate weights $\hat{\mathbf{x}}$ to behave roughly as $\mathbf{x}^\star$. For regression tasks, this means understanding the prediction inner product $\langle \mathbf{A}_i, \hat{\mathbf{x}} \rangle$ for a query data point $\mathbf{A}_i$ and asking how $\langle \mathbf{A}_i, \hat{\mathbf{x}} - \mathbf{x}^\star \rangle$ behaves. Over a collection of training points $1 \leq i \leq n$, this inner product can be summed to obtain the predictive semi-norm $\|\mathbf{z}\|_{\mathbf{A}}^2 = {}^1\!/_n \|\mathbf{A}\mathbf{z}\|_2^2$ where the vector of interest will often be $\mathbf{z} = \hat{\mathbf{x}} - \mathbf{x}^\star$. Pilanci and Wainwright formalised the weakness of the sketch-and-solve model through the following argument and some of these results will guide our later commentary. We use a linear model

$$\mathbf{y} = \mathbf{A}\mathbf{x}^\dagger + \boldsymbol{\omega} \tag{5.12}$$

with $\mathbf{A} \in \mathbb{R}^{n \times d}$ and noise $\boldsymbol{\omega} \sim \mathcal{N}\left(\mathbf{0}_{n\times 1}, \sigma^2 \mathbf{I}_n\right)$ so that $\mathbf{y} \sim \mathcal{N}\left(\mathbf{A}\mathbf{x}^\dagger, \sigma^2 \mathbf{I}_n\right)$. The *model weights* $\mathbf{x}^\dagger$ are not observed in practice. Then for any estimator $\mathbf{x}'$, the *expected solution error* is the quantity:

$$\mathbb{E}_{\boldsymbol{\omega}} \left\| \mathbf{x}' - \mathbf{x}^\dagger \right\|_{\mathbf{A}}^2 .$$

Under this setup [PW16] demonstrates that an optimal estimator $\mathbf{x}' = \mathbf{x}^\star$ has expected solution error $\mathbb{E}_{\boldsymbol{\omega}} \left\| \mathbf{x}^\star - \mathbf{x}^\dagger \right\|_{\mathbf{A}}^2 = O(\sigma^2/n)$, characterised in the following proposition.

Proposition 5.1.1 ([PW16])**.** *Let $\mathcal{K}$ be an arbitrary convex constraint set and $\mathbf{A}, \mathbf{y}$ be given by the linear model* (5.12)*. If $\mathbf{x}^\star = \operatorname{argmin}_{\mathbf{x}\in\mathcal{K}} \frac{1}{2} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2$, then $\left\| \mathbf{x}^\star - \mathbf{x}^\dagger \right\|_{\mathbf{A}}^2 = O(\Delta_{\mathcal{K}}/n)$ for a problem-dependent parameter $\Delta_{\mathcal{K}} \leq \sigma^2 d$. A special case is ordinary least squares which has $\Delta_{\mathcal{K}} = \sigma^2 d$*

However, the same is not true of one-shot sketches. In the same work,

[PW16] also shows that *any* estimated weights $\hat{\mathbf{x}}$ returned solely on the information $(\mathbf{SA}, \mathbf{Sy})$ can obtain $\mathbb{E}\left\|\hat{\mathbf{x}} - \mathbf{x}^\dagger\right\|_{\mathbf{A}}^2 = O(\sigma^2 d/n)$, *but only if the projection dimension* $m = \Omega(n)$. Given that we want to use sketches to reduce the burden on n, this is problematic. The key result is the following theorem.

Theorem 5.1.1 ([PW16, Theorem 1])**.** *Let* $\mathbf{A}, \mathbf{y}$ *be the data and targets for a linear model* (5.12) *and let* $\mathcal{K}$ *be an arbitrary convex constraint set. Suppose that* $\mathbf{S} \in \mathbb{R}^{m \times n}$ *is a random projection satisfying the spectral norm bound*

$$\left\| \mathbb{E}_{\mathbf{S}} \left[\mathbf{S}^\top \left(\mathbf{S}\mathbf{S}^\top \right)^{-1} \mathbf{S} \right] \right\|_{op} \leq \frac{\eta m}{n}. \tag{5.13}$$

Then any weights

$$\hat{\mathbf{x}} = \operatorname*{argmin}_{\mathbf{x} \in \mathcal{K}} \frac{1}{2} \left\| \mathbf{S}(\mathbf{A}\mathbf{x} - \mathbf{y}) \right\|_2^2$$

have

$$\sup_{\mathbf{x}^\dagger \in \mathcal{K}} \mathbb{E}_{\mathbf{S},\boldsymbol{\omega}} \left\| \hat{\mathbf{x}} - \mathbf{x}^\dagger \right\|_{\mathbf{A}}^2 = \Omega(1/m)$$

A consequence of this result is that for sketched weights $\hat{\mathbf{x}}$ to achieve comparable performance to optimal weights $\mathbf{x}^\star$, we need m to grow linearly with n. This can be seen through

$$\frac{c}{m} \leq \mathbb{E}_{\mathbf{S},\boldsymbol{\omega}} \left\| \hat{\mathbf{x}} - \mathbf{x}^\dagger \right\|_{\mathbf{A}}^2 \leq \frac{c'}{n}$$

where the lower bound follows from Theorem 5.1.1 and the upper bound is the desired accuracy from Proposition 5.1.1. Consequently, $m = \Omega(n)$.

In summary, optimal weights for regression problems have their predictive performance improving as $1/n$ [PW16]. Ideally, we would like our sketched weights to nearly match the predictive power of an optimal estimator. However, Theorem 5.1.1 shows this cannot be done using one sketch in with $m = o(n)$. We will show that the condition (5.13) is achieved by the CountSketch in Section 5.2. Following this, we will show how the subspace embedding can be used in an iterative sketching setup to achieve the reciprocal error rate with only small sketches. The price to pay for this is that we have to adopt the multi-round optimisation model, rather than a single-pass matrix stream.

5.2 CountSketch in the Sketch-and-Solve Model

We motivate the usage of CountSketch in an iterative model by first showing a negative result. Our result shows that the error when using CountSketch in the sketch-and-solve model is always lower bounded by a suboptimal constant. This is to be contrasted with the optimal estimator $\mathbf{x}^\star$ approaching $\mathbf{x}^\dagger$ as n increases. The main result, Theorem 5.2.1, involves understanding certain

spectral properties of compositions of CountSketch matrices. Following this negative result, we show sufficient conditions under which the CountSketch can be easily 'plugged into' an iterative model to achieve high-accuracy in the subsequent section.

It is sufficient to show that a CountSketch matrix $\mathbf{S}$ has the spectral property from Theorem 5.1.1. We can then instantiate the proof of Theorem 5.1.1 from [PW16] with our result for the CountSketch. Our main contribution is the following theorem that demonstrates the necessary spectral property.

Theorem 5.2.1. *Let $\mathbf{S} \in \{0, \pm 1\}^{m \times n}$ be a CountSketch matrix with no all-zero rows. Then*

$$\left\| \mathbb{E}\left[\mathbf{S}^\top \left(\mathbf{S}\mathbf{S}^\top \right)^{-1} \mathbf{S} \right] \right\|_{op} \leq \frac{\eta m}{n}. \tag{5.14}$$

It transpires that due to the simple combinatorial structure of the CountSketch we can analyse the quantity (5.14) fairly easily. The main idea is to analyse the CountSketch as a random linear transform $\mathbf{S} = \mathbf{P}\mathbf{D}$ with $\mathbf{P} \in \{0, 1\}^{m \times n}$ the matrix that distributes the n rows of a length n input matrix into m buckets. Meanwhile, $\mathbf{D} \in \{0, \pm 1\}^{n \times n}$ is a diagonal matrix which is determined by the random sign function used in the CountSketch definition (i.e., for every row we assign a ± 1 with equal probability). Note that the restriction on all-zero rows is a convenient simplification since the proofs involve a matrix inverse which is not guaranteed to exist if a sketch has a row entirely zero. However, this is not a limitation as we can simply consider the sketch with an all-zero row removed (corresponding to buckets that have no input elements mapped to them). For a fixed row i, the probability that bucket j is not selected is $\mathbf{S}_{ij} = 0$ is $1 - 1/m$, so the probability that $\mathbf{S}_{i*} = \mathbf{0}_{1 \times n}$ is entirely zero is $(1 - 1/m)^n$ which approaches 0 quickly as n becomes large. Before proving Theorem 5.2.1, we need some supporting lemmas to aid the analysis which are simple consequences of the structure of $\mathbf{S}$.

5.2.1 Structural Properties of CountSketch

Lemma 5.2.1. *Let $\mathbf{S}$ be a CountSketch matrix and let N_i denote the number of nonzeros in the row $\mathbf{S}_i$. Then $\mathbf{S}\mathbf{S}^\top$ is a diagonal matrix with $(\mathbf{S}\mathbf{S}^\top)_{ii} = N_i$ and hence distinct rows of $\mathbf{S}$ are orthogonal. Secondly, $\mathbb{E}(\mathbf{S}\mathbf{S}^\top) = n/m \mathbf{I}_m$.*

Proof. The entries of the matrix $\mathbf{S}\mathbf{S}^\top$ are given by the inner products between pairs of rows of $\mathbf{S}$. Hence we consider the inner products $\langle \mathbf{S}_i, \mathbf{S}_j \rangle$. By construction, $\mathbf{S}$ has exactly one non-zero entry in each columns; thus for $i \neq j$ and a column index $1 \leq k \leq n$ we must have $\mathbf{S}_{ik}\mathbf{S}_{jk} = 0$ meaning that $\langle \mathbf{S}_i, \mathbf{S}_j \rangle = 0$ which establishes orthogonality between the rows. Meanwhile, the diagonal entries are given by $\|\mathbf{S}_i\|_2^2 = \sum_{j=1}^n \mathbf{S}_{ij}^2$ which is 1 if and only if $\mathbf{S}_{ij}$ is nonzero so this counts the number of nonzeros N_i in $\mathbf{S}_i$.

We can extend this analysis similarly by taking the expectation $\mathbb{E}(\langle \mathbf{S}_i, \mathbf{S}_j \rangle) = \sum_{k=1}^{n} \mathbf{S}_{ik}\mathbf{S}_{jk}$. By the orthogonal property it suffices to check the case $i = j$ as otherwise $\langle \mathbf{S}_i, \mathbf{S}_j \rangle = 0$. Hence, we have a sum of n random entries which have $\mathbf{S}_{ik}^2 = 1$ with probability $1/m$ (coming from the two events $\mathbf{S}_{ik} = \pm 1$ each with probability $1/m$). Finally, by linearity of expectation $\mathbb{E}(\langle \mathbf{S}_i, \mathbf{S}_i \rangle) = n/m$. □

The above arguments are made to understand the behaviour over pairs of rows, we can make a similar argument for the *columns* of a CountSketch.

Lemma 5.2.2. *Let* $\mathbf{S}$ *be a CountSketch matrix. Then* $\mathbb{E}(\mathbf{S}^\top \mathbf{S}) = \mathbf{I}_n$.

Proof. Observe that $(\mathbf{S}^\top \mathbf{S})_{ij}$ is the matrix whose entries are inner products between *columns* of the sketch matrix $\mathbf{S}$. Recalling the construction of $\mathbf{S}$, we see that column i of $\mathbf{S}$ is a signed canonical vector so $\mathbf{S}^i = \pm \mathbf{e}_{h(i)}$. Thus, $\langle \mathbf{S}^i, \mathbf{S}^j \rangle$ is zero if $h(i) \neq h(j)$ as $\mathbf{e}_{h(i)}, \mathbf{e}_{h(j)}$ have disjoint support. Otherwise, when $h(i) \neq h(j)$, it suffices to consider the single product $\mathbf{S}_{h(i),i} \cdot \mathbf{S}_{h(j),j}$ which is 1 when the two entries have the same sign allocated, or -1 if they have opposite signs. These two events have equal probability so are zero in expectation. In summary, $\mathbb{E}(\mathbf{S}^\top \mathbf{S})_{ij} = 1$ if $i = j$ and is zero otherwise. □

5.2.2 Spectral Properties of CountSketch

Recall that we must show for a constant η independent of m and n the bound from (5.14)

$$\left\| \mathbb{E}\left[\mathbf{S}^\top \left(\mathbf{S}\mathbf{S}^\top \right)^{-1} \mathbf{S} \right] \right\|_{\text{op}} \leq \frac{\eta m}{n}.$$

We start with a lemma on the behaviour of Bernoulli random variables which supports the analysis.

Lemma 5.2.3. *Let* X_i *be independently identically distributed Bernoulli random variables with parameter* p *and* $X = \sum_{i=0}^{n} X_i$. *Then:*

$$\mathbb{E}\left(\frac{1}{1+X} \right) = \frac{1 - (1-p)^{n+1}}{(n+1)p}.$$

Proof. Observe that X is a sum of Bernoulli random variables so shares the Binomial probability mass function, hence:

$$\begin{aligned}
\mathbb{E}\left(\frac{1}{1+X} \right) &= \sum_{k=0}^{n} \frac{1}{1+k} \binom{n}{k} p^k (1-p)^{n-k} \\
&= \frac{1}{n+1} \sum_{k=0}^{n} \binom{n+1}{k+1} p^k (1-p)^{n-k} \\
&= \frac{1}{p(n+1)} \sum_{k=0}^{n} \binom{n+1}{k+1} p^{k+1} (1-p)^{n-k}.
\end{aligned}$$

The second equality follows from the identity $(k+1)\binom{n+1}{k+1} = (n+1)\binom{n}{k}$. Then apply a change of variables $N = n+1, j = k+1$ to recognise that $\sum_{j=1}^{N}\binom{N}{j}p^j(1-p)^{N-j}$ is a sum over the entire Binomial distribution except the first term, $\binom{N}{0}p^0(1-p)^N$ which establishes the claim. □

We are finally in a position to prove Theorem 5.2.1. The main idea here is to use our knowledge of the structure of the CountSketch matrices to reduce the matrix $\mathbf{S}^\top\left(\mathbf{S}\mathbf{S}^\top\right)^{-1}\mathbf{S}$ instead to a scaled variant of $\mathbf{S}^\top\mathbf{S}$. Subsequently, we can show that the only necessary entries (in expectation) are a sum of independent Bernoullis which will allow us to use Lemma 5.2.3.

Proof of Theorem 5.2.1. We analyse the matrix $\mathbb{E}\left[\mathbf{S}^\top\left(\mathbf{S}\mathbf{S}^\top\right)^{-1}\mathbf{S}\right]$ and show that it is a diagonal matrix whose entries are a constant multiple of m/n. First we deal with the terms $\mathbf{S}\mathbf{S}^\top$ which, by Lemma 5.2.1 is a diagonal matrix whose entries are N_i, the number of nonzeros in row $\mathbf{S}_i$. Hence, we write $\mathbf{D} = \mathbf{S}\mathbf{S}^\top$ with entries $\mathbf{D}_{ii} = N_i$ and off-diagonal entries all zero. By the assumption that $\mathbf{D}_{ii} > 0, \mathbf{D}^{-1}$ exists such that $\mathbf{D}_{ii}^{-1} = 1/N_i$; hence write $\mathbf{S}^\top\left(\mathbf{S}\mathbf{S}^\top\right)^{-1}\mathbf{S} = \mathbf{S}^\top\mathbf{D}^{-1}\mathbf{S}$. Next, we may distribute the entries of $\mathbf{D}^{-1}$ as $\mathbf{S}^\top\mathbf{D}^{-1/2}\mathbf{D}^{-1/2}\mathbf{S}$ which is well- defined as $\mathbf{D}^{-1}$ is both diagonal and positive. Finally, set $\bar{\mathbf{S}} = \mathbf{D}^{-1/2}\mathbf{S}$ which is simply $\mathbf{S}$ but scaled by its row norms as:

$$\bar{\mathbf{S}}_{ij} = \frac{\mathbf{S}_{ij}}{\sqrt{N_i}} \tag{5.15}$$

This expresses our matrix of interest as a matrix $\bar{\mathbf{S}}^\top\bar{\mathbf{S}}$ which is the collection of scaled inner products between the columns of $\mathbf{S}$. In particular, for every column $\bar{\mathbf{S}}^j$, the unique nonzero entry is located at $\bar{\mathbf{S}}_{h(j),j}$ and takes value $\mathbf{S}_{h(j),j}$.

Since $\bar{\mathbf{S}}$ has the same sparsity structure as $\mathbf{S}$ and its entries are just rescaled versions of those in $\mathbf{S}$, we know from Lemma 5.2.2 that in expectation the only entries we need consider are those on the diagonal of $\bar{\mathbf{S}}^\top\bar{\mathbf{S}}$. Then the inner product to consider is $\langle\bar{\mathbf{S}}_i, \bar{\mathbf{S}}_i\rangle$, which is exactly the column norm of $\bar{\mathbf{S}}_i$. Equation (5.15) means that $(\bar{\mathbf{S}}^\top\bar{\mathbf{S}})_{ii} = 1/N_{h(i)}$ and Lemma 5.2.3 can be used to bound $1/N_{h(i)}$ in expectation as it is a sum of Bernoulli random variables. Indeed, $N_{h(i)} = \sum_{j=1}^{n}\mathbf{1}(\mathbf{S}^2_{h(i),j})$ where $\mathbf{1}(\mathbf{S}^2_{h(i),j})$ is an indicator variable for the presence of a nonzero at location $\mathbf{S}_{h(i),j}$. More concretely $\mathbf{1}(\mathbf{S}^2_{h(i),j}) = 1$ when j is hashed to the same bucket as i. No row of $\mathbf{S}$ is identically zero so certainly $\mathbf{S}^2_{h(i),i} = 1$ and for any $j \neq i$ then $\mathbf{1}(\mathbf{S}_{h(i),j}) = 1$ with probability $1/m$; for a given column, non-zero rows are chosen uniformly at random in the CountSketch construction. This is exactly when $h(j) = h(i)$ so:

$$N_{h(i)} = \sum_{j=1}^{n}\mathbf{S}^2_{h(i),j} = 1 + \sum_{\substack{j\neq i\\ h(j)=h(i)}}^{n}\mathbf{S}^2_{h(i),j}. \tag{5.16}$$

Hence, we may write $N_{h(i)} = 1 + X$ and invoke Lemma 5.2.3 with $p = 1/m$ and $n - 1$ trials (due to the $n - 1$ columns $j \neq i$) to obtain:

$$\begin{aligned}\mathbb{E}\left(\frac{1}{1+X}\right) &= \frac{1-(1-1/m)^n}{n/m} \\ &\leq \frac{1}{n/m} = \frac{m}{n}.\end{aligned}$$

Thus, we have shown that $\mathbb{E}\left[\bar{\mathbf{S}}^\top \bar{\mathbf{S}}\right]_{ij} \leq m/n$ if $i = j$ and is zero otherwise, hence, $\left\|\mathbb{E}\left[\bar{\mathbf{S}}^\top \bar{\mathbf{S}}\right]_{ij}\right\|_{\text{op}} \leq m/n$ and (5.14) is met with $\eta = 1$, independent of n and m, as required. □

With this in hand we have the following corollary.

Corollary 5.2.1. *Let $\mathbf{S} \in \mathbb{R}^{m\times n}$ be a CountSketch subspace embedding. Suppose that the regression problem $\mathbf{y} = \mathbf{A}\mathbf{x}^\dagger + \boldsymbol{\omega}$ is defined as in* (5.12)*. Then the classical sketch-and-solve technique with $\mathbf{S}$ has expected error lower bounded as:*

$$\mathbb{E}_{\mathbf{S},\boldsymbol{\omega}} \left\|\hat{\mathbf{x}} - \mathbf{x}^\dagger\right\|_{\mathbf{A}}^2 = \Omega(1/m). \tag{5.17}$$

Proof. This is simply the combination of Theorem 5.1.1 and Theorem 5.2.1. □

In summary, having established Theorem 5.2.1 we have shown that in order for a sketch-and-solve algorithm with CountSketch to obtain error comparable to the optimal estimator requires m to grow linearly with n. At its heart, this is due to the algorithm's ε-error depending on ε^{-2} for the projection dimension. Next, we will study an algorithm that achieves $\varepsilon_\star \ll \varepsilon$ error and whose running time depends on $\log(1/\varepsilon_\star)$. However, we must sacrifice the requirement of only visiting the data once to achieve such a guarantee.

5.3 CountSketch in the IHS model

Next we present the second technical contribution of this chapter. This section shows how CountSketch can be used within the Iterative Hessian Sketch (IHS) framework. Iterative Hessian Sketch is motivated by the the sketch-and-solve weights $\hat{\mathbf{x}}$ being suboptimal estimators of the model weights $\mathbf{x}^\dagger$, as presented in the preceding section. We now make two changes as we adopt the multi-round optimisation model:

(i) Multiple passes over the data are permitted;

(ii) An independent sketch is generated for every pass.

Our contribution here is to show that subspace embeddings (Section 5.1.1) are sufficient for convergence under the IHS scheme.

Algorithm 8: Iterative Hessian Sketch (IHS)

Input: Data $\mathbf{A} \in \mathbb{R}^{n \times d}$, targets $\mathbf{y} \in \mathbb{R}^n$, sketch size m, number of iterations $T \geq 1$, random projection method Sketch from Table 5.1

Output: Weights $\hat{\mathbf{x}} \in \mathbb{R}^d$

1 $\mathbf{SA} = \mathsf{Sketch}(\mathbf{A})$
2 $\mathbf{z} = \mathbf{A}^\top \mathbf{y}$
3 $\mathbf{x}^{(0)} = \mathbf{0}_d$
4 **for** $i = 1 : T$ **do**
5 | Solve $(\mathbf{SA})^\top (\mathbf{SA})\mathbf{z} = -\nabla f(\mathbf{x}^{(t)})$
6 | $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \mathbf{z}$
7 **end**
8 $\hat{\mathbf{x}} = \mathbf{x}^{(T)}$

Recall that the optimal weights are $\mathbf{x}^\star$ and the estimate weights are $\hat{\mathbf{x}}$. By the triangle inequality, we have

$$\|\mathbf{x}^\dagger - \hat{\mathbf{x}}\|_{\mathbf{A}} \leq \|\mathbf{x}^\dagger - \mathbf{x}^\star\|_{\mathbf{A}} + \|\mathbf{x}^\star - \hat{\mathbf{x}}\|_{\mathbf{A}}.$$

The first term $\|\mathbf{x}^\dagger - \mathbf{x}^\star\|_{\mathbf{A}}$ is problem dependent *"model error"*. The second term $\|\mathbf{x}^\star - \hat{\mathbf{x}}\|_{\mathbf{A}}$ is often that which we seek to minimise in a numerical scheme, the *"optimisation error"*. We can think of candidate weights $\hat{\mathbf{x}}$ being a good approximation for $\mathbf{x}^\star$ if the optimisation error is small.

Let $z^\star = \|\mathbf{x}^\dagger - \mathbf{x}^\star\|_{\mathbf{A}}$. Thus, if we want $\|\mathbf{x}^\dagger - \hat{\mathbf{x}}\|_{\mathbf{A}} = (1 + c)z^\star$ then we must find $\|\mathbf{x}^\star - \hat{\mathbf{x}}\|_{\mathbf{A}} = cz^\star$. The Iterative Hessian Sketch was designed so that $\|\mathbf{x}^\dagger - \hat{\mathbf{x}}\|_{\mathbf{A}}$ behaves as $O(z^\star)$ by substantially reducing the error $\|\mathbf{x}^\star - \hat{\mathbf{x}}\|_{\mathbf{A}}$ in comparison to the sketch-and-solve methods.

Algorithm 8 is an implementation of the Iterative Hessian Sketch (IHS). For simplicity, we have presented it in the unconstrained setting; if convex constraints $\mathcal{K}$ are necessary, then we must project $\mathbf{z}$ onto the convex constraint set ($\mathbf{z} \leftarrow \Pi_{\mathcal{K}}(\mathbf{z})$) before updating the iterates. Line 5 is presented as a linear solve which is a more efficient way of performing the Newton update $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \hat{\mathbf{H}}^{-1}\nabla f(\mathbf{x}^{(i)})$ to avoid computing the inverse explicitly. Based upon this equivalence, Line 6 is used to additively correct the iterates.

We will analyse the CountSketch in the Iterative Hessian Sketch (IHS) model. To define the variational quantities under iterative sketching, we need the following definition which is a set of 'residual' vectors over which the sketch acts.

Definition 5.3.1. *Let $\mathcal{K}$ be a closed convex constraint set and let $\mathbf{x}^\star$ be the solution to the least squares problem over $\mathcal{K}$. The* tangent cone *over $\mathcal{K}$ is*

$$K = \{\mathbf{v} \in \mathbb{R}^d : \mathbf{v} = t\mathbf{A}(\mathbf{x} - \mathbf{x}^\star), t \geq 0, \mathbf{x} \in \mathcal{K}\} \tag{5.18}$$

For an estimate $\hat{\mathbf{x}}$, the error vector $\hat{\mathbf{e}} = \mathbf{A}(\hat{\mathbf{x}} - \mathbf{x}^\star) \in K$. Let $\mathcal{S}^{n-1}$ be the set of n-dimensional vectors of unit Euclidean norm. After applying a suitable random projection $\mathbf{S}$, the quantities we must analyse are:

$$Z_1 = \inf_{\mathbf{v} \in K \cap \mathcal{S}^{n-1}} \|\mathbf{S}\mathbf{u}\|_2^2 \tag{5.19}$$

$$Z_2 = \sup_{\mathbf{v} \in K \cap \mathcal{S}^{n-1}} |\langle \mathbf{S}\mathbf{v}, \mathbf{S}\mathbf{u} \rangle - \langle \mathbf{v}, \mathbf{u} \rangle| \,. \tag{5.20}$$

In (5.20), $\mathbf{u}$ is an arbitrary unit norm vector. We will show that Z_1 and Z_2 are approximately preserved under the action of a CountSketch.

Lemma 5.3.1 (Subspace embedding implies IHS)**.** *Let $\mathbf{S}$ be a CountSketch matrix with $m = O(d^2/\varepsilon^2\delta)$ that is a $1 \pm \varepsilon$ subspace embedding for $\mathbf{A}$. Then $\mathbf{S}$ has $Z_1 \geq 1 - \varepsilon$ and $Z_2 \leq \varepsilon/2$.*

Proof. The subspace embedding property holds for all vectors in the column span of $\mathbf{A}$, in particular for unit vectors in $K \cap \mathcal{S}^{n-1} \subset \mathrm{col}\,(\mathbf{A})$. Therefore, $Z_1 \geq 1 - \varepsilon$ is an immediate consequence of $\mathbf{S}$ being a a $1 \pm \varepsilon$ subspace embedding for $\mathbf{A}$. On the other hand, if we use the standard relation

$$\langle \mathbf{u}, \mathbf{v} \rangle = \frac{1}{4}\left(\|\mathbf{u} + \mathbf{v}\|_2^2 - \|\mathbf{u} - \mathbf{v}\|_2^2 \right)$$

then it is straightforward to show that for *unit vectors* $\mathbf{u}, \mathbf{v} \in \mathrm{col}\,(\mathbf{A})$:

$$\begin{aligned}
\langle \mathbf{S}\mathbf{u}, \mathbf{S}\mathbf{v} \rangle &= \frac{1}{4}\left(\|\mathbf{S}(\mathbf{u} + \mathbf{v})\|_2^2 - \|\mathbf{S}(\mathbf{u} - \mathbf{v})\|_2^2 \right) \\
&\leq \frac{1}{4}\left((1+\varepsilon)\|\mathbf{u} + \mathbf{v}\|_2^2 - (1-\varepsilon)\|\mathbf{u} - \mathbf{v}\|_2^2 \right) \\
&= \langle \mathbf{u}, \mathbf{v} \rangle + \frac{\varepsilon}{4}\left(\|\mathbf{u} + \mathbf{v}\|_2^2 - \|\mathbf{u} - \mathbf{v}\|_2^2 \right) \\
&= \langle \mathbf{u}, \mathbf{v} \rangle + \varepsilon \langle \mathbf{u}, \mathbf{v} \rangle.
\end{aligned}$$

Hence, $\langle \mathbf{S}\mathbf{u}, \mathbf{S}\mathbf{v} \rangle \leq \langle \mathbf{u}, \mathbf{v} \rangle + \varepsilon$ by applying the Cauchy-Schwarz inequality on $\langle \mathbf{u}, \mathbf{v} \rangle$ as $\mathbf{u}, \mathbf{v}$ are unit vectors. The same argument applies for the lower bound which establishes establish $Z_2 \leq \varepsilon$. After rescaling this satisfies the claim. □

We need one final lemma to support our final theorem.

Lemma 5.3.2 ([PW16, Theorem 2])**.** *Let $\mathbf{x}^\star$ be the solution to a convex constrained least squares problem. Conditioned on $Z_1 \geq 1 - \varepsilon_{sk}$ and $Z_2 \leq \varepsilon_{sk}$ for every iteration $1 \leq i \leq T$, the solution $\hat{\mathbf{x}} = \mathbf{x}^{(T)}$ satisfies:*

$$\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_{\mathbf{A}} \leq \left(\prod_{i=1}^{T} \frac{Z_2}{Z_1} \right) \|\mathbf{x}^\star\|_{\mathbf{A}} \,.$$

Theorem 5.3.1. *Let $\mathbf{x}^\star$ be the solution to a least squares problem over convex constraint set $\mathcal{K}$. Let $\mathbf{S}$ be a $1 \pm \varepsilon_{sk}$ subspace embedding for the column space of*

$\mathbf{A}$ *for* $\varepsilon_{sk} < 1/2$. *Conditioned on* $\mathbf{S}$ *achieving a subspace embedding for every iteration* $1 \leq i \leq T$, *the IHS method with* CountSketch *returns an estimate with* $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_{\mathbf{A}} \leq \varepsilon_{sk}^T \|\mathbf{x}^\star\|_{\mathbf{A}}$. *The number of iterations is* $T = \Theta(\log 1/\varepsilon_\star)$ *for an error of* $\varepsilon_\star \|\mathbf{x}^\star\|_{\mathbf{A}}$ *and every iteration can be performed in time proprtional to* $O(nnz(\mathbf{A})) + \text{poly}(d/\varepsilon)$.

Proof. The error bound is an immediate consequence of Lemma 5.3.1 across every iteration. We can then invoke Lemma 5.3.2 and since $\varepsilon_{\mathsf{sk}} < 1/2$ we have

$$\begin{aligned}\frac{Z_2}{Z_1} &\leq \frac{\varepsilon}{2(1-\varepsilon)} \\ &\leq \varepsilon.\end{aligned}$$

Thus the $\varepsilon_\star = \varepsilon^T$ error is immediate. The running time is that needed to obtain an embedding, perform inner products and, and a direct solve at a cost of T_{solve}, hence we obtain $O(\mathsf{nnz}(\mathbf{A}) + T_{\mathsf{solve}}) \log(1/\varepsilon_\star))$. The worst-case solve time is $T_{\mathsf{solve}} = \text{poly}(d/\varepsilon)$ which varies depending on the constraint set $\mathcal{K}$. □

Comparison of Theorem 5.3.1 to [PW16]. The repeated need of a direct solver costing T_{solve} could appear unattractive. Importantly, this cost is over a smaller instance of size $m \times d$, not an $n \times d$ matrix. Hence, T_{solve} should be thought of as being small in comparison to T_{sketch}.

In [PW16], it is shown that the IHS algorithm with (sub)Gaussian sketches and the SRHT uses a projection dimension of $m < d$ which is proportional to the so-called Gaussian width $\mathcal{W}_{\mathcal{K}}$ of the constraint set $\mathcal{K}$. This Gaussian width measures the complexity of the problem and is proportional to d when the problem is unconstrained. More structured problems can have much smaller Gaussian width. For example, sparse least squares has $\mathcal{W}_{\mathcal{K}}$ grow logarithmically in d [PW16]. The work of [BDN15] shows that the SJLT also preserves norms on convex constraint sets in $m = \tilde{O}(\mathcal{W}_{\mathcal{K}}^2)$ projections modulo some poly (log) factors. However, the same property is *not* known for the CountSketch.

More precisely, when using the optimal sketches in IHS, it is proven that we may sample only $m = \tilde{O}(\mathcal{W}_{\mathcal{K}}^2)$ projections which can be much smaller or independent of d. Hence, a quadratic program solver could solve the dual program over a $m \times d$ instance, which in the worst case may be cubic in m, not d. Thus, the major polynomial time cost in T_{solve} is the poly (m) contribution, which is significantly less than $O(d^3)$ for a linear solve. However, as the worst-case guarantee of the CountSketch subspace embedding is that $O(d^2)$ projections are used at every round. Then the solve step could need time $O(d^4)$ to generate $\mathbf{A}^\top \mathbf{S}^\top \mathbf{S}\mathbf{A}$. Our contribution shows that the iterates can be performed quickly but we do not have any guarantees for an improved embedding dimension.

It would be useful to have a better understanding of how well the CountSketch estimates norms of vectors on a convex constraint set $\mathcal{K} \subsetneq \mathbb{R}^d$. Although [BDN15, LLW21] show that under certain assumptions, the CountSketch can have the optimal projection dimension of $m = \tilde{O}(d \log(d/\delta)/\varepsilon^2)$, they give no theoretical analysis of IHS when fewer than d rows are used in the sketch. Showing that CountSketch preserved vector norms on the constraint set $\mathcal{K}$ using fewer than d directions would go some way to explaining the empirical performance observed when using CountSketch for LASSO [CD19] and other convex constrained programs [LLW21].

Term	Symbol	Notes
Model weights	$\mathbf{x}^\dagger$	Idealised weights for theoretical model. Not observed in practice.
Optimal weights	$\mathbf{x}^\star$	The weights that minimise the training loss, found by an idealised solver.
Estimate weights	$\hat{\mathbf{x}}$	Any weights found when using a sketch to approximate $\mathbf{x}^\star$.

Table 5.2: Recap of the different weights used for measuring errors in regression.

5.4 Experiments

A recap of the different weights that we study is given in Table 5.2. Recall that the iterative method is motivated by the weakness of the sketch-and-solve weights $\hat{\mathbf{x}}$ in comparison to the optimal weights $\mathbf{x}^\star$. This gives rise to two notions of error that we can test empirically:

1. *Model coefficient error*: $\left\|\hat{\mathbf{x}} - \mathbf{x}^\dagger\right\|_{\mathbf{A}}$ measures how well the sketched weights $\hat{\mathbf{x}}$ describe the process $\mathbf{y} = \mathbf{A}\mathbf{x}^\dagger + \boldsymbol{\omega}$. If data were continually generated according to this process and this norm is small, then we would expect $\hat{\mathbf{x}}$ to continue to be a good proxy for $\mathbf{x}^\dagger$.

2. *Optimal coefficient error*: $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_{\mathbf{A}}$ measures the distance from estimate weights $\hat{\mathbf{x}}$ to the optimal weights $\mathbf{x}^\star$ which minimise the training loss.

We can only test the former on synthetic data, while the latter can be tested on both real and synthetic data. Our experiments take place in the multi-round optimisation model (Definition 2.1.5) with one party holding the weights. After every iteration, the weights are updated. The sketch-and-solve experiments can be thought of as completing only one round in this model.

Remark 5.4.1. *We implement the SVD in most of our experiments for comparison. This is idealised and in large-scale settings would not be practical. Firstly, the time is $O(nd^2)$ to obtain either the SVD or $\mathbf{A}^\top\mathbf{A}$ before a solver is*

called. Distributed or multipass map-reduce implementations for the SVD for matrices with $n \gg d$ often combine 'local' copies of the covariance matrix $\mathbf{A}^\top\mathbf{A}$ based upon the data that a party holds [BGD13, Sch20].[4] *These implementations might be more suitable for a comparison with IHS in the multi-round optimisation model. We leave a rigorous comparison to more sophisticated implementations of SVD for future work.*

In light of Remark 5.4.1, we often find that the IHS is slower than an exact SVD of an $n \times d$ matrix in our implementation. There are a few reasons for this: firstly, a call to SVD in `NumPy` calls C++ code. If the data is already loaded into memory, this can be fast but is not representative of a large-scale comparison. Secondly, the iterates of IHS from Algorithm 8 are implemented by solving a linear system through an SVD of an $m \times d$ matrix. Thus, we have a call to SVD for every iteration which aggregates. However, the SVD in Algorithm 8 is over small $m \times d$ matrices, so for larger instances, this would be small in comparison to an SVD over the entire $n \times d$ dataset. Also, we are interested in time comparisons *between* the sketches and the models, rather than optimising solely to beat the SVD time.

This disparity is not problematic for our comparison as an SVD would be difficult to compute even if one were allowed many passes over the data whereas the IHS only needs a pass for (i) sketching and (ii) inner products. Another implementation detail that would improve the time performance of sparse sketches in IHS is to use sparse *inner product* methods. We only exploit sparsity for the sketching step, whereas it would also cause a speedup in the gradient update. Hence, we claim that the IHS would be scalable even in settings when the SVD would not be and defer exploring exactly when this occurs until future work.

5.4.1 Synthetic Experiments

We provide a baseline comparison using the synthetic examples as presented in [PW16] on Ordinary Least Squares (OLS) problems. The aim here is to verify whether the CountSketch and Sparse Johnson-Lindenstrauss Transform (SJLT) perform comparably to the dense sketches (Gaussian and Subsampled Randomised Hadamard Transform (SRHT)) that were proposed in the original Iterative Hessian Sketch (IHS) framework. The central questions that we address are:

1. The CountSketch may need $O(d)$ more rows to obtain a subspace embedding compared to all other sketches we consider. Does this result in substantially weaker error performance on some simple test cases?

[4] This idea can be compared with Algorithm 3 except the (numerically) exact SVD is returned, not an approximate ℓ_p subspace embedding.

2. If we fix a projection dimension, say $m = \gamma d$, then one would expect the CountSketch to obtain weaker approximation than the other sketches we consider. Consequently, we may expect less progress to be made per iteration using the CountSketch than using other methods. Do we pay for this by needing substantially many more iterations to converge?

All of the synthetic experiments use a data matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ which is chosen with $\mathbf{A}_{ij} \sim \mathcal{N}(0,1)$. A ground truth vector describing the model $\mathbf{x}^\dagger \in \mathbb{R}^d$ is chosen with every $\mathbf{x}_i^\dagger \sim \mathcal{N}(0,1)$ and then $\mathbf{x}^\dagger$ is normalised to unit length. Finally, random Gaussian noise $\boldsymbol{\omega}_i \sim \mathcal{N}(0,1)$ is added to generate the target vector $\mathbf{y} = \mathbf{A}\mathbf{x}^\dagger + \boldsymbol{\omega}$. The three different sets of weights that we study are given in Table 5.2.

Model error compared to data size (Figure 5.1).

The first experiment seeks to understand how the error responds as n increases. This is motivated by Proposition 5.1.1 which shows the optimal estimator for OLS:

$$\left\|\mathbf{x}^\dagger - \mathbf{x}^\star\right\|_{\mathbf{A}}^2 = \frac{\sigma^2 d}{n}. \tag{5.21}$$

For sketch-and-solve, Corollary 5.2.1 shows that $\|\hat{\mathbf{x}} - \mathbf{x}^\dagger\|_{\mathbf{A}} \geq \sigma^2 dc/m$. In fact, for the problem setup we will use described below, we expect $\|\hat{\mathbf{x}} - \mathbf{x}^\dagger\|_{\mathbf{A}} \geq c'/7$. This states that the error between the model weights and the sketch-and-solve weights is lower bounded by a constant. For IHS we can make the model error much smaller. By the triangle inequality and Theorem 5.3.1 we should expect the IHS method with CountSketch after T iterations to achieve:

$$\left\|\mathbf{x}^\dagger - \hat{\mathbf{x}}\right\|_{\mathbf{A}} \leq \varepsilon_{\mathsf{sk}}^T \left\|\mathbf{x}^\star\right\|_{\mathbf{A}} + \frac{\sigma^2 d}{n}. \tag{5.22}$$

The optimisation error term $\varepsilon_{\mathsf{sk}}^T \left\|\mathbf{x}^\star\right\|_{\mathbf{A}}$ approaches zero when n increases. Also, there is always a problem-specific residual $\left\|\mathbf{x}^\dagger - \mathbf{x}^\star\right\|_{\mathbf{A}}$ that is not recoverable even in the case of using optimal weights $\mathbf{x}^\star$. This residual has diminishing effect as n increases as per (5.21).

Data $\mathbf{A} \in \mathbb{R}^{n \times d}$ is generated for a fixed value of $d = 10$ and

$$n \in \{100 \times 2^i \text{ for } i \in \{1, 2, \ldots, 10\}\}.$$

The number of iterations is $T = 1 + \log(n)$ and the sketch size for all projections in the IHS framework is $m = 7d$. Ten independent random trials were performed with the mean being reported. We instantiate only the CountSketch in the IHS to avoid cluttering the plots. For the sketch-and-solve method, we tested the Gaussian, SRHT, SJLT, and CountSketch. The sketch-and-solve methods are instantiated using $m' = Tm$ projections which is the sum of sketch sizes

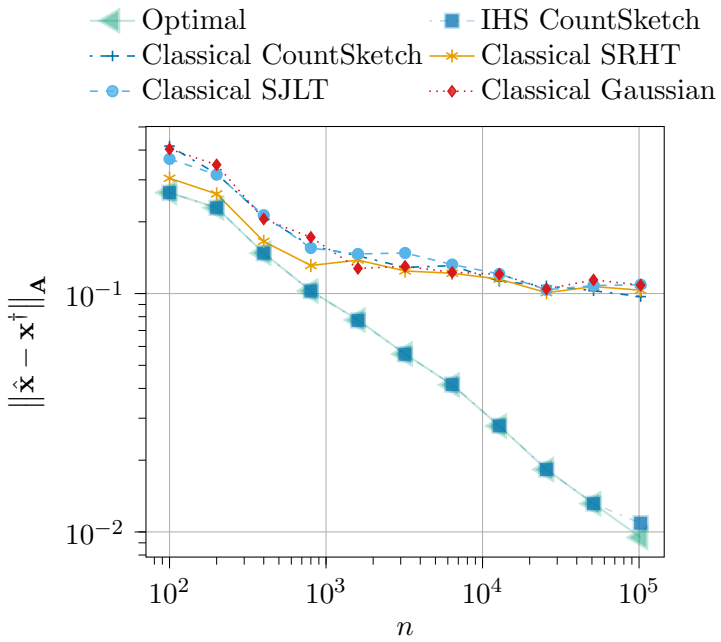


Figure 5.1: Error to model weights $\mathbf{x}^\dagger$ under an optimal solver, classical sketch, or IHS.

used in the IHS framework. We will investigate these sketches in the IHS framework in subsequent experiments. Additionally, we obtain an "idealised estimate" found by performing no sketching and returning the OLS solution. This is equivalent to one round of IHS with $\mathbf{S} = \mathbf{I}_d$ which is simply one step of the exact Newton method (see Section 2.3.2 and table 2.2). Hence, it is sufficient to obtain an SVD of $\mathbf{A}$ and return $\hat{\mathbf{x}} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^\top\mathbf{y}$. Our measure of performance is the *error to model weights* $\mathbf{x}^\dagger$ under the semi-norm $\left\|\mathbf{x}^\dagger - \hat{\mathbf{x}}\right\|_{\mathbf{A}}$ for estimate weights $\hat{\mathbf{x}}$, returned under sketch-and-solve or IHS methods.

Consistent with the theory, the sketch-and-solve model with any sketch is significantly weaker than the optimal weights, and the IHS method with CountSketch. Although the CountSketch yields a weaker embedding guarantee than the other random projections, there is little noticeable change in the error performance in the sketch-and-solve model and its average error performance appears to plateau out at roughly the same constant of 0.1 as when other sketches are used. The SRHT appears preferable for smaller instances, but these differences become less noticeable as n increases. All methods show mild descent, but this begins to flatten at an error that is suboptimal compared to the reciprocal decay of the optimal weights.

In contrast, using CountSketch in the IHS model sees the error decay reciprocally in n (as per (5.22)) and tracks the behaviour of the optimal estimator almost indistinguishably. The number of iterations increases only very slowly with n so the cause of this can be attributed to refining a small

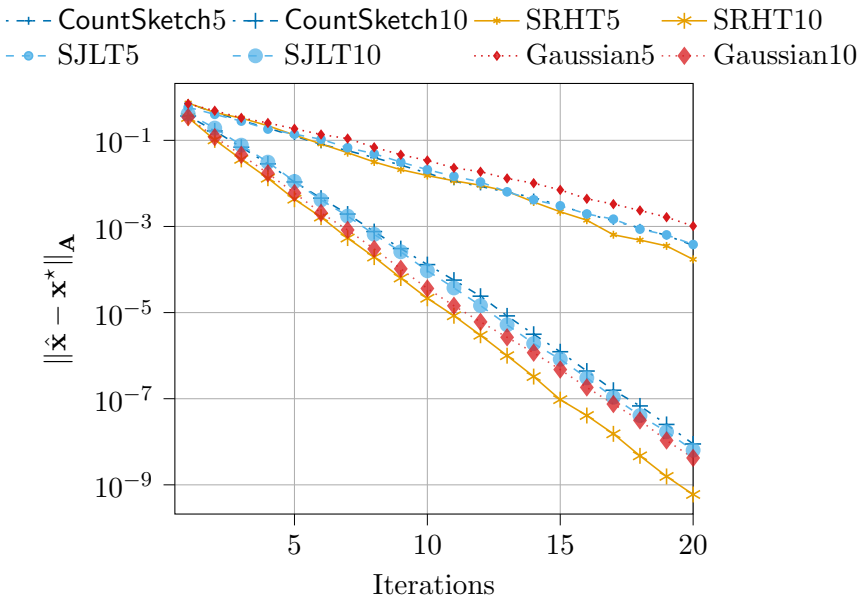


Figure 5.2: IHS optimal coefficient error between the sketched weights $\hat{\mathbf{x}}$ and the optimal weights $\mathbf{x}^\star$ over iterations.

number of rough estimates that move closer towards $\mathbf{x}^\star$ which can take better advantage of the increased evidence made available from the model. Overall, this means that the estimate returned after the IHS procedure will be much more representative for prediction than simply using the sketch-and-solve estimators. We conclude that if one operates in the multi-round optimisation model (Definition 2.1.5) then taking more randomised steps with a small sketch is preferable to a one-shot algorithm with only one large sketch.

Optimal Coefficient Error vs Number of Iterations (Figures 5.2 and 5.3)

We next study the convergence properties of IHS by varying the sketch size and the number of iterations completed. The test instance is again OLS regression with data generated as before but with $n = 6000$, $d = 200$ and sketch sizes $m = \gamma d$ for $\gamma = 5, 10$. The IHS method using a Gaussian, SRHT, SJLT, CountSketch was run for $T = 20$ iterations. We measure

(i) the coefficient error to optimal weights $\mathbf{x}^\star$, evaluated as $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_{\mathbf{A}}$ in Figure 5.2 and

(ii) model error $\left\|\hat{\mathbf{x}} - \mathbf{x}^\dagger\right\|_{\mathbf{A}}$ in Figure 5.3.

Results are averaged over ten trials.

In Figure 5.2, all methods converge towards the optimal solution. Convergence is faster when the sketch size is larger as the sketch parameters Z_1 (5.19) and Z_2 (5.20) are more accurate. At both $m = 5d, 10d$, the SRHT appears

to be the best sketch as its error can be slightly lower than the competing methods due to a larger per-step error reduction. At the larger sketch size the SRHT can be between one half or one full order of magnitude more accurate than other sketches after a given number of steps. For the smallest sketch sizes, the Gaussian sketch seems least competitive, however, for the larger sketch of size $m = 10d$, the worst-performing sketch is the CountSketch. This is interesting as it suggests that for a higher dimensionality problem, we begin to see slight degradation in the quality of the estimates returned from IHS with CountSketch, yet this difference is very small in *absolute* terms. The performance of the SJLT is comparable to other methods.

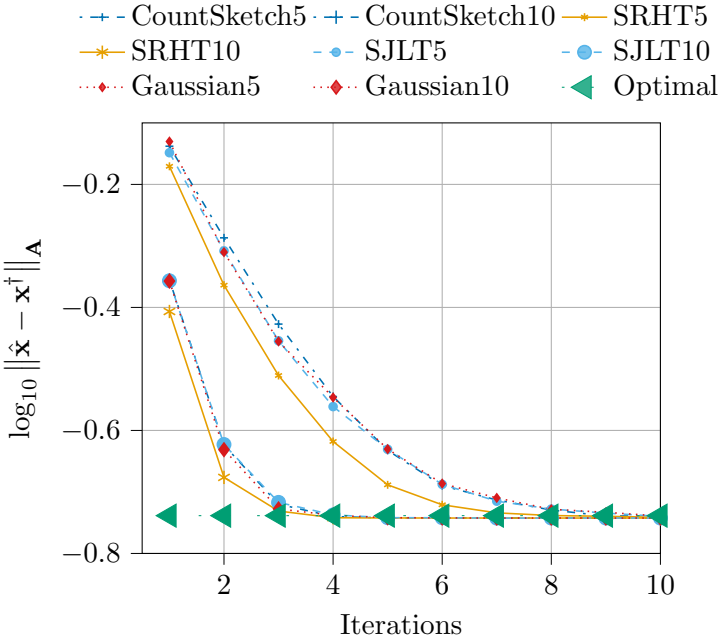


Figure 5.3: IHS error to model weights vs iterations. The model error from the optimal weights given data $\mathbf{A}$ is denoted: Optimal $\approx \|\mathbf{x}^\star - \mathbf{x}^\dagger\|_{\mathbf{A}}$.

Next, we return again to the model error, illustrated in Figure 5.3. The horizontal line plotted for the optimal weights $\mathbf{x}^\star$ is determined by the problem dimensionality. Recall that $\|\mathbf{x}^\star - \mathbf{x}^\dagger\|_{\mathbf{A}} \approx \sqrt{\sigma^2 d/n}$ which in this setup is $\sqrt{\sigma^2 d/n} = \sqrt{2/60}$. Then we plot $\log_{10}\left(\|\mathbf{x}^\star - \mathbf{x}^\dagger\|_{\mathbf{A}}\right)$ which roughly has an optimum value of $\log_{10}\sqrt{2/60} \approx -0.73$.

All sketch methods have error converging to the optimum and this is achieved more quickly for larger sketch sizes. This behaviour should be contrasted with Figure 5.1 in which the sketch-and-solve does not decay to $\mathbf{x}^\dagger$. Figure 5.2 shows the distance between estimates $\hat{\mathbf{x}}$ and optimal weights $\mathbf{x}^\star$ becomes vanishingly small as the number of iterations increases. However, we see in Figure 5.3 that after only a small number of iterations all methods descend towards the *statistically* optimal model error of $\|\mathbf{x}^\star - \mathbf{x}^\dagger\|_{\mathbf{A}}$, irrespective of the

sketch method chosen. We find that the SRHT again seems the best method as it decays to a statistically optimal result in mildly fewer iterations than all other random projections at both $m = 5d$ and $m = 10d$. For smaller sketches, we do see slightly worse performance for the CountSketch but this difference becomes negligible once 2-3 more iterations are completed. Indeed, after 6 iterations, the average performance of CountSketch at $m = 5d$ becomes comparable to the SJLT and Gaussian sketch. Nonetheless, the difference to Figure 5.1 is stark; if extra steps can be made, then they should be taken. Moreoever, the number of extra steps (or passes over the data/rounds of communication) necessary to obtain weights that represent the model decreases as the sketch size is increased. Thus, in practice, a tradeoff should be made between the sketch size and the acceptable number of gradient steps completed.

5.4.2 Real Dataset and Error-Time Performance

Taken in the round, Figure 5.1, Figures 5.2 and 5.3 suggest that in this example, the CountSketch outperforms its worst-case guarantees. Through the iterative scheme, the CountSketch is competitive with more theoretically robust sparse and dense embeddings. Nonetheless, the main win from using the sparse embeddings, CountSketch and SJLT, is that we can run the IHS in time proportional to the number of nonzeros in the data; we will explore this subsequently on a real dataset.

We take the California Housing [PVG+11] dataset and, for reproducibility, followed the open-source implementation for data preprocessing [Gér] which included some light data cleaning and feature engineering. Finally, we used a random train-test split into $\mathbf{A}_{\mathsf{train}}, \mathbf{y}_{\mathsf{train}}$ and $\mathbf{A}_{\mathsf{test}}, \mathbf{y}_{\mathsf{test}}$ of size $n_{\mathsf{train}} = 16512, n_{\mathsf{test}} = 4128$ and $d = 16$. We refer to this as the "raw" dataset which is 75% dense, that is $\mathsf{nnz}\left(\mathbf{A}_{\mathsf{train}}\right)/dn_{\mathsf{train}} = 0.75$.

Experimental Setup

We train an OLS regression model on $\mathbf{A}_{\mathsf{train}}, \mathbf{y}_{\mathsf{train}}$ using the a classical sketch-and-solve with CountSketch and the IHS model with all sketch methods CountSketch, SJLT, SRHT, and Gaussian. Ten trials were performed each using an independent random permutation of the training data. The optimal coefficient error $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_{\mathbf{A}}$ between estimate weights $\hat{\mathbf{x}}$ and optimal weights $\mathbf{x}^\star$ is measured. The mean performance over all trials is reported. For IHS, the projection dimension is chosen as $m = 5d$ to illustrate the tradeoffs of performance, it is not necessarily chosen to be an optimal sketch size. The classical sketch-and-solve method is employed by sampling $m' = Tm$ projections for a CountSketch. In addition, the SJLT is initialised with 10 nonzeros per column.

The optimal OLS weights $\mathbf{x}^\star$ for regression are obtained via the SVD,

acknowledging the caveats made in Remark 5.4.1. We additionally measure the wall-clock time taken to execute the 10 iterations and will record error against both number of iterations and the wall-clock time. First, we will investigate the "raw" training data, followed by various levels of "sparsified" data by randomly zeroing out entries to test whether the sparse transforms offer a noticeable improvement over dense sketches in wall-clock time.

Experiment 1: Raw data, error compared to iterations and time (Figure 5.4).

In this experiment we find that there is little discernible difference when we compare error as a function of iterations for each of the different sketch methods in the IHS model. Thus, we compare how the error decreases as a function of time which is evaluated as $\log_2$ of time in seconds to execute the 10 iterations of IHS. Base 2 logarithms are chosen solely for ease of comparison with some later plots in Figure 5.5 to better illustrate the differences.

In Figure 5.4 we see that sketch-and-solve is faster than an exact SVD solver (the marker is to the left of the SVD time on the wall-clock time plot) yet the solution error is poor, roughly 10^{-2}. It is fast but is not a good estimate. On the other hand, all of the sketch methods used in the IHS model descend to an estimate roughly 4 orders of magnitude more accurate than the classical sketch-and-solve weights after ten iterations. There is negligible difference between the random projections on the error-vs-iterations profile that becomes clear when we look at error against wall-clock time.

We see clear separation between the sketching methods in IHS in wall-clock time. The Gaussian sketch is slowest, completing 10 iterations in 1 second on average. In line with the theory, we see that the SRHT is faster than the Gaussian to complete the iterations, needing a little over 2^{-3} seconds to terminate. This includes time necessary to pad $\mathbf{A}_{\mathsf{train}}$ with zeros so that it is viable size for the Hadamard Transform which we found had little practical impact but could be non-negligible for arbitrary n [AMT10]. Interestingly, the sparse sketches are faster than the SRHT even though this data is relatively dense, having roughly 75% nonzeros. The SJLT is almost a factor of 2 faster than the SRHT to complete and similarly, the CountSketch is a factor of 2 faster than the SJLT. Note that this is slightly better than the theory as we would expect the SJLT to use $s = 10$ times longer to sketch as there are more nonzeros per column. It is worth cautioning that using more nonzeros (increasing s) for SJLT would be a concern for larger-scale implementations as generating the random variables or hash functions can be an expensive overhead. Moving forward, it will be worth noting that in this case, with density of 75%, the CountSketch performed 2 full steps of IHS in at most the time of an SVD call;

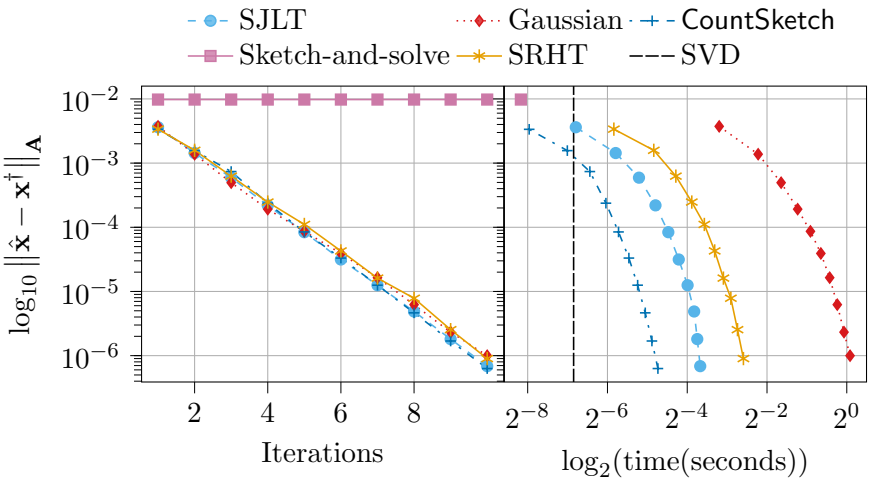


Figure 5.4: Error profile for IHS on California Housing dataset

when we alter the sparsity of the data, this will be a useful litmus test to establish the speed of the algorithm. Simply put, in about 2^{-5} seconds, the CountSketch achieves error less than 10^{-6} meanwhile the SJLT is the nearest competitor, being over two orders of magnitude worse in the same time.

Experiment 2: Sparsified data, error compared to time (Figure 5.5).

Next we consider the same experimental setup as above but we will randomly sparsify the data. The aim of this is to check that speed of the IHS with sparse embeddings remains superior to the dense sketches as the number of nonzeros in the data decreases. We will take $\mathbf{A}_{\mathsf{train}}$ and randomly set some of the $\mathbf{A}_{ij} = 0$ to achieve data densities of $12.5\%, 25\%, 50\%$. Below 12.5% density, we found that the fidelity of the OLS solution is compromised so only study data in this range. Again we repeat the experiment for 10 iterations and 10 independent trials. Each trial uses a fresh sparsification of the data.

The results are presented in Figure 5.5. Note that the curves for SRHT and Gaussian are unaltered at all densities. This is to be expected as these methods should not have their time dependent on the density of data. For sparse sketches on the other hand, there is a gentle time increase with the density of the data as expected from the theory; albeit the increase is less than a factor of 2 overall when the density increases from 12.5% to 50%. This is best viewed by scanning Figure 5.5 from top (density = 12.5%) to bottom (density = 50%). Note that the curves are translated mildly to the right (increased time) for denser data. As the data density increases, we see that fewer iterations can be completed in time less than the exact SVD solver. This is shown by the number of markers plotted on the curve prior to the SVD line: 5 for 12.5%, 4 for 25%, 3 for 50% when using the CountSketch.

By inspecting the time to execute 10 steps when no change is made to the

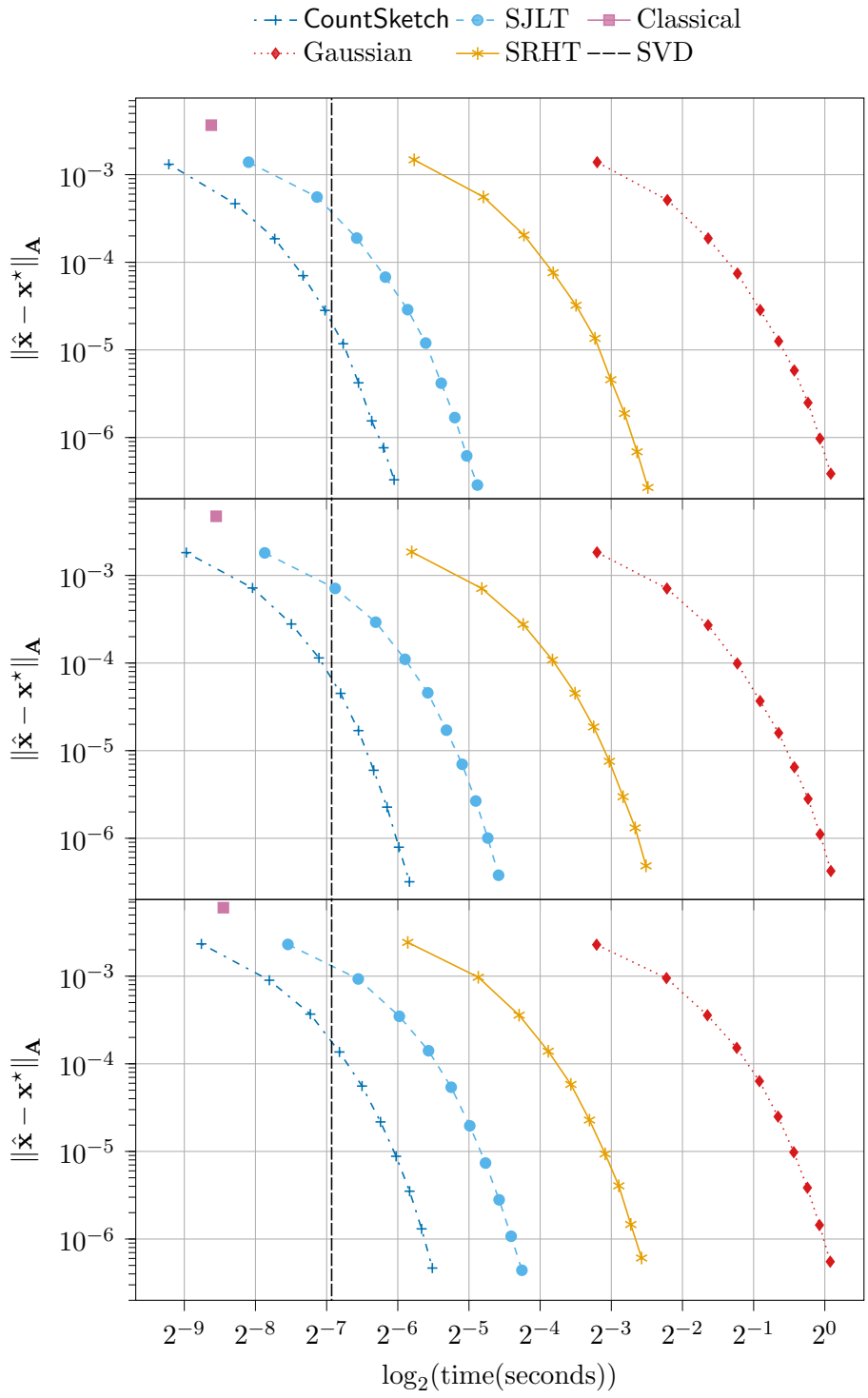


Figure 5.5: Error profile for IHS on California Housing dataset for density 12.5% (top), 25% (middle), 50% (bottom).

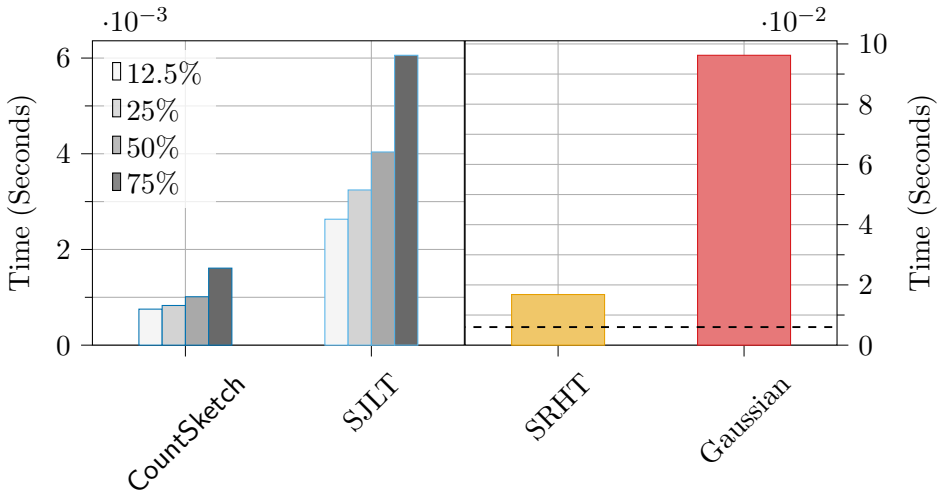


Figure 5.6: Mean sketch time on different density data for sparse sketches (left) and mean sketch time over all 4 densities for dense sketches (right). Note the different y axis scale in each pane. The black dashed line in the right-hand pane is added for comparison. It plots the *largest* average sparse sketch time, namely SJLT on density of 75%.

density (Figure 5.4) the CountSketch is roughly a factor of $2^{-2.5}/2^{-5} = 2^{2.5}$ (approximately a factor of 10) faster than the "optimal" dense method SRHT to complete 10 iterations. However, when we sparsify the data, this speedup can be increased to roughly $2^{-2.5}/2^{-6} = 2^{3.5}$, as shown in the bottom pane of Figure 5.5. The SJLT can be seen as a compromise between the "suboptimal" CountSketch and the optimal dense sketches in that it permits a time improvement over the dense sketches whilst maintaining sparsity in the sketch. It also exhibits the same properties as the CountSketch mentioned above so is also a sparse alternative to the SRHT.

Experiment 2a: Explaining the improved time profile Figure 5.6.

The improved error-time profile of the sparse sketches can be attributed to the much reduced T_{sketch} in comparison to dense sketches. In Figure 5.6 we have plotted the mean sketch time of all of the sketches over all sparsfied versions of the dataset. For clarity, the sparse sketches are plotted in the left hand pane, separate to the dense sketches which are roughly order of magnitude slower and plotted in the right hand pane. This plot clearly shows that the sparse sketches have their runtime increase proportionally with the density of the data as the four bars have their height increasing linearly. Interestingly, there appears to be a sharper increase between $50\% - 75\%$ density. As the dense sketches SRHT and Gaussian do not have their runtime dependent on the density of the data, we simply report the mean time over all 40 sketches (e.g. 4 datasets and 10 iterations for each IHS).

By evaluating

$$\text{xSpeedup} = \frac{\text{slow time}}{\text{fast time}}$$

we can easily see that the CountSketch is at least a factor 10 faster than SRHT in the "raw" data setting of 75% density (approximately $2 \times 10^{-2}/2 \times 10^{-3}$), but when the data is substantially sparser, this can be closer to a factor 20 speedup (approximately $2 \times 10^{-2}/1 \times 10^{-3}$). Similarly, the SJLT is between a factor 3 and 8 faster than the SRHT. In summary, both sparse sketches are at least an order of magnitude faster than the Gaussian sketch, with the CountSketch being at least an order of magnitude faster than the SRHT. We observe an expected constant factor scaling between the CountSketch and the SJLT.

These results are to be compared with Figures 5.4 and 5.5 in which at 12.5% density we observed a wall clock time speedup of roughly $2^{3.5} \approx 11$. The discrepancy in the sketch time speedup factor compared to wall clock time speedup is because the iterative updates coincidentally take time approximately equal to that of a single CountSketch. Thus, the wall clock time for sparse sketches is quite sensitive to the inner product update time, but this is essentially negligible in comparison to the cost of 10 dense sketches. It is possible that this could be alleviated by a gradient update that better exploits data sparsity.

Concretely, the total sketching time when using an SRHT is roughly $10 \times 1.75 \times 10^{-2}$ secs. $= 0.175$ secs. meanwhile the total time to complete all ten iterations of IHS roughly $0.1768 \approx 2^{-2.5}$ seconds; the absolute difference of these two numbers is 0.0018 secs. to perform all ten updates, about 1% of the total wall clock time. On the other hand, sketching the sparsest data 10 times with a CountSketch takes roughly 7.5×10^{-3} secs. overall. However, the time to complete all ten iterations of the IHS in this case is $2^{-6} = 0.015$ secs., i.e. roughly twice the cost of the overall time to sketch. This is illustrated by the shortest bar in the leftmost pane of Figure 5.7. Overall, we see that updates account for roughly half of the wall clock time for IHS with CountSketch versus about 1% of the wall clock time for IHS with SRHT. This relationship is illustrated in Figure 5.7 where we see that the updates can take a significant fraction of the total time for sparse sketches, for instance almost half of the total time at a density of 12.5% when using CountSketch, but are negligible compared to the cost of all dense sketches.

Experiment 3: Sparsified data, test error compared to time (Equation (5.23)).

We additionally present some supplementary results regarding the test error of the IHS model with different sketches. Although we do not have theory to support this experiment, it is useful to consider how one might deploy such an

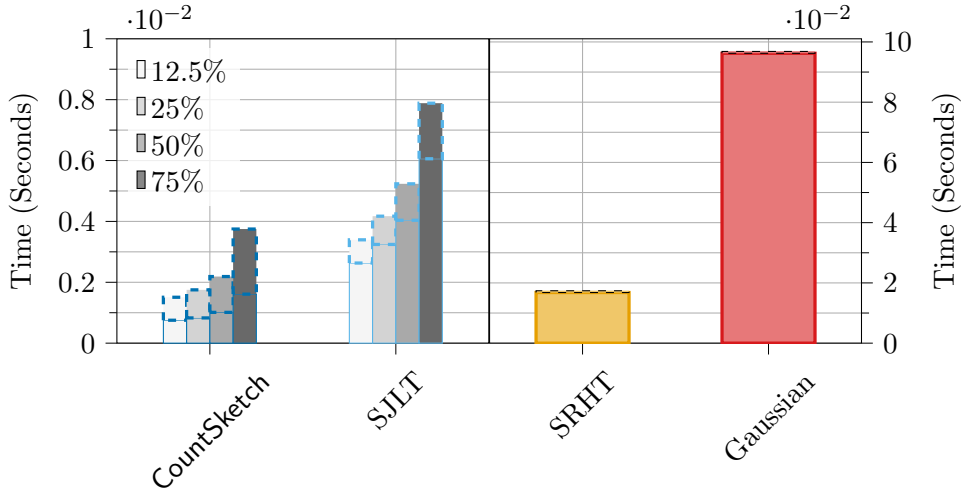


Figure 5.7: Mean total sketch time (thin lines) and total update/gradient step time (thick dashed lines) marked on different density data for sparse sketches (left) and mean sketch time over all 4 densities for dense sketches (right). Note the different y axis scale in each pane.

iterative method in comparison to sketch-and-solve. We measure the *test error approximation ratio*

$$\text{Test error approximation ratio} = \frac{\|\mathbf{A}_{\text{test}}\hat{\mathbf{x}} - \mathbf{y}_{\text{test}}\|_2}{\|\mathbf{A}_{\text{test}}\mathbf{x}^\star - \mathbf{y}_{\text{test}}\|_2} \tag{5.23}$$

which is the ratio of testing errors when using the optimal weights $\mathbf{x}^\star$ or the estimated weights $\hat{\mathbf{x}}$. When $\hat{\mathbf{x}}$ better approximates $\mathbf{x}^\star$ the approximation ratio approaches 1. Indeed, the classical sketch-and-solve method with CountSketch is fast, but has test error off by a factor of 2, meanwhile when using the IHS, we see that, in general, test performance is improved as more iterations are completed. Our approach of using the CountSketch in IHS is vindicated as it obtains better test performance against time, consistent with Figures 5.4 and 5.5.

5.5 Conclusion

We have investigated sparse sketches for scalable regression when the aim is to provide high-accuracy guarantees on the estimated weights $\hat{\mathbf{x}}$ compared to the optimal weights $\mathbf{x}^\star$. This was motivated by a weakness of the sketch-and-solve method for prediction after the regression model is fit, as evidenced for the CountSketch in Section 5.2. When comparing the Iterative Hessian Sketch (IHS) method with all random projections, we saw little difference in terms of error when using the optimal sketches (Gaussian,SRHT,SJLT) which suggested that on the examples tested, the CountSketch outperformed its worst-case

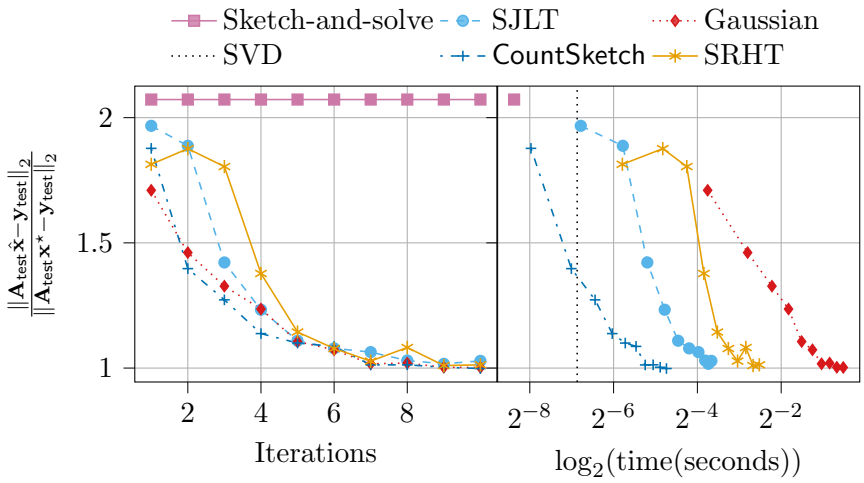


Figure 5.8: Test error approximation factor when using IHS

guarantees. When we studied the error performance over time, we saw that the CountSketch yielded comparably accurate estimates to the optimal sketches in a fraction of the time. Future investigation should study whether any guarantees can be given using a projection dimension dependent on the Gaussian width of the constraint set, rather than needing a full-subspace embedding.

Chapter 6

Sketched Ridge Regression

The results of Chapter 5 are flexible and apply to arbitrary convex constrained least squares problems. In this chapter we change direction and focus solely on the widely-used variant of least squares known as Ridge Regression (RR). Iterative Hessian Sketch (IHS) can be applied to ridge regression as it is a regularised version of unconstrained regression. In contrast to the prior chapter which focused on random projections, we have the following differences:

1. The sketching technique we study is the *deterministic* Frequent Directions (FD) algorithm and thus our iterative scheme is entirely deterministic.

2. This algorithm operates only in the row-arrival version of the multi-round optimisation model as defined in Definition 2.1.5.

3. Motivated by low-rank representations of large-scale data, the Frequent Directions (FD) summary keeps fewer than d directions in contrast to the randomised methods which keep at least d projections. FD has a different guarantee than the subspace embeddings from Section 5.1.1, yet we are still able to prove convergence.

4. We use a single sketch for all iterations rather than a new sketch for every iteration. Hence, once the sketch is found, a pass over the data is only needed for inner product queries, unlike the previous chapter which needs to sketch and take inner products for every iteration.

These changes are motivated by the previous chapter which found that obtaining the sketches was the most time-consuming part of the IHS. Thus, rather than finding a random sketch that is fast and can directly slot in for slower sketches, we will find a sketch that is more accurate but takes longer to obtain. Our results will show that investing more time in the sketching part of the algorithm is beneficial as we can use a single sketch for all iterates.[1]

[1]Single *random* sketch approaches are discussed in Section 6.1.1: Multi-Round Optimisation Model.

Chapter Outline

The main contribution of this chapter is a deterministic approximate Newton method for ridge regression. We use FD as a preconditioner so the working space is $O(md)$ space for $m < d$. These iterations are highly efficient, taking only linear time rather than polynomial which results in a highly scalable (nearly) linear time algorithm to achieve convergence. Our method is simple to analyse and implement. We follow the outline given below.

- **Section 6.1** formalises the problem and highlights the similarities and differences from the previous chapter.
- **Section 6.2** introduces the key tools that we need to establish our theory.
- **Sections 6.3 and 6.4** provide proofs of correctness for our algorithm. This is followed by time and space complexity analysis in **section 6.5**.
- **Sections 6.6 and 6.7** presents an empirical investigation comparing our approach to the methods established in the previous chapter.

6.1 Introduction

Our focus is Ridge Regression (RR) which has become a key tool in data analysis. RR is as expensive as Ordinary Least Squares (OLS) to solve at large-scale and in high dimensions. Ridge regression is most useful in the case when d is relatively large as the regularisation term suppresses the magnitude of the returned weights along some of the d components. This is beneficial as if $\mathbf{A}$ has many small singular directions, then $\mathbf{A}^\top\mathbf{A}$ could be close to singular. Such behaviour can cause numerical instability in evaluating $\left(\mathbf{A}^\top\mathbf{A}\right)^{-1}$ which is the key quantity for solving OLS as some of the components in the solution weights can be arbitrarily inflated. Another interpretation of ridge regression is to bias the solution weights in order to decrease the variance of predictions on unseen test data.

Recall that for input data $\mathbf{A} \in \mathbb{R}^{n\times d}$ and $\gamma > 0$, ridge regression asks us to find the weights:

$$\begin{aligned} \mathbf{x}^\star &= \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) \\ f(\mathbf{x}) &= \frac{1}{2}\left\|\mathbf{A}\mathbf{x}-\mathbf{y}\right\|_2^2 + \frac{\gamma}{2}\left\|\mathbf{x}\right\|_2^2. \end{aligned} \tag{6.1}$$

Solving (6.1) when $n > d$ by the SVD (or other related decompositions) requires $O(nd^2)$ time and $O(d^2)$ space to evaluate

$$\mathbf{x}^\star = \left(\mathbf{A}^\top\mathbf{A} + \gamma\mathbf{I}_d\right)^{-1}\mathbf{A}^\top\mathbf{y}. \tag{6.2}$$

We also have:

$$\nabla f(\mathbf{x}) = \left(\mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d\right) \mathbf{x} - \mathbf{A}^\top \mathbf{y} \tag{6.3}$$

$$\nabla^2 f(\mathbf{x}) = \mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d. \tag{6.4}$$

Because $\nabla^2 f(\mathbf{x})$ is constant with respect to $\mathbf{x}$ we will henceforth take $\mathbf{H} = \mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d$ so that computing $\nabla^2 f(\mathbf{x})$ for any $\mathbf{x}$ amounts to only needing $\mathbf{H}$. Again, recall that we need not materialise $\mathbf{H}$, merely to understand its action on vectors from $\mathbb{R}^d$. We will assume that $n > d$ and the input data has rank $(\mathbf{A}) = d$ so that $\mathbf{x}^\star$ is uniquely defined.

The task is to find, or estimate $\mathbf{x}^\star = \operatorname{argmin}_{\mathbf{x}} f(\mathbf{x})$. The notion of approximation we adopt is under the Euclidean norm: for an estimate $\hat{\mathbf{x}}$ how small can the *solution error* $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2$ be made? We will repeatedly use that for $\mathbf{H} = \mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d$, $\nabla f(\mathbf{x}) = \mathbf{H}(\mathbf{x} - \mathbf{x}^\star)$ (Lemma 6.3.1) which trivially follows from expanding the gradient term and invoking the normal equations $\mathbf{H}\mathbf{x}^\star = \mathbf{A}^\top \mathbf{y}$.

A crucial quantity in both solving and approximating RR is the *Hessian*[2] matrix $\mathbf{H}_\gamma = \mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d$. Obtaining $\mathbf{H}_\gamma$ costs $O(nd^2)$ time and $O(d^2)$ space due to $\mathbf{A}^\top \mathbf{A}$. A large enough $\gamma > 0$ will bound all singular values of $\mathbf{A}$ away from 0, so the input matrix is always full-rank. Thus, RR can be solved using the same direct methods as OLS from Section 2.3.1. The same nuances regarding the maintainence of $\mathbf{H}_\gamma$ by rank-one updates in $O(d^2)$ space and $O(nd^2)$ time from Section 2.3.1 apply. In contrast, our iterative result obtains $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 \leq \varepsilon_\star \|\mathbf{x}^\star\|_2$ using a sketch with error parameter $\varepsilon < 1$ and has running time $O(\log(1/\varepsilon_\star) nd/\varepsilon)$.

6.1.1 Related Work: Ridge Regression with Random Sketches

Sketch-and-solve Model

The sketch-and-solve analogues of the estimate weights are the 'classical' estimator

$$\mathbf{x}^C = \left(\mathbf{A}^\top \mathbf{S}^\top \mathbf{S}\mathbf{A} + \gamma \mathbf{I}_d\right)^{-1} \mathbf{A}^\top \mathbf{S}^\top \mathbf{S}\mathbf{y} \tag{6.5}$$

or the Hessian sketch estimator

$$\mathbf{x}^H = \left(\mathbf{A}^\top \mathbf{S}^\top \mathbf{S}\mathbf{A} + \gamma \mathbf{I}_d\right)^{-1} \mathbf{A}^\top \mathbf{y}. \tag{6.6}$$

Both of these definitions are simply the ridge regression analogues of the estimators defined in Equations 5.4 and 5.7. However, naïvely applying the subspace embedding guarantees of Table 5.1 would require a projection di-

[2] Due to the fact it is the matrix of second derivatives of $f(\mathbf{x})$ in (6.1) matrix. It is composed of the data covariance $\mathbf{A}^\top \mathbf{A}$ and a regularisation term $\gamma \mathbf{I}_d$.

mension $m = \tilde{O}(d \operatorname{poly}\log(d))$ meaning that the space usage grows as $\tilde{O}(d^2)$ for one-shot sketching. Thus, there is no asymptotic space benefit compared to the brute-force outer product solver. Additionally, the quadratic space dependence on d is problematic for high-dimensional data; this is a typical use-case of ridge regression, unlike least-squares regression which is not suited to large d. Although $\mathbf{x}^C$ and $\mathbf{x}^H$ can be computed quickly, they inherit the coarseness of approximation that we observed of sketch-and-solve methods in the previous chapter. Prior work using randomised sketch-and-solve methods focus on estimating the *objective function* $f(\mathbf{x})$ in small space. The space bounds grow proportional to the statistical dimension of the problem which can be significantly less than d [ACW17].

Definition 6.1.1 (Statistical Dimension)**.** *Let $\mathbf{A} \in \mathbb{R}^{n\times d}$ have rank d and singular values $\{\sigma_i\}_{i=1:d}$. Suppose that $\gamma > 0$ is a ridge regression hyperparameter, then the* statistical dimension of $\mathbf{A}$ *is*

$$\mathsf{sd}_\gamma(\mathbf{A}) = \sum_{i=1}^{d} \frac{\sigma_i^2}{\sigma_i^2 + \gamma} \tag{6.7}$$

Increasing γ reduces the effect of more σ_i^2 so the problem becomes easier as fewer directions are needed to describe the data when composed with the diagonal term $\gamma\mathbf{I}_d$.

Multi-Round Optimisation Model

To combat the weak coarse bounds of one-shot algorithms, some prior work has adopted multipass algorithms with running time growing as $O(\log(1/\varepsilon_\star))$ rather than poly $(1/\varepsilon_\star)$ for $\varepsilon_\star \ll \varepsilon$. These algorithms are randomised approximate Newton methods and can be thought of as being similar to the IHS of the previous section [PW16, LP19, CYD18, ZMJ+13, WLM+17, PW17]. These algorithms can give strong relative error guarantees such as $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_\star \leq \varepsilon_\star \|\mathbf{x}^\star\|_\star$ for $\star \in \{\mathbf{A}, 2\}$. However, they need $O(d)$ projections, resulting in a summary of $O(d) \times d$ which is the same space footprint as sketch-and-solve methods. Additionally, there is an $O(d^3)$ cost for SVD or related solver steps. Such behaviour was seen in the prior section for the CountSketch but it remains true of other random projections. Occasionally, this can be improved by keeping summaries with $m < d$ rows, but where this is possible, often a fresh sketch is required for every iteration [PW16, CYD18]. Each of these issues could hinder multi-shot iterative sketching for large-scale application.

Rather than using a fresh sketch for every iteration, some works have studied Iterative Hessian Sketch (or related approaches [WLM+17]) using only a single sketch [LP19]. One benefit of this approach is that it can be faster: only one of the passes is needed to obtain a sketch. However, if only one

sketch is used then either extra parameters need to be updated to make better gradient steps or a step size needs to be tuned for optimum performance. Unless parameters are correctly tuned, in practice, one may need to use a larger sketch size in this setting than if a fresh sketch is used for every iteration [WLM+17] which could increase the per-step solve time. Hence, we do not regard this as a direct replacement for the multi-sketch version of IHS.

6.1.2 Related Work: FD and Ridge Regression

Frequent Directions

As described in Section 2.3.2, obtaining the SVD of $\mathbf{H}_\gamma$ would yield an optimal preconditioner for RR. However, needing $O(nd^2)$ time, this is not a scalable approach. Hence, it would be ideal to perform an online or streaming variant of SVD keeping only the informative parts of the spectrum. Liberty introduced Frequent Directions (FD) (Algorithm 1) for exactly this problem; to find a matrix summary $\mathbf{B} \in \mathbb{R}^{m \times d}$ that well approximates the information one would obtain from performing an SVD of $\mathbf{A}$ [Lib13].

Here, $m \ll n$ and $m \ll d$ so $\mathbf{B}$ can be used as a proxy for $\mathbf{A}$ with many fewer rows. Choosing $m = \lceil k + 1/\varepsilon \rceil$, which is slightly larger than a rank parameter k, can yield an accuracy guarantee whose error bound decays with $\|\mathbf{A} - \mathbf{A}_k\|_F$. That is, the error term decays with the proportion of space not represented in the top k components [GLP16]. Given that ridge regression is useful when d is large, and large data can often be approximately low-rank [UT19], FD is a natural candidate sketch for approximating ridge regression. Further discussion surrounding Frequent Directions is given in Section 2.2.2.

As $\mathbf{H}_\gamma = \mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d$ is a fundamental operator in RR, one would hope that FD can be used as the sketch here, rather than random projection. The empirical findings of [GLP16] show that FD is much more accurate in estimating $\mathbf{A}^\top \mathbf{A}$ than random projections so we have cause for optimism. Nonetheless, it is only relatively recent work of [SP21] that shows how FD can be used for ridge regression. However, it remains the case that despite being a high-quality sketch, FD is under-exploited in regression tasks. Our motivation is to better understand how FD can be used in ridge regression. Prior work has failed to address whether *Frequent Directions can be used as a preconditioner for ridge regression to yield highly accurate estimates with low time and space overhead?*

Recall that the Frequent Directions guarantee of Theorem 2.2.1 is the following. Let $\Delta_k = \|\mathbf{A} - \mathbf{A}_k\|_F^2$ and $\alpha = 1/m - k$, if we use standard Frequent Directions then $\alpha' = \alpha$, however if using Robust Frequent Directions, $\alpha' = \alpha/2$.

$$\|\mathbf{A}\mathbf{x}\|_2^2 - \alpha' \Delta_k \|\mathbf{x}\|_2^2 \leq \left\|\bar{\mathbf{B}}\mathbf{x}\right\|_2^2 \leq \|\mathbf{A}\mathbf{x}\|_2^2 \tag{6.8}$$

where $\bar{\mathbf{B}} = \mathbf{B}$ if the 'standard' Frequent Directions algorithm is used, or $\bar{\mathbf{B}} = (\mathbf{B}^\top\mathbf{B} + \delta\mathbf{I}_d)^{1/2}$ if Robust Frequent Directions is used. Recall that δ is adaptively chosen regularisation applied to every direction.

FD for Ridge Regression

As mentioned in the previous section, using random projections in the sketch-and-solve model is typically used to find bounds on the objective function. In contrast, recent work of [SP21] returns an estimate $\hat{\mathbf{x}}$ which satisfies a coarse bound $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 \leq \varepsilon \|\mathbf{x}^\star\|_2$ in $O(d/\varepsilon)$ space. This is the first streaming $o(d^2)$ space algorithm with a guarantee on $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2$. Let $m = O(1/\varepsilon) < d$ be the number of rows in (and the rank of) $\mathbf{B} = \text{sketch}(\mathbf{A})$. If the interplay between the regularisation γ and the approximation error from FD are correctly balanced, then $\mathbf{x}^\star$ can be reasonably approximated up to ε relative error. The FD sketch is used by setting $\hat{\mathbf{H}} = \mathbf{B}^\top\mathbf{B} + \gamma\mathbf{I}_d$ to approximate $\mathbf{H}_\gamma$. It is also shown that the action $\hat{\mathbf{H}}^{-1}$ can be evaluated in $O(md)$ space. Algebraically, $\hat{\mathbf{x}} = \hat{\mathbf{H}}^{-1}\mathbf{A}^\top\mathbf{y}$ using $m = O(\|\mathbf{A} - \mathbf{A}_k\|_F^2/\gamma\varepsilon)$, which is a deterministic analogue of the Hessian Sketch weights $\mathbf{x}^H$ from (6.6).

In comparison to randomised methods, the number of rows in the sketch grows according to $O(1/\varepsilon)$ for ε accuracy. Although this improves upon the $O(1/\varepsilon^2)$ rows needed for randomised sketches, it is still problematic if ε is very small, say 10^{-8} or less. This is common if our estimate weights are required to accurately describe the 'true' optimal weights as seen in the previous chapter. This deficiency motivates our study of FD in the multi-round optimisation model for an approximate Newton method.

6.1.3 Contributions

Before presenting our contributions, we first highlight shortcomings of previous methods.

1. Section 5.4 showed that the most expensive part of an iterative method was the sketching step.

2. Established iterative methods using randomised sketches need a fresh sketch for every iteration (Section 6.1.1).

3. If only one sketch is used then we need to tune step size parameters or use a Gaussian sketch and this is not suitable at large-scale.

4. Only coarse approximation guarantees have been established in using an FD sketch.

Taken together, these four points motivate our contribution. Rather than generating a fresh sketch for every step, we seek to only find one sketch.

Although this could be an expensive upfront cost, if it is accurate enough then there could be a deferred time benefit in how many gradient steps are needed. Note that we do not measure the cost of revisiting the data, which could be expensive. The prior work presented above fails to address these criteria simultaneously.

We will devise an iterative scheme for which the coefficient error[3] $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2$ can be made exponentially small. The iterations will use a preconditioner that uses only $O(md)$ space for $m < d$ which is found by combining the Frequent Directions (FD) algorithm (Algorithm 1) with the regularisation term. FD returns $\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ such that $\mathbf{B}^\top\mathbf{B} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top$ is approximately $\mathbf{A}^\top\mathbf{A}$. The structure of our approximate Hessian matrix $\mathbf{B}^\top\mathbf{B} + \gamma\mathbf{I}_d$ permits small-space inversion as we can consider the rank m space spanned by the sketch separate to the remainder of the space. Consequently, inversion will only cost $O(m)$ time as the orthogonal matrices are self-inverse, so the only inverse that needs to be taken is over $\mathbf{\Sigma}^2 + \gamma\mathbf{I}_m$, this is much more scalable. Hence the approximate inverse can be applied to a vector in $O(md)$ time in comparison to $O(d^2)$ space and $O(d^3)$ inversion cost if $O(d)$ projections are used. Note that even though FD uses the SVD, since the sketch dimension $m < d$, the time for an SVD is quadratic in the sketch dimension m, not the feature dimension. Our technical contribution leverages properties of the FD sketch to control the quantity $\left\|\mathbf{I}_d - \hat{\mathbf{H}}^{-1}\mathbf{H}\right\|_2$.

Our work is the first to give high-accuracy guarantees on the returned solution vector. Although we are not the first to study FD in regression tasks, prior work has different motivations and presents *complementary* results to ours. For example, [Hua18] propose using FD for *adversarial online learning* through an approximate Newton method. However, their application (and hence guarantees) are much different from ours; no bounds on the solution estimation $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2$ are provided.

1. We present a *simple* but novel analysis of Frequent Directions (FD) in an iterative scheme for small-space and scalable sketched ridge regression. Our approach obtains a *strong relative-error* approximation guarantee to the optimal weights $\mathbf{x}^\star$. After T iterations the estimate $\hat{\mathbf{x}}$ achieves, $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 \leq \varepsilon^T \|\mathbf{x}^\star\|_2$. Our approach is entirely deterministic. The single-pass approximation of weights using FD was given by [SP21] but this does not obtain the exponentially decaying error over multiple rounds. Hence our estimates are significantly more accurate, although, we need $O(\log(1/\varepsilon^T))$ gradient steps whereas their bound operates in a one-shot model.

2. We obtain an $O(md)$ sketch with $m \ll d$ which "preconditions" the

[3]The switch from $\mathbf{A}$-norm to Euclidean norm is deliberate.

gradient descent so that convergence is rapid. As our summary is retained in factored form, iterates are evaluated in *linear time*, roughly $O(md)$, which is substantially more scalable than the polynomial iteration time complexities of [PW16, CYD18] or the typical $O(d^2)$ of preconditioned gradient methods with randomised sketches [Woo14b, Section 2.6].

3. We vindicate our approach with empirical evidence, showing that despite our sketch being more expensive to obtain, it yields estimates that converge more quickly in number of iterations and wall-clock time.

6.2 Structural Properties of $\hat{\mathbf{H}}$

Rather than computing the exact Hessian matrix $\mathbf{H}_\gamma$, we estimate it using the FD sketch $\mathbf{B}$. The estimated Hessian $\hat{\mathbf{H}} = \mathbf{B}^\top\mathbf{B}+\gamma\mathbf{I}_d$ will be our approximation to $\mathbf{H}_\gamma$. Although our iterates in Algorithm 9 are written using $\hat{\mathbf{H}}^{-1}$, this is never computed explicitly and we only need its behaviour as an operator. We defer discussing computation issues surrounding $\hat{\mathbf{H}}^{-1}$ until Section 6.5. Before giving the details of the iterative scheme, there are a few structural results that describe the behaviour of $\hat{\mathbf{H}}$. To introduce these results, we first need to understand order properties of *symmetric positive definite matrices*, known as the *Löwner Order*.

A symmetric positive definite (s.p.d.) matrix $\mathbf{X} \in \mathbb{R}^{d\times d}$ is symmetric and has all eigenvalues strictly bounded away zero on the positive side, written $\mathbf{X} \succ \mathbf{0}$. Positive definite means that $\mathbf{u}^\top\mathbf{X}\mathbf{u} > 0$ for all $\mathbf{u}$ nonzero. Applied to covariance matrices of full rank, for example $\mathbf{X} = \mathbf{A}^\top\mathbf{A}$ this is equivalent to $\|\mathbf{A}\mathbf{u}\|_2^2 > 0$ for all $\mathbf{u} \neq \mathbf{0}_{d\times 1}$. The strictness of each of the above inequalities can be relaxed to allow equality if we permit symmetric positive *semi* definite matrices (spsd) For any two sp(s)d matrices we write $\mathbf{X} \preceq \mathbf{Y}$ if and only if $\mathbf{Y} - \mathbf{X} \succeq \mathbf{0}_{d\times d}$.

Some useful facts on the Löwner ordering are given below (see [Har],[DK06, Appendix A]):

Fact 6.2.1. *Let $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ be arbtitrary symmetric positive definite matrices. The ordering takes place over the cone of s.p.d matrices where we write $\mathbf{X} \preceq \mathbf{Y}$ iff $\mathbf{0} \preceq \mathbf{X} - \mathbf{Y}$.*

1. *If $\mathbf{X} \preceq \mathbf{Y}$ then $\mathbf{Z}\mathbf{X}\mathbf{Z}^\top \preceq \mathbf{Z}\mathbf{Y}\mathbf{Z}^\top$. In fact, this is an if and only if when $\mathbf{Z}$ is of full rank.*

2. *Let $\lambda_{\min}$ and $\lambda_{\max}$ be the smallest and largest eigenvalues of $\mathbf{X}$. Then $\lambda_{\min}\mathbf{I}_d \preceq \mathbf{X} \preceq \lambda_{\max}\mathbf{I}_d$.*

3. *If $\mathbf{X} \preceq \mathbf{Y}$ then $\mathbf{Y}^{-1} \preceq \mathbf{X}^{-1}$*

Under the Löwner ordering we recognise that Theorem 2.2.1 and Equation (6.8) are equivalent to the following relation which is the starting point of our analysis.

$$\mathbf{A}^\top\mathbf{A} - \alpha'\Delta_k\mathbf{I}_d \preceq \mathbf{B}^\top\mathbf{B} \preceq \mathbf{A}^\top\mathbf{A} \tag{6.9}$$

6.2.1 Exploiting the Löwner Order for $\hat{\mathbf{H}}$

The Löwner properties from Fact 6.2.1 allow us to establish the following spectral bounds between the approximate and true Hessians:

Lemma 6.2.1. *Let* $\mathbf{A} \in \mathbb{R}^{n\times d}$, $\gamma > 0$ *and* $\mathbf{H}_a = \mathbf{A}^\top\mathbf{A} + a\mathbf{I}_d$. *If* $\gamma' = \gamma - s > 0$, *then* $\gamma'/\gamma\mathbf{H}_\gamma \preceq \mathbf{H}_{\gamma'} \preceq \mathbf{H}_\gamma$.

Proof. This proof builds upon the following simple scalar bound: if $x \geq 0$ and let $t > s > 0$, then

$$\frac{t-s}{t}(x+t) \leq x + t - s < x + t.$$

The upper bound follows trivially since $t - s < t$. For the lower bound,

$$\begin{aligned}\frac{t-s}{t}(x+t) &= \frac{t-s}{t}x + (t-s)\\ &\leq x + (t-s)\end{aligned}$$

since $\frac{t-s}{t} < 1$ and $x \geq 0$. We apply this result to every entry on the diagonal matrix $\mathbf{\Sigma}$ containing the squared singular values of $\mathbf{A}$ which yields:

$$\frac{\gamma'}{\gamma}(\sigma_i^2 + \gamma) \leq \sigma_i^2 + \gamma' \leq \sigma_i^2 + \gamma$$

so by applying Fact 6.2.1(1) above with $\mathbf{Z} = \mathbf{V}$

$$\frac{\gamma'}{\gamma}\left(\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top + \gamma\mathbf{I}_d\right) \preceq \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top + \gamma'\mathbf{I}_d \preceq \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top + \gamma\mathbf{I}_d.$$

Since $\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top = \mathbf{A}^\top\mathbf{A}$, we obtain $\gamma'/\gamma\mathbf{H}_\gamma \preceq \mathbf{H}_{\gamma'} \preceq \mathbf{H}_\gamma$ as required. □

We can combine the above result with the FD guarantee to obtain the following corollary. This will be useful for later spectral analysis of the sketched Hessian $\hat{\mathbf{H}}$.

Corollary 6.2.1. *Let* $\mathbf{H}_\gamma = \mathbf{A}^\top\mathbf{A} + \gamma\mathbf{I}_d$ *and* $\hat{\mathbf{H}}_\gamma = \mathbf{B}^\top\mathbf{B} + \gamma\mathbf{I}_d$. *Then* $\gamma'/\gamma\mathbf{H}_\gamma \preceq \hat{\mathbf{H}}_\gamma \preceq \mathbf{H}_\gamma$ *and* $\mathbf{H}_\gamma^{-1} \preceq \hat{\mathbf{H}}_\gamma^{-1} \preceq \gamma/\gamma'\mathbf{H}_\gamma^{-1}$.

Proof. Let $\gamma' = \gamma - \alpha'\Delta_k > 0$. After adding $\gamma\mathbf{I}_d$ to all terms in (6.9) we have:

$$\mathbf{A}^\top\mathbf{A} + \gamma'\mathbf{I}_d \preceq \mathbf{B}^\top\mathbf{B} + \gamma\mathbf{I}_d \preceq \mathbf{A}^\top\mathbf{A} + \gamma\mathbf{I}_d.$$

Algorithm 9: Iterative Frequent Directions Ridge Regression

Input: Data $\mathbf{A} \in \mathbb{R}^{n \times d}$, targets $\mathbf{y} \in \mathbb{R}^n$, regularisation $\gamma > 0$, sketch size m, num. iterations $T \geq 1$, Method $\mathsf{Sk} \in \{\mathsf{FD}, \mathsf{RFD}\}$

Output: Weights $\hat{\mathbf{x}} \in \mathbb{R}^d$

$\mathbf{B}, \delta = \mathsf{Sk}(\mathbf{A})$ ▷ $\delta = 0$ if $\mathsf{Sk} = \mathsf{FD}$ is nonzero otherwise

$\hat{\mathbf{H}} = \mathbf{B}^\top \mathbf{B} + (\gamma + \delta)\mathbf{I}_d$

$\mathbf{x}^{(0)} = \mathbf{0}_d$

for $t = 1 : T$ **do**

  $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \hat{\mathbf{H}}^{-1}\left[\mathbf{A}^\top\left(\mathbf{A}\mathbf{x}^{(t)} - \mathbf{y}\right) + \gamma \mathbf{x}^{(t)}\right]$

end

$\hat{\mathbf{x}} = \mathbf{x}^{(t)}$

Applying this relation to the result of Lemma 6.2.1 obtains:

$$\frac{\gamma'}{\gamma}\left(\mathbf{A}^\top\mathbf{A} + \gamma\mathbf{I}_d\right) \preceq \mathbf{A}^\top\mathbf{A} + \gamma'\mathbf{I}_d \preceq \mathbf{B}^\top\mathbf{B} + \gamma\mathbf{I}_d \preceq \mathbf{A}^\top\mathbf{A} + \gamma\mathbf{I}_d \tag{6.10}$$

Finally, we can use the inverse rule Fact 6.2.1(3) above to obtain:

$$\left(\mathbf{A}^\top\mathbf{A} + \gamma\mathbf{I}_d\right)^{-1} \preceq \left(\mathbf{B}^\top\mathbf{B} + \gamma\mathbf{I}_d\right)^{-1} \preceq \frac{\gamma}{\gamma'}\left(\mathbf{A}^\top\mathbf{A} + \gamma\mathbf{I}_d\right)^{-1}. \tag{6.11}$$

□

6.3 Iterative Methods using Frequent Directions for Ridge Regression

Much like in the Iterative Hessian Sketch models from Section 5.3, we will refine an initial estimate of the weights to better approximate $\mathbf{x}^\star$. This is done via iterative gradient steps at the cost of further passes over the data. We will argue that these gradient steps exploit second-order information yet are still cheap to compute. Thus, our proposal (Algorithm 9) is an approximate Newton-type algorithm that exploits scalable approximation to the Hessian $\mathbf{H}_\gamma$. Our approach is reminiscent of other iterative sketching algorithms [PW16, CYD18]. In common with them is that our summary $\mathbf{B}$ has $o(d)$ rows, a substantial saving over explicitly using the $d \times d$ size Hessian matrix, or the full subspace embedding from Section 5.3. The structure of the sketched approximation $\hat{\mathbf{H}}$ avoids the $O(d^3)$ time cost for inversion due to the trick of [SP21] or Woodbury's Identity and their work can also be considered as one iteration of Algorithm 9.

To prove correctness of Algorithm 9 we closely follow typical proofs for gradient descent-type algorithms. We use Lemma 6.3.1 to express $\nabla f(\mathbf{x}) = \mathbf{H}_\gamma(\mathbf{x} - \mathbf{x}^\star)$. Then we are able to analyse the sequence of iterates relative to their distance from $\mathbf{x}^\star$. Crucially, we obtain $\mathbf{x}^{(t+1)} - \mathbf{x}^\star = \left(\mathbf{I}_d - \hat{\mathbf{H}}^{-1}\mathbf{H}_\gamma\right)\left(\mathbf{x}^{(t)} - \mathbf{x}^\star\right)$.

Lemma 6.3.1. $\nabla f(\mathbf{x}) = \mathbf{H}(\mathbf{x} - \mathbf{x}^\star)$

Proof. Follows directly from the normal equations $\mathbf{H}\mathbf{x}^\star = \mathbf{A}^\top \mathbf{y}$:

$$\begin{aligned}
\nabla f(\mathbf{x}) &= \mathbf{A}^\top (\mathbf{A}\mathbf{x} - \mathbf{y}) + \gamma \mathbf{x} \\
&= \left(\mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d\right) \mathbf{x} - \mathbf{A}^\top \mathbf{y} \\
&= \left(\mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d\right) (\mathbf{x} - \mathbf{x}^\star) \\
&= \mathbf{H} (\mathbf{x} - \mathbf{x}^\star)
\end{aligned}$$

where the penultimate equation is due to the normal equations:

$$\left(\mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d\right) \mathbf{x}^\star = \mathbf{A}^\top \mathbf{y}.$$

□

A strength of our work is in the simplicity of its analysis as the main result, Theorem 6.3.1, will follow a standard gradient descent-type proof. We build upon Corollary 6.2.1 which argues that approximate Hessian has similar spectral behaviour to the true Hessian. First, we represent the current iterate $\mathbf{x}^{(t+1)}$ as a function of the previous iterate's distance from the optimal solution.

Lemma 6.3.2. *The sequence of iterates* $\{\mathbf{x}^{(t+1)}\}_{i \geq 0}$ *follows:*

$$\mathbf{x}^{(t+1)} - \mathbf{x}^\star = \left(\mathbf{I}_d - \hat{\mathbf{H}}^{-1}\mathbf{H}\right)\left(\mathbf{x}^{(t)} - \mathbf{x}^\star\right). \tag{6.12}$$

Proof. Applying Lemma 6.3.1 to the iterates as defined in Algorithm 9, Line 5, we obtain:

$$\mathbf{x}^{(t+1)} - \mathbf{x}^\star = \mathbf{x}^{(t)} - \mathbf{x}^\star - \hat{\mathbf{H}}^{-1}\mathbf{H}\left(\mathbf{x}^{(t)} - \mathbf{x}^\star\right)$$

which yields the claim after factorisation. □

Taking the norm of both sides of Equation 6.12 and invoking submultiplicativity we have $\left\|\mathbf{x}^{(t+1)} - \mathbf{x}^\star\right\|_2 \leq \left\|\mathbf{I}_d - \hat{\mathbf{H}}^{-1}\mathbf{H}\right\|_2 \left\|\mathbf{x}^{(t)} - \mathbf{x}^\star\right\|_2$. The first 2-norm is the spectral norm over matrices, while the second 2-norm is the Euclidean norm over vectors. Hence, to show $\left\|\mathbf{x}^{(t+1)} - \mathbf{x}^\star\right\|_2 \leq \left\|\mathbf{x}^{(t)} - \mathbf{x}^\star\right\|_2$ it suffices to show $\left\|\mathbf{I}_d - \hat{\mathbf{H}}^{-1}\mathbf{H}\right\|_2 < 1$. Showing this amounts to manipulating the FD guarantee of Theorem 2.2.1 alongside properties of the Löwner ordering given in Fact 6.2.1. The starting point is to analyse the spectrum of $\mathbf{I}_d - \hat{\mathbf{H}}^{-1}\mathbf{H}_\gamma$. By matrix similarity we instead analyse $\mathbf{I}_d - \hat{\mathbf{H}}^{-1/2}\mathbf{H}_\gamma\hat{\mathbf{H}}^{-1/2}$ but specifically need the extremal eigenvalues of the auxiliary matrix $\mathbf{E} = \hat{\mathbf{H}}^{-1/2}\left(\mathbf{A}^\top\mathbf{A} + \gamma\mathbf{I}_d\right)\hat{\mathbf{H}}^{-1/2}$. Crucially, we show that all $\lambda_i(\mathbf{E}) \in [1, 1/1-q]$ where $q = {\|\mathbf{A} - \mathbf{A}_k\|_F^2}/{(m-k)\gamma}$. This implies that the largest distortion $|1 - \lambda_i(\mathbf{E})|$ occurs at $|1 - \frac{1}{1-q}|$.

Lemma 6.3.3. *If* $2\alpha\Delta_k < \gamma$, *then* $\left\|\mathbf{I}_d - \hat{\mathbf{H}}^{-1}\mathbf{H}\right\|_2 < 1$

Proof. Since $\hat{\mathbf{H}}$ is positive definite it has a positive definite square root $\mathbf{R} = \hat{\mathbf{H}}^{1/2}$. We pre and post multiply to obtain $\mathbf{R}(\mathbf{I}_d - \hat{\mathbf{H}}^{-1}\mathbf{H})\mathbf{R}^{-1} = \mathbf{I}_d - \hat{\mathbf{H}}^{-1/2}\mathbf{H}\hat{\mathbf{H}}^{-1/2}$. Since $\mathbf{I}_d - \hat{\mathbf{H}}^{-1}\mathbf{H}$ is similar to $\mathbf{I}_d - \mathbf{E}$ with $\mathbf{E} = \hat{\mathbf{H}}^{-1/2}\mathbf{H}\hat{\mathbf{H}}^{-1/2}$, they share the same eigenvalues. Hence we must show $\|\mathbf{I}_d - \mathbf{E}\|_2$ is no greater than 1, specifically $\max_{\mathbf{u}} |1 - \mathbf{u}^\top \mathbf{E}\mathbf{u}| < 1$. Since $\mathbf{E}$ is spsd, we may use the Rayleigh quotient characterisation of eigenvalues (cf. Section 5.1.3) and analyse $\mathbf{u}^\top \mathbf{E}\mathbf{u}$. To do so, we need a few properties of the FD sketch. Let $\alpha = 1/m - k$ and $\Delta_k = \|\mathbf{A} - \mathbf{A}_k\|_F^2$. In Corollary 6.2.1 we showed that

$$\mathbf{A}^\top \mathbf{A} + (\gamma - \alpha\Delta_k)\mathbf{I}_d \preceq \mathbf{B}^\top \mathbf{B} + \gamma \mathbf{I}_d \preceq \mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d. \tag{6.13}$$

Provided that $\gamma > \alpha\Delta_k \mathbf{I}_d$, all of the above terms are strictly lower bounded by $\mathbf{0}_{d\times d}$. This is equivalent to saying that all eigenvalues are positive, hence the matrices are full rank and inverses are well-defined. Denote $\gamma' = \gamma - \alpha\Delta_k$. Corollary 6.2.1 shows that $\gamma'/\gamma \left(\mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d\right) \preceq \mathbf{A}^\top \mathbf{A} + \gamma' \mathbf{I}_d$. Let $q = \alpha\Delta_k/\gamma > 0$ so that $\gamma'/\gamma = 1 - q$. Invoking (6.13) we obtain the ordering:

$$(1-q)\left(\mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d\right) \preceq \mathbf{B}^\top \mathbf{B} + \gamma \mathbf{I}_d \preceq \mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d.$$

Now use Fact 6.2.1(1) on all three terms in the above ordering with $\mathbf{Z} = \hat{\mathbf{H}}^{-1/2}$, noting that $\mathbf{Z}^\top \mathbf{H}_\gamma \mathbf{Z} = \mathbf{E}$. Again, since all of the matrices in question are symmetric positive definite, they have unique symmetric positive definite square roots so we are free to apply the Löwner multiplication order.

$$\begin{aligned}(1-q)\hat{\mathbf{H}}^{-1/2}\left(\mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d\right)\hat{\mathbf{H}}^{-1/2} &\preceq \mathbf{I}_d \\ &\preceq \hat{\mathbf{H}}^{-1/2}\left(\mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d\right)\hat{\mathbf{H}}^{-1/2}.\end{aligned} \tag{6.14}$$

The above equation also implies that $\hat{\mathbf{H}}^{-1/2}\left(\mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d\right)\hat{\mathbf{H}}^{-1/2} \preceq \frac{1}{1-q}\mathbf{I}_d$. Hence, we also have

$$\mathbf{I}_d \preceq \hat{\mathbf{H}}^{-1/2}\left(\mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d\right)\hat{\mathbf{H}}^{-1/2} \preceq \frac{1}{1-q}\mathbf{I}_d. \tag{6.15}$$

The Löwner ordering also ensures that $\lambda_{\min}(\mathbf{M})\mathbf{I} \preceq \mathbf{M} \preceq \lambda_{\max}(\mathbf{M})\mathbf{I}$. Hence, we have shown that

$$\lambda_i(\hat{\mathbf{H}}^{-1/2}\left(\mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d\right)\hat{\mathbf{H}}^{-1/2}) \in \left[1, \frac{1}{1-q}\right]. \tag{6.16}$$

Finally, it remains to ensure that $\max_{\mathbf{u}} |1 - \mathbf{u}^\top \mathbf{E}\mathbf{u}| < 1$. Since all $\lambda_i(\mathbf{E}) \geq 1$, the largest displacement occurs at $\lambda_{\max}(\mathbf{E})$. Therefore, q must be set so that $\left|1 - \frac{1}{1-q}\right| < 1$ that is,

$$\frac{q}{1-q} < 1 \tag{6.17}$$

which occurs provided $q \in (0, 1/2)$ and is thus satisfied by the assumption $2\alpha\Delta_k < \gamma$ as $q = \alpha\Delta_k/\gamma$. □

Remark 6.3.1. *We claim that the assumption on γ in Lemma 6.3.3 is valid. Since $m - k \geq 1$ the assumption asks that γ is some fraction of the tail or residual of the mass. As ridge regression is intended to apply in the high-dimensional setting with much redundancy in the feature space, it is typical to assume that the regularisation exceeds the tail in such a fashion.*

Recall that for convergence we required $\|\mathbf{I}_d - \hat{\mathbf{H}}^{-1}\mathbf{H}\|_2 < 1$ which is satisfied provided $|1 - \frac{1}{1-q}| < 1$. Hence, we need $q < 1/2$ which is true by the assumption of Lemma 6.3.3. The preceding result can be used iteratively. In summary, the following theorem establishes that choosing $\gamma > 2\alpha\Delta_k$ ensures the distance from $\mathbf{x}^{(t+1)}$ to $\mathbf{x}^\star$ is at most an $\alpha\Delta_k/\gamma$ factor smaller than that of $\mathbf{x}^{(t)}$ to $\mathbf{x}^\star$. The convergence theorem we present follows by combining all of the pieces we have established above.

Theorem 6.3.1. *Let $b \in (0, 1/2)$ and suppose that $\alpha\Delta_k = b\gamma$. Running $t + 1$ steps of Algorithm 9 with Frequent Directions satisfies*

$$\left\|\mathbf{x}^{(t+1)} - \mathbf{x}^\star\right\|_2 \leq \left(\frac{b}{1-b}\right)^{t+1} \|\mathbf{x}^\star\|_2 \tag{6.18}$$

Proof. Let $\beta = q/1-q$ as in Equation (6.17) and $\gamma' = \gamma - \alpha\Delta_k$. Hence, $\beta = \alpha\Delta_k/\gamma'$. Assuming that $\alpha\Delta_k = b\gamma$ so $\gamma' = (1-b)\gamma$ means $\beta = b/1-b$. Since $b < 1/2$ we have $\alpha\Delta_k < \gamma/2$ hence Lemma 6.3.3 establishes that $\left\|\mathbf{I}_d - \hat{\mathbf{H}}^{-1}\mathbf{H}\right\|_2 \leq \beta$. Thus; $\left\|\mathbf{x}^{(t+1)} - \mathbf{x}^\star\right\|_2 \leq \beta \left\|\mathbf{x}^{(t)} - \mathbf{x}^\star\right\|_2$. By induction, we can iterate this argument to obtain $\left\|\mathbf{x}^{(t+1)} - \mathbf{x}^\star\right\|_2 \leq \beta^{t+1} \|\mathbf{x}^\star\|_2$ which follows by recalling that $\mathbf{x}^{(0)} = \mathbf{0}_d$. □

Theorem 6.3.1 demonstrates that convergence is governed by an interplay between the regularisation parameter and the tail of mass. Let $\beta = \frac{b}{1-b}$ so that $\beta = \alpha\Delta_k/(\gamma - \alpha\Delta_k)$. When β is smaller, descent is faster. Hence, we can understand the tradeoff between regularisation and sketch accuracy necessary for convergence. Decreasing β can be achieved by increasing γ or by reducing $\alpha\Delta_k$. The former regularises the data more (less importance is placed on the observed data) while the latter is equivalent to choosing a greater sketch size. For example, taking $b = 1/4$, Theorem 6.3.1 yields $\gamma = 4\alpha\Delta_k$ so $\beta = 1/3$ & the error decreases by (at worst) a factor of 3 each iteration.

Remark 6.3.2. *Although $\|\mathbf{A} - \mathbf{A}_k\|_F^2$ may not be known (or cannot be estimated) in advance, setting $k = 0$ amounts to taking $b = \|\mathbf{A}\|_F^2/m\gamma$, but this may be too pessimistic in practice: $\|\mathbf{A}\|_F^2$ can be maintained in small space while observing the stream.*

6.4 Improving Convergence with Robust Frequent Directions

While the assumption of $2\alpha\Delta_k < \gamma$ in Theorem 6.3.1 is valid, it would be preferable to weaken this constraint. Indeed, this is possible due to the improved sketch quality of Robust Frequent Directions. Theorem 6.4.1 weakens the assumption of $2\alpha\Delta_k < \gamma$ to ask for $\alpha\Delta_k < \gamma$, while simultaneously improving the rate of convergence from $b/1-b$ to $b/2-b$. Recalling the previous example of taking $b = 1/4$, this is an improvement from $\beta = 1/3$ by Theorem 6.3.1 to $\beta = 1/7$.

We can slot the robust variant of FD into the iterative framework. The proofs follow on as before with a mild adjusting of the constants. Again, the key technical detail is, for $\hat{\mathbf{H}}_{\delta,\gamma} = \mathbf{B}^\top\mathbf{B} + (\delta+\gamma)\mathbf{I}_d$, establishing that $\|\mathbf{I}_d - \hat{\mathbf{H}}_{\delta,\gamma}\mathbf{H}\|_2 < 1$. The improvement in using RFD is that we can weaken the hypothesis necessary for the result.

Lemma 6.4.1. *If $\alpha\Delta_k < \gamma$, then $\left\|\mathbf{I}_d - \hat{\mathbf{H}}_{\delta,\gamma}^{-1}\mathbf{H}\right\|_2 < 1$*

This proof follows the same outline as for Lemma 6.3.3 except we can leverage the improved RFD guarantee.

Proof. We follow the proof of Lemma 6.3.3 almost exactly but with the following modifications. In Equation (6.13) we use the improved RFD guarantee (Theorem 2.2.1) which tightens (6.9) to

$$\mathbf{A}^\top\mathbf{A} + \left(\gamma - \frac{\alpha\Delta_k}{2}\right)\mathbf{I}_d \preceq \mathbf{B}^\top\mathbf{B} + \gamma\mathbf{I}_d \preceq \mathbf{A}^\top\mathbf{A} + \gamma\mathbf{I}_d.$$

Then take $\gamma' = \gamma - \alpha\Delta_k/2$ and $q = \alpha\Delta_k/2\gamma$. Hence, $\gamma'/\gamma = 1 - q$ as before. As in (6.17), we require $q/(1-q) < 1$ so $q < 1/2$. By assumption $\alpha\Delta_k < \gamma$ as $q = \alpha\Delta_k/\gamma$ so $q < 1/2$ is satisfied. □

Due to the theory established for Theorem 6.3.1, we can essentially repeat the proof, adjusting for the necessary constants which arise due to using the RFD sketch $\mathbf{B}^\top\mathbf{B} + \delta\mathbf{I}_d$ instead of $\mathbf{B}^\top\mathbf{B}$.

Theorem 6.4.1. *Let $b \in (0,1)$ and suppose that $\alpha\Delta_k = b\gamma$. Algorithm 9 with Robust Frequent Directions satisfies $\left\|\mathbf{x}^{(t+1)} - \mathbf{x}^\star\right\|_2 \leq \left(\frac{b}{2-b}\right)^{t+1}\|\mathbf{x}^\star\|_2$.*

Proof. Same proof as Theorem 6.3.1 except noting that the choice of q is

different due to Lemma 6.4.1, then $\beta = q/1-q$ reduces to $\beta = b/2-b$

$$\begin{aligned}\beta &= \frac{q}{1-q} \\ &= \frac{\alpha\Delta_k/2\gamma}{(2\gamma-\alpha\Delta_k)/2\gamma} \\ &= \frac{\alpha\Delta_k}{2\gamma-\alpha\Delta_k} \\ &= \frac{b}{2-b}.\end{aligned}$$

□

6.5 Time and Space analysis of Algorithm 9

We recall the following notions that should help understand the computational benefit of using Algorithm 9. The gradient $\nabla f(\mathbf{x}) \in \mathbb{R}^d$ and can be computed in $O(nd)$ time. This is done by applying $\mathbf{A}^\top\mathbf{A} + \gamma\mathbf{I}_d$ through matrix-vector products, evaluated from right to left to avoid the matrix-matrix multiplication.

Since we operate in the multi-round optimisation model (Definition 2.1.5), we can think of the gradient steps as simple passes through the dataset. Alternatively, in the distributed setting this may be thought of as many parties sending the coordinator their local part of the gradient vector which are centrally summed to obtain the full gradient. Algorithm 9 complements both of these perceptions. In the single-party setting we think of a user as passing over the data once to sketch the data and any other passes are for gradient evaluations. In the multi-party setting, one round is taken for all parties to communicate local $O(md)$ sized sketches of their data. Thanks to the mergeability of FD, the coordinator can merge the sketches to obtain one global sketch. Any further rounds of communication only require the parties to send local gradients as is the case for gradient descent.

If the iterations $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \hat{\mathbf{H}}^{-1}\nabla f(\mathbf{x}^{(t)})$ are performed naïvely, then we will use space $O(d^2)$, which is no better than a brute force solve, and need time $O(nd+d^2)$ per update step. The following lemma shows that the iterations can be performed in time $O((n+m)d)$ time while using only $O(md)$ space. Since $m \ll d, n$ this can be a huge saving, meaning that we get to use approximate second-order information for almost the same cost as evaluating a gradient. The idea follows by recalling that FD maintains $\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ in factored form, so we can cleverly distribute these orthonormal bases and diagonal matrices out of the inverse after applying the Woodbury matrix identity. Each of these matrices has size $\mathbf{U} \in \mathbb{R}^{m\times m}, \mathbf{\Sigma} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{d\times m}$. This approach means we never exceed space $O(md)$ for $m < d$ and any matrix inverses are either over diagonal or orthogonal matrices. This was originally observed by

[SP21] and we combine their result with the iterations to obtain the necessary time/space guarantees of our algorithm.

Lemma 6.5.1.

$$\left(\mathbf{B}^\top\mathbf{B}+\gamma\mathbf{I}_d\right)^{-1}=\frac{1}{\gamma}\mathbf{I}_d-\frac{1}{\gamma}\mathbf{B}^\top(\mathbf{B}\mathbf{B}^\top+\gamma\mathbf{I}_m)^{-1}\mathbf{B}$$

Proof. First note that

$$\left(\mathbf{B}^\top\mathbf{B}+\gamma\mathbf{I}_d\right)^{-1}=\frac{1}{\gamma}\left(\frac{\mathbf{B}^\top\mathbf{B}}{\gamma}+\mathbf{I}_d\right)^{-1}. \tag{6.19}$$

We apply the Woodbury matrix identity

$$(\mathbf{I}+\mathbf{U}\mathbf{V})^{-1}=\mathbf{I}-\mathbf{U}\left(\mathbf{I}+\mathbf{V}\mathbf{U}\right)^{-1}\mathbf{V}$$

to $\left(\frac{\mathbf{B}^\top\mathbf{B}}{\gamma}+\mathbf{I}_d\right)^{-1}$ with $\mathbf{U}=\mathbf{B}^\top/\gamma$ and $\mathbf{V}=\mathbf{B}$. This achieves

$$\mathbf{I}_d-\frac{1}{\gamma}\mathbf{B}^\top(\frac{1}{\gamma}\mathbf{B}\mathbf{B}^\top+\mathbf{I}_m)^{-1}\mathbf{B}.$$

Bringing the $1/\gamma$ term into the inverse term yields

$$\mathbf{I}_d-\mathbf{B}^\top(\mathbf{B}\mathbf{B}^\top+\gamma\mathbf{I}_m)^{-1}\mathbf{B}$$

which after premultiplying by the $1/\gamma$ factor from (6.19) achieves the claim. □

Lemma 6.5.2. *The iteration* $\mathbf{x}^{(t+1)}=\mathbf{x}^{(t)}-\hat{\mathbf{H}}^{-1}\nabla f(\mathbf{x}^{(t)})$ *can be implemented in* $O((m+n)d)$ *time and* $O(md)$ *space.*

Proof. Let $\mathbf{B}=\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ be the $m\times d$ FD sketch of $\mathbf{A}\in\mathbb{R}^{n\times d}$. The same proof holds for RFD except we have to account for the extra regularisation parameter δ that is applied to each of the directions in the sketch (see Algorithm 1). This is easily accounted for by resetting $\gamma\leftarrow\gamma+\delta$.

Note that $\mathbf{U}\in\mathbb{R}^{m\times m}$ and $\mathbf{V}^\top\in\mathbb{R}^{m\times d}$ have orthonormal columns. Then we apply Lemma 6.5.1 on $\hat{\mathbf{H}}^{-1}$ to obtain:

$$\hat{\mathbf{H}}^{-1}=\frac{1}{\gamma}\mathbf{I}_d-\frac{1}{\gamma}\mathbf{B}^\top(\mathbf{B}\mathbf{B}^\top+\gamma\mathbf{I}_m)^{-1}\mathbf{B}$$

after which using the SVD $\mathbf{B}=\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ results in

$$\hat{\mathbf{H}}^{-1}=\frac{1}{\gamma}\mathbf{I}_d-\frac{1}{\gamma}\mathbf{V}^\top\mathbf{\Sigma}\mathbf{U}^\top(\mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top+\gamma\mathbf{I}_m)^{-1}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top.$$

Orthonormality of $\mathbf{U}$ then gives

$$\hat{\mathbf{H}}^{-1}=\frac{1}{\gamma}\mathbf{I}_d-\frac{1}{\gamma}\mathbf{V}^\top(\mathbf{\Sigma}^2+\gamma\mathbf{I}_m)^{-1}\mathbf{\Sigma}^2\mathbf{V}^\top$$

where the two factors of $\boldsymbol{\Sigma}$ have been consolidated to the right of the inverse term as diagonal matrices commute. This allows us to use $(\boldsymbol{\Sigma}^2 + \gamma\mathbf{I}_m)^{-1}\boldsymbol{\Sigma}^2 = \mathbf{I}_m - \gamma(\boldsymbol{\Sigma}^2 + \gamma\mathbf{I}_m)^{-1}$. Consequently,

$$\hat{\mathbf{H}}^{-1} = \frac{1}{\gamma}(\mathbf{I}_d - \mathbf{V}\mathbf{V}^\top) + \mathbf{V}(\boldsymbol{\Sigma}^2 + \gamma\mathbf{I}_m)^{-1}\mathbf{V}^\top.$$

First, obtaining $(\boldsymbol{\Sigma}^2 + \gamma\mathbf{I}_m)^{-1}$ is $O(m)$ time so it can be evaluated once and stored at an extra $O(m)$ space cost which is negligible compared to everything else we store. The matrix-vector product $\mathbf{V}^\top\mathbf{u}$ is $O(md)$ time for every $\mathbf{u}$. Hence, $\mathbf{V}\left[(\boldsymbol{\Sigma}^2 + \gamma\mathbf{I}_m)^{-1}(\mathbf{V}^\top\mathbf{u})\right]$ takes $O(md)$ time. Similarly, evaluating $1/\gamma(\mathbf{I}_d - \mathbf{V}\mathbf{V}^\top)\mathbf{u}$ can be separated so that it only costs $O(md)$. Obtaining the gradient vectors $\nabla f(\mathbf{x}^{(t)})$ always costs $O(nd)$ time. However, the above argument shows that applying $\hat{\mathbf{H}}^{-1}$ to vectors costs only $O((n+m)d)$ time to apply to gradients rather than the naïve $O(d^2)$ time. □

Remark 6.5.1. *Although the proof of Lemma 6.5.2 is simple, the key property we exploit from FD is that it is stored in* factored form*. This means that the Woodbury identity manipulation essentially comes for free. The same property is not true of randomised linear transforms which store* $\mathbf{B} = \mathbf{S}\mathbf{A}$ *explicitly and then need to evaluate an SVD at a cost of* $O(md\min(m,d))$ *before decomposing as for the above result.*

The above analysis leads us to the following theorem:

Theorem 6.5.1. *Let* $\mathbf{A} \in \mathbb{R}^{n\times d}, \mathbf{y} \in \mathbb{R}^n$ *denote the input to the ridge regression problem with a hyperparameter* $\gamma > 0$*. If the frequent directions sketch satisfies the requirements of Theorem 6.3.1 or Theorem 6.4.1, then the running time of Algorithm 9 for* T *iterations is* $O(Tnd/\varepsilon)$*.*

Proof. This immediately follows from the assumption on the sketch size from Theorems 6.3.1 and 6.4.1 which, for a constant ε needs a sketch size $m = O(1/\varepsilon)$. Hence the sketch time for Algorithm 1 is $O(nd/\varepsilon)$ which we only need once. Lemma 6.5.2 demontrates that one iteration costs $O((n + 1/\varepsilon)d)$ and we need T such iterations to achieve a relative error $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 \leq \varepsilon^T \|\mathbf{x}^\star\|_2$. Combining these two costs achieves the stated claim. □

This is nearly linear time because the T/ε constant is small in comparison to n and d. As ε is chosen to be a small but non-negligible constant (e.g., $1/2$) the term $1/\varepsilon$ is not too large. Although the number of iterations that we choose to perform is dependent on the desired accuracy of ε^T, one can show that if we need a solution of accuracy $\varepsilon^T = \varepsilon_\star$, then T grows slowly as $O(\log(1/\varepsilon_\star))$. This would result in a running time of $O(\varepsilon^{-1}\log(1/\varepsilon_\star)nd)$ for $\varepsilon_\star$ accuracy by using an ε-accurate (R)FD sketch of size $m = O(1/\varepsilon) \times d$.

6.6 Synthetic Data: Error & Sketch Size

Our aim in this experiment is to understand how the regularisation γ and sketch size interact in accordance with Theorems 6.3.1 and 6.4.1. We test the pessimistic setting when nothing is known a priori about the data spectrum. This corresponds to setting $k = 0$ in the FD bounds so Theorem 2.2.1 gives an error of $\alpha\Delta_k = \|\mathbf{A}\|_F^2/m$. For such k, Theorem 6.3.1 ensures convergence provided that $2\|\mathbf{A}\|_F^2/m < \gamma$. This understanding is necessary as the best k may not known in advance, so a user needs to understand how to set the regularisation γ for good model performance. Similarly, it is more likely that a user has a fixed space budget and wants to understand the range of γ for which their sketch yields a good model, rather than fixing a γ and then solving for the optimal projection dimension.

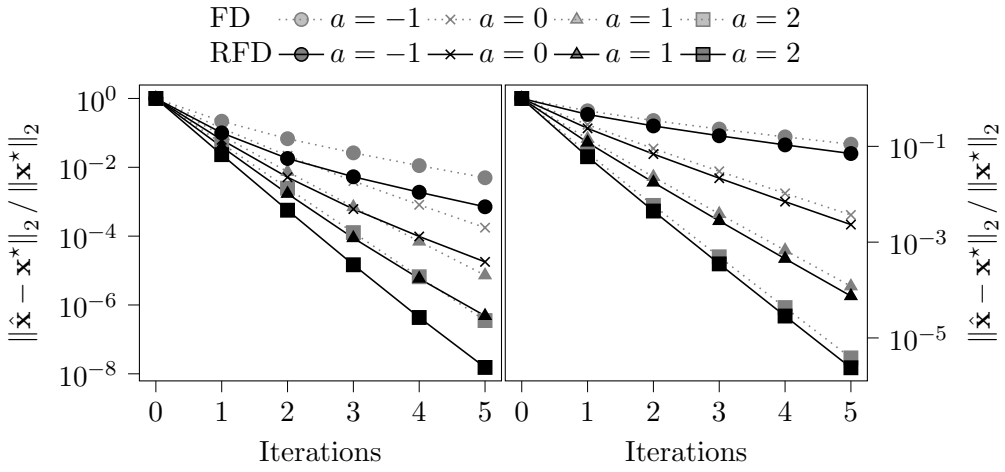


Figure 6.1: Relative error $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 / \|\mathbf{x}^\star\|_2$ plotted on a log scale against the number of iterations for various choices of $\gamma = 2^a\|\mathbf{A}\|_F^2/m$ with $a \in \{-1, 0, 1, 2\}$. Left hand plot has $R = 0.1d$ and right hand plot has $R = 0.5d$. Frequent Directions is plotted in dashed lines and Robust Frequent Directions is in solid lines. Note the different scales on left and right hand plots.

6.6.1 Experimental Setup

We repeat the following process over 10 independent draws of random synthetic data with the mean results being reported. We report the Euclidean error between the estimated weights $\hat{\mathbf{x}}$ after the iterations are completed and the optimal weights $\mathbf{x}^\star$, measured as $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 / \|\mathbf{x}^\star\|_2$. We generate a synthetic dataset (as described below) with $n = 2^{15}$ and $d = 2^{12}$ with an *effective rank* of $R = rd$ to ensure that most of the signal is captured on roughly the top $r\%$ of the directions and the remaining $(1 - r)\%$ being noise. To simulate approximately low-rank data, we choose $r = 0.1$ to enforce redundancy in the

feature space so that a summary with fewer than d rows can be maintained. For higher rank data, we also test $r = 0.5$ to see how the methods compare when there is less redundancy in the feature space. Algorithm 9 is implemented with both FD and RFD used as the sketch with a projection dimension of $m = 300$. We then fix a $\gamma_a = 2^a \|\mathbf{A}\|_F^2/m$ for $a \in \{-1, 0, 1, 2\}$ and perform 5 iterations using Section 6.5 for efficient updates in the gradient step.

This experiment is vital in order to verify our theory. One might ask how the accuracy behaves as the sketch dimension m varies. As γ and m are inversely related in Theorems 6.3.1 and 6.4.1, varying γ for a fixed m is equivalent to varying m for a fixed γ.

Ridge Regression Synthetic Data

We adopt the synthetic dataset found in Section 4 [SP21]. The *effective dimension* $R = \lfloor r \cdot d + 0.5 \rfloor$ is chosen for $r \in \{0.1, 0.5\}$. This is used to set the number of nonzero indices in the ground truth vector $\mathbf{x}_0$ and the number of standard deviations for the multivariate normal distribution used in generating $\mathbf{A}$. The first R components of $\mathbf{x}_0$ are sampled from a standard normal distribution and the remainder are 0; $\mathbf{x}_0$ is then normalised to unit length. The samples (rows) $\mathbf{A}_i$ are generated by a normal distribution with standard deviation $\exp(-(i-1)^2/R^2)$ for $i = 1 : n$. Finally, we rotate $\mathbf{A}$ by a discrete cosine transform. We sample noise a noise vector $\boldsymbol{\varepsilon}$ with $\boldsymbol{\varepsilon}_i \sim \mathcal{N}\left(0, 2^2\right)$ and set $\mathbf{y} = \mathbf{A}\mathbf{x}_0 + \boldsymbol{\varepsilon}$.

6.6.2 $R = 0.1d$

The results for $R = 0.1d$ are reported in the left-hand pane of Figure 6.1. In accordance with Theorems 6.3.1 and 6.4.1 we observe convergence when γ is sufficiently large. Recall that from Theorem 6.3.1, convergence is governed by the parameter $\beta = {}^{\alpha\Delta_k}/_{\gamma - \alpha\Delta_k}$ which in our experimental setup has $k = 0$ so $\alpha = 1/m$ and $\Delta_k = \|\mathbf{A}\|_F^2$. Hence, $\beta = {}^{\|\mathbf{A}\|_F^2}/_{m\gamma - \|\mathbf{A}\|_F^2}$ so increasing γ has the effect of reducing β. This explains why performance improves for larger γ. Thus, for a fixed projection dimension m, we expect faster convergence when γ is increased and this behavior is borne out in the plots. Denote $\gamma_a = 2^a \|\mathbf{A}\|_F^2/m$. Our theory for FD could be a little weak as we observe convergence (albeit slow for small γ_a) across all γ_a, even if it lies outside of the hypothesis range of Theorem 6.3.1.

When RFD is used, the general trends exhibit the same properties in that as γ_a increases, convergence speed is improved. For all $a \geq 0$, convergence is reported in line with the theory as Theorem 6.4.1 only ensures convergence provided that ${}^{\|\mathbf{A}\|_F^2}/_m < \gamma$ and clearly $\gamma_{-1} < {}^{\|\mathbf{A}\|_F^2}/_m$. Additionally, and as expected from the theory, we see constant factor improvement in the accuracy

Sketch Method	a			
	-1	0	1	2
FD	4.97×10^{-3}	1.76×10^{-4}	7.32×10^{-6}	3.52×10^{-7}
RFD	7.12×10^{-4}	1.79×10^{-5}	4.82×10^{-7}	1.53×10^{-8}
Error Ratio				
FD/RFD	6.97	9.86	15.2	23.0

Table 6.1: Relative error ($\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 / \|\mathbf{x}^\star\|_2$) comparison of iFDRR after 5 iterations on synthetic data with effective rank $R = 0.1d$ using Frequent Directions (FD) and Robust Frequent Directions (RFD). The error when using FD is consistently larger by a factor of roughly 7 to 23. The parameter a is used for regularisation $\gamma_a = 2^a \|\mathbf{A}\|_F^2 / m$.

$\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 / \|\mathbf{x}^\star\|_2$. This is difficult to see on the plots as they are plotted on a $\log_{10}$ scale, but inspecting the raw errors after 5 iterations we see that FD is consistently more than a factor of 5 less accurate than RFD for $R = 0.1d$, as described in Table 6.1. This disparity increases as a increases, reaching roughly 23 at $a = 2$.

6.6.3 $R = 0.5d$

In the right-hand pane of Figure 6.1 the effective rank R is increased from $0.1d$ to $0.5d$ and a mild change in behaviour is elicited. The error profile is still consistent with our theory as we observe convergence, but it is noticeably slower. For γ_2, a weaker relative error on the order of 10^{-5} after 5 iterations is obtained, compared to error of 10^{-8} when $R = 0.1d$. In general, each of the iterates is at least a factor of 10 less accurate at this effective dimension. This discrepancy can be put down to the sketch being less accurate as a smaller proportion of the directions are well-approximated, so not as much progress is made in each step. As before, we see a constant factor improvement in using RFD, but it is much milder than in the case $R = 0.1d$, as FD remains consistently only a factor of 1.5 less accurate than when RFD is used. The results are presented in Table 6.2.

6.6.4 Conclusions

We verified Theorems 6.3.1 and 6.4.1 showing that convergence is improved when either of γ or m is increased. The theory might be a little pessimistic as we observed convergence when the regularisation was outside of the hypothesis range of Theorem 6.3.1. Otherwise, expected behaviour was reported.

Sketch Method	a			
	-1	0	1	2
FD	1.11×10^{-1}	3.69×10^{-3}	1.20×10^{-4}	3.83×10^{-6}
RFD	7.08×10^{-2}	2.32×10^{-3}	7.49×10^{-5}	2.38×10^{-6}
Error Ratio				
FD/RFD	1.57	1.60	1.60	1.60

Table 6.2: Relative error ($\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2 / \|\mathbf{x}^\star\|_2$) comparison of iFDRR after 5 iterations on synthetic data with effective rank $R = 0.5d$ using Frequent Directions (FD) and Robust Frequent Directions (RFD). The error when using FD is consistently larger by a factor of roughly $1.5 \sim 1.6$. The parameter a is used for regularisation $\gamma_a = 2^a \|\mathbf{A}\|_F^2 / m$.

6.7 Real Data Experiments

We now explore how the results of Theorems 6.3.1 and 6.4.1 manifest on real data. Additionally, we implement the Iterative Hessian Sketch (IHS) of the previous section and a variant of IHS using only one sketch. The latter is to investigate a compromise between our algorithm, iFDRR Algorithm 9, which uses only one deterministic sketch, and the standard IHS algorithm which uses a new sketch for every iteration.

6.7.1 Experimental Setup

We illustrate results on two datasets (CoverType [AN07] & w8a [CL11]) over 5 independent trials. Each trial is completed with a uniformly randomly chosen subset of $n = 20,000$ data points. On the CoverType dataset, there are 54 columns over which we generate polynomial features of size $\binom{d+2}{2} \approx 1540$. We use the RBF sampler to approximate the Gaussian kernel with 2500 features for the w8a dataset with a variance parameter of 10^{-4}. This setup is to demonstrate that the method behaves similarly on different feature maps rather than arguing in favour of either feature map. Indeed, this is to be expected as we have made no assumptions over the feature space.

For all of these experiments, we take $\mathbf{A}$ to be the featurised dataset, $\gamma = \|\mathbf{A}\|_F^2/m$ and $m = 300$. Note that this is more optimistic than the set of permissible γ when using FD Theorem 6.3.1, but is acceptable for RFD (Theorem 6.4.1). Nonetheless, we find that using this value of γ does not inhibit the performance of Algorithm 9 with FD.

Competing Methods

We use 3 methods to approximate $\mathbf{H}_\gamma$:

1. Deterministic: our proposal of using Algorithm 9 with (R)FD;

2. Iterative (Single) Hessian sketch [LP19]: generates a single random sketch to perform all of the iterations from line 5 of Algorithm 9;

3. Iterative (Multi) Hessian sketch [PW16]: generates a fresh random sketch to perform all of the iterations from line 5 of Algorithm 9.

The multiple sketch version of Iterative Hessian Sketch is the same as introduced in Section 5.3; we implement it with the CountSketch and SRHT. Additionally, we also test this setting with the SJLT which, recall from Section 5.1.1 provides a compromise between the sparsity of the CountSketch and the optimal embedding dimension of the SRHT. We will refer to all methods tested in this regime as IHS:`sketch` where `sketch` is a random projection. The SJLT is implemented with 5 nonzeros per column to show a clear tradeoff between the aforementioned sketches rather than being chosen optimally. Notwithstanding computational issues, iterates from the IHS method with a fresh sketch are of the form

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - (\hat{\mathbf{H}}^{(t)})^{-1}\nabla f(\mathbf{x}^{(t)}).$$

The primary interst of this experimental section is to validate the performance of the (robust) Frequent Directions algorithms in the iterative setting. Recall that at a high level, Algorithm 9 first sketches the data and secondly performs all iterations using a single sketch. Therefore, we perform the same process but with a CountSketch and an SRHT. All of these methods are 'single shot' iterative methods as we only use one sketch: Theorems 6.3.1 and 6.4.1 govern convergence for the (R)FD methods, but the picture isn't as clear for single-shot iterative scheme and is included only for comparison. It can be shown that for a Gaussian sketch, if one tunes the step size correctly then a single sketch can be used [LP19], however, we would prefer to avoid parameter tuning, as discussed when introducing gradient descent in Section 2.3.2. Hence, we only use unit-step size for the randomised, single-sketch version of IHS. Moreover, [LP19] only includes analysis for the unscalable Gaussian sketch.

Nevertheless, as FD only has meaningful guarantees when $m < d$, we will test all methods at the same projection dimension of m. As this setup is useful when data is approximately low rank, we still expect the randomised methods to perform well *if m is sufficiently large*. This could be formalised by analysing the results of [PW16, CYD18] with the stable rank (and related statistical dimension Definition 6.1.1) results of [CNW16]. However, such guarantees are implicit in those works so we will assume that using these randomised methods may converge *if* the sketch size and/or regularisation is appropriately set. In contrast to the previous experiments of Section 5.4 we will *always* maintain the same sketch dimension to ensure a straight comparison between deterministic

and randomised sketches in the same framework at the same dimension.

Experimental Summary

In summary, we test CountSketch, SJLT, SRHT in the standard *multiple* sketch version of IHS. In the single sketch version of IHS (akin to our Algorithm 9) we use random projections CountSketch, SRHT and deterministic methods Robust Frequent Directions (RFD) and Frequent Directions (FD). All methods use a projection dimension of $m = 300$. The deterministic methods use $T = 15$ iterations and the randomised methods use $T = 45$ iterations, irrespective of the single/multiple version of IHS employed. Experiments were run 5 times with the mean results being reported in the plots.

Computational Issues Regarding Updates

As we will be interested in the time that all of these methods take, we need to establish how best to separate the build and update costs. Recall that a sketch $\mathbf{B} \in \mathbb{R}^{m \times d}$ of the data $\mathbf{A} \in \mathbb{R}^{n \times d}$ is evaluated. If the sketch method is either of robust or standard Frequent Directions, then $\mathbf{B}$ is stored in its SVD form. Otherwise, if the sketch method is a random projection then $\mathbf{B} = \mathbf{S}\mathbf{A}$. Algebraically, we estimate the Hessian $\mathbf{H} = \mathbf{A}^\top \mathbf{A} + \gamma \mathbf{I}_d$ through an approximation of the covariance matrix $\mathbf{A}^\top \mathbf{A}$ by $\mathbf{B}^\top \mathbf{B}$, albeit the latter is not materialised explicitly. Recall that the iterations for all methods are of the form:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \hat{\mathbf{H}}_*^{-1} \nabla f(\mathbf{x}^{(t)})$$

which can be implemented more stably by the two updates:

$$\hat{\mathbf{H}}_* \mathbf{z} = -\nabla f(\mathbf{x}^{(t)}) \tag{6.20}$$

$$\mathbf{x}^{(t+1)} = \mathbf{z} + \mathbf{x}^{(t)}. \tag{6.21}$$

Note that because we will project to $m < d$ directions, (6.20) can be efficiently solved by the same method of Section 6.5 if the sketch is (R)FD *or* a random projection. For single-sketch methods, we evaluate a single approximation $\hat{\mathbf{H}}_* = \hat{\mathbf{H}}$ while for multi-sketch IHS, we need $\hat{\mathbf{H}}_* = \mathbf{H}^{(t)}$ for every iteration t. Thus, there is a slightly different time cost to measure in each setting.

1. Single sketch
   (a) Obtain the sketch $\mathbf{B}$ *and* compute its SVD.
   (b) Use the SVD of $\mathbf{B}$ for every iterative update/solve from (6.20) as in Section 6.5.
2. Multi sketch

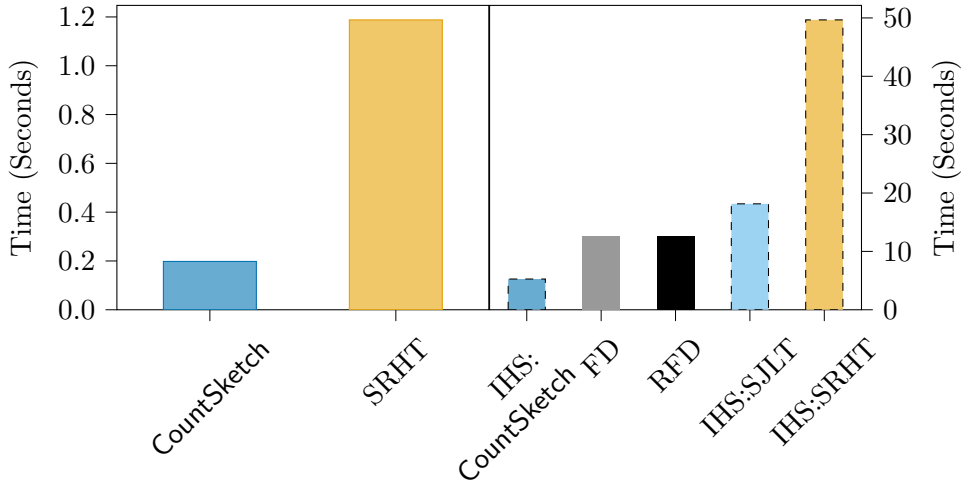


Figure 6.2: *Total* sketch build time: CoverType. Black dashes indicate multiple rounds of sketching, rather than just one.

(a) Obtain the sketch $\mathbf{B}$

(b) For every iteration, evaluate the SVD of $\mathbf{B}$ and use it to solve (6.20).

We can clearly see a distinction that needs to be treated carefully. The SVD is computed for every iteration of multi-sketch IHS at a cost of $O(m^2d)$ per update. This could aggregate substantially as $O(Tm^2d)$ for T steps and adversely affect the wall-clock time. Although the sketching time of FD can be larger than the sparse methods, it has a linear iteration time (Lemma 6.5.2), rather than quadratic in m. Thus we need to understand the tradeoff between an expensive initial sketch with cheap linear iterations versus sketches that are fast to obtain but can have iteration times growing with $O(m^2d)$.

6.7.2 Experiment 1: Sketch Build Time. Figures 6.2 and 6.3

We measure how long it takes to build or initialise the sketches used for the iterative update (6.20). In the single sketch models, this is the time to build *one* sketch $\mathbf{S}$ *and* perform an SVD of that sketch. Note that the SVD is free for (R)FD as it is stored in factored form, whereas for the randomised methods it must be computed after the sketching operation. Since $m < d$, the SVD costs $O(m^2d)$. The time costs to build a sketch of size m are $O(ndm)$ for (R)FD, $O(nd \log n + m^2d)$ for the SRHT and $O(\mathsf{nnz}(\mathbf{A}) + m^2d)$ for a CountSketch. Thus the build time for the IHS methods should be T times the cost of building one random sketch (without the SVD cost).

In Figure 6.2 we present the findings of the CoverType dataset. In summary, we find that the CountSketch random projection is fastest, taking about 0.2 seconds to return a single sketch and its SVD. Next fastest is the single SRHT method needing roughly 1.2 seconds. We find that obtaining $T = 45$

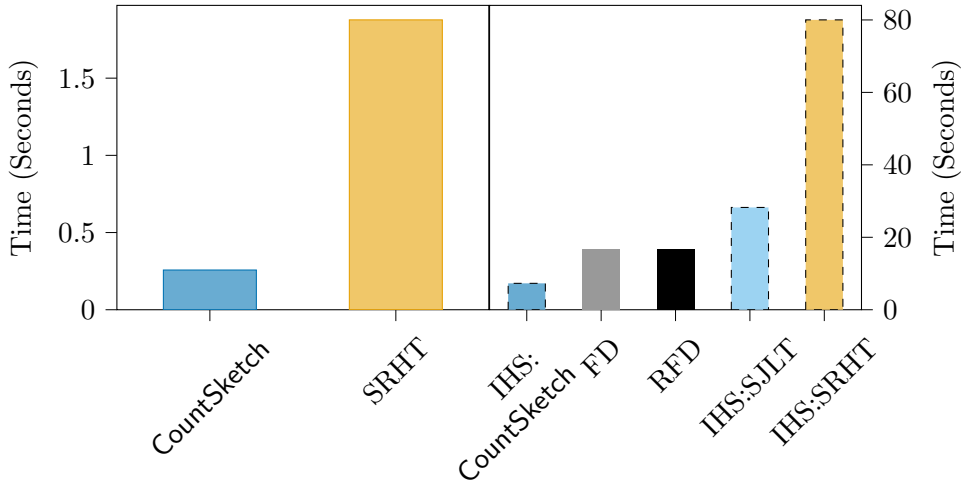


Figure 6.3: *Total* sketch build time: w8a. Black dashes indicate multiple rounds of sketching, rather than just one.

CountSketches $\mathbf{S}^{(t)}\mathbf{A}$ took about 5 seconds. This is faster than evaluating a single deterministic sketch of FD or RFD type, both needing on average roughly 12 seconds. On the other hand, the IHS:SJLT and IHS:SRHT were slowest, needing about 20 and 50 seconds to obtain all 45 sketches on average. From this perspective, our introduction of CountSketch into the (multi-shot) IHS model from Section 5.3 again appears vindicated, assuming there is no significant drop in accuracy.

We find a similar conclusion for the w8a presented in Figure 6.3. Namely, the single random projections are fastest, followed by the IHS:CountSketch. Next are both deterministic methods which take longer than IHS:CountSketch but less time than IHS:SJLT. Both multi sketch methods take a constant factor longer than their single sketch counterparts, as expected. In all settings we see that the (R)FD methods take a strong constant factor longer than the single-sketch randomised methods to sketch the data. In particular, the time cost is one the order of tens of seconds for (R)FD rather than the random projections which takes on the order of seconds. The time difference between deterministic methods and the multi-sketch models is less severe and these methods take roughly comparable time, with the exception of IHS:SRHT which is comfortably the slowest.

6.7.3 Experiment 2: Update Time. Figures 6.4 and 6.5

We report the times taken to evaluate the update step in our algorithm Line 5, $\mathbf{x}^{(t)} = \mathbf{x}^{(t)} - \hat{\mathbf{H}}^{-1}\nabla f(\mathbf{x}^{(t)})$ using all of the methods. When using a single sketch our expectation from Lemma 6.5.2 and Theorem 6.5.1 is that the iteration time cost using iFDRR with either (R)FD or the two random sketches should scale approximately linearly in the input parameters $O(ndm)$ as per Lemma 6.5.1.

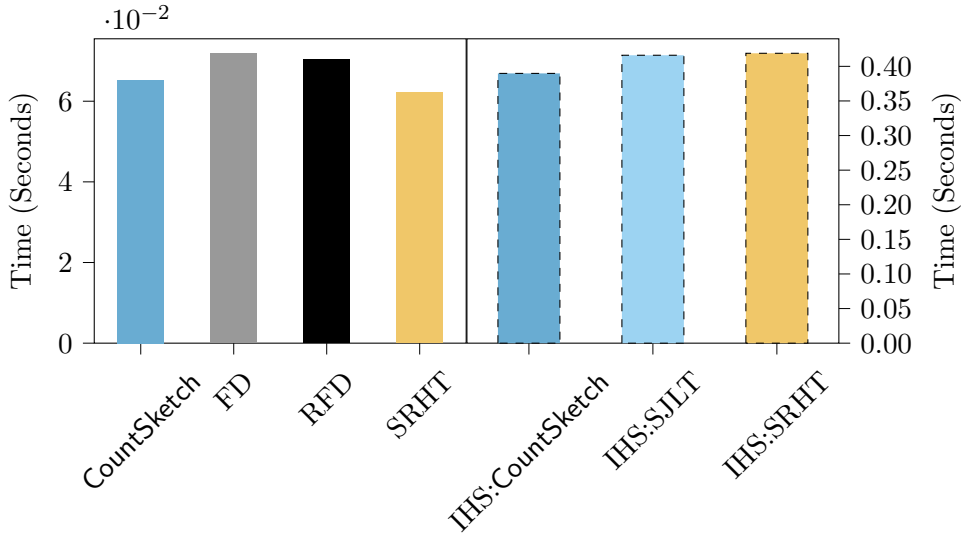


Figure 6.4: Update time: CoverType. Note the different y axis scales.

Since the FD sketch is stored in factored form, we do not perform any expensive matrix operations after obtaining the sketch. The inverse is easily computed and hence the updates can be performed in close to linear time. On the other hand, the updates when using the multiple sketch IHS needs a new SVD after obtaining the sketch so the time for the updates should scale as $O(m^2d+ndm)$. Figures 6.4 and 6.5 do not include the sketching time, only the time take to evaluate (6.20) efficiently and then the update (6.21).

Note that even on this small-scale data we can see separation between the single sketch methods and the multi sketch methods as shown in Figure 6.4. The single sketches take on average about 0.06 seconds on average, which is irrespective of the sketch employed, as expected. Meanwhile, the multiple sketch framework takes about 0.4 seconds, again almost independent of the type of sketch used. Thus, one can see that due to needing a fresh sketch at every stage, the increased cost of needing an SVD at every iteration could become prohibitive. This behaviour is consistent on the w8a dataset as shown in Figure 6.5 in that we see the expected separation between single and multi sketch frameworks. In both datasets, there appears to be roughly a factor 6 more time needed per iteration to evaluate the update $\mathbf{x}^{(t+1)}$ from $\mathbf{x}^{(t)}$ which is explained by the extra SVD step when using a fresh sketch for every iteration.

6.7.4 Experiment 3: Error vs iterations and time. Figures 6.6 and 6.7

Recall that $T = 15$ iterations were executed using the deterministic sketches of FD and RFD whilst $T = 45$ iterations were completed for all randomised methods. The reason for this is clearly illustrated in Figures 6.6 and 6.7 where

there is clear separation between the deterministic sketches and the randomised counterparts.

Experiment 3a: Error vs iterations (left hand panes of Figures 6.6 and 6.7)

In both plots, the first point to take away is that both randomised single sketch methods diverge at a sketch dimension of $m = 300$. Thus, although the speed of using a single random sketch is attractive, it is not accurate compared to all other methods. We see that the accuracy of the deterministic methods is very good; a strong relative error of 10^{-13} is achieved in at most 15 iterations for both CoverType and w8a. This is roughly 7 orders of magnitude more accurate than all multi sketch randomised approaches after 15 iterations for CoverType and almost 10 orders of magnitude more accurate on the w8a. The multi-sketch setups converge very slowly towards the optimum in comparison to the deterministic methods, as illustrated by the flatter gradient of their error profiles. Consistent with Section 6.6 and Theorems 6.3.1 and 6.4.1, the performance of RFD is slightly better than using FD, but this difference is marginal compared to the advantage of using the deterministic sketches over any of the randomised methods. Nonetheless, the deterministic methods use roughly 3 times fewer iterations to approach high accuracy compared to the multi sketch IHS methods.

Experiment 3b: Error vs time (right hand panes of Figures 6.6 and 6.7)

The real win of using FD and RFD is presented in the error-time profiles. From Figures 6.2 and 6.3 we saw that the build time of FD and RFD was substantially higher than the randomised single-sketch methods, however they had poor accuracy. On the other hand, the build time for a single deterministic sketch was comparable (or better) than that to obtain T random sketches for IHS:`sketch`. Coupled with the improved update times of Figures 6.4 and 6.5, we need to understand whether the extra time invested in generating the sketch was worthwhile when comparing error versus time.

The time required for convergence of FD is a factor of at least 2 less than IHS:SJLT &, IHS:SRHT on both CoverType and w8a. Although the deterministic methods are slightly slower than IHS:CountSketch, they remain within a factor of 1.5 of the overall time taken for IHS:CountSketch to achieve the same accuracy: roughly 9 vs. 12 seconds on CoverType and 12 vs 17 on w8a. This is done using a fraction of iterations as seen in the previous analysis.

The error profile using (R)FD can be attributed to a much more accurate sketch: as seen in Figures 6.6 and 6.7, on both datasets, essentially all of the

time cost has been invested in obtaining a high quality deterministic sketch. This is represented by the nearly horizontal grey and black lines with no markers, indicating that no steps have been taken. However, once the sketch is obtained, descent to the optimal solution is rapid both in time because of the fast updates seen in Experiment 2, but also because only a small number of steps after sketching are necessary, indicated by the near vertical lines. The same property is not true of the randomised methods which obtain a weaker sketch more quickly, but they make far less progress per step. In comparison, the shallow initial gradient of all IHS approaches only steepens after many iterations. Despite the speed of IHS:CountSketch, it needs nearly three times many iteration rounds to achieve the same accuracy in only a marginal time improvement over (R)FD. In large-scale systems, the extra cost of communication and distribution could further inflate the cost of extra iteration rounds.

These error-iteration-time plots are consistent with our theory in that faster convergence is observed by using RFD vs. using FD. We find that the per-iteration error decrease is uniformly better when using deterministic sketches compared to using randomised sketches. From this perspective, the deterministic methods look favourable as they achieve high accuracy in a small number of iterations and are less than a factor of 2 slower than the fastest IHS method.

6.7.5 Summary

The main point to takeaway from our experiments is that performance is dependent upon the compromises that users have to make in their computing architecture. Given an unconstrained computation model, one would always use an SVD, but in the realistic setting when a user is heavily constrained by space or time, this is not possible. When adding communication/number of iteration rounds into the constraint space, the picture becomes even less clear as the user needs to minimise across all three of these parameters while retaining good performance.

Viewing our results in the context of these three criteria, we can see that both FD and RFD are performant and scalable sketches that return highly accurate weight estimates in a timely fashion. While such a sketch is expensive to build, it is extremely accurate for estimating the optimal weights, so only a small number of gradient steps are necessary. The expense of the Newton step has instead been shifted onto efficiently estimating the action of $\hat{\mathbf{H}}^{-1}$. The small number of gradient steps shows that if this were in a distributed setting, then only a small number of communication rounds would be necessary. This is not true of the randomised methods which generate cheaper but less effective

sketches and need significantly more communication/gradient steps or need a new sketch every round so have a higher iteration complexity. The weaker estimation of randomised sketches can result in needing three times as many rounds of communication to achieve comparable accuracy.

6.8 Conclusion

We studied higher dimensional ridge regression problems and saw that evaluating a new sketch for every iteration could be prohibitively expensive. We adapted the IHS approach for use with a single sketch. This work is the first analysis of Frequent Directions with Newton update model to estimate the regression weights in the multi-round optimisation model. We have shown that despite (approximate) Newton updates typically being expensive, the special structure of Frequent Directions enables an efficient algorithm. The Frequent Directions sketch takes substantially longer to generate than a random projection of the data but this does not inhibit overall performance. Using the robust variant of Frequent Directions can enhance performance both in practice and in theory. Our experimental section vindicated this approach as the sketch is significantly more accurate than the competing randomised methods. Descent towards $\mathbf{x}^\star$ is faster in terms of number of iterations by a factor of at least 3 when using (R)FD. Consequently, our approach is competitive with the fastest method IHS:CountSketch, while using many fewer rounds of iterations. This firmly places iFDRR as a scalable Newton Method which should be considered for use in large n and large d regimes.

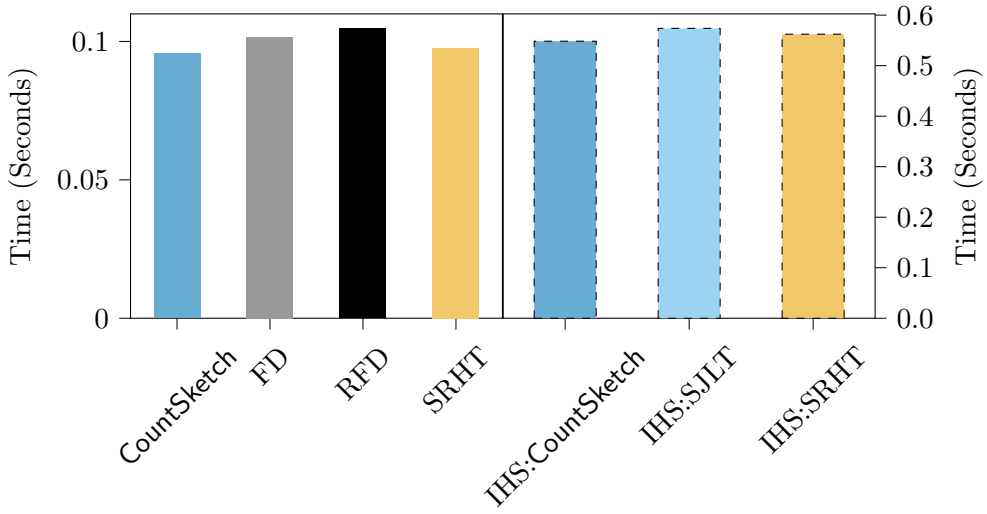


Figure 6.5: Update time: w8a. Note the different y axis scales.

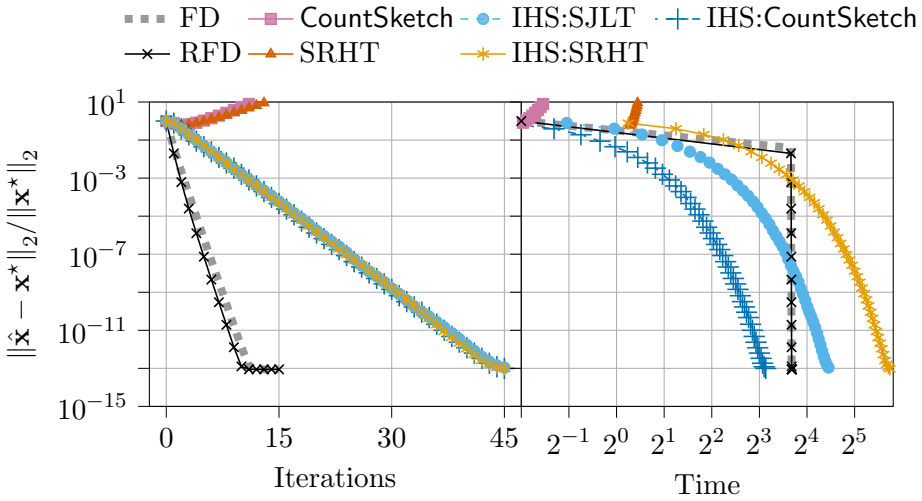


Figure 6.6: Error profile over time for CoverType

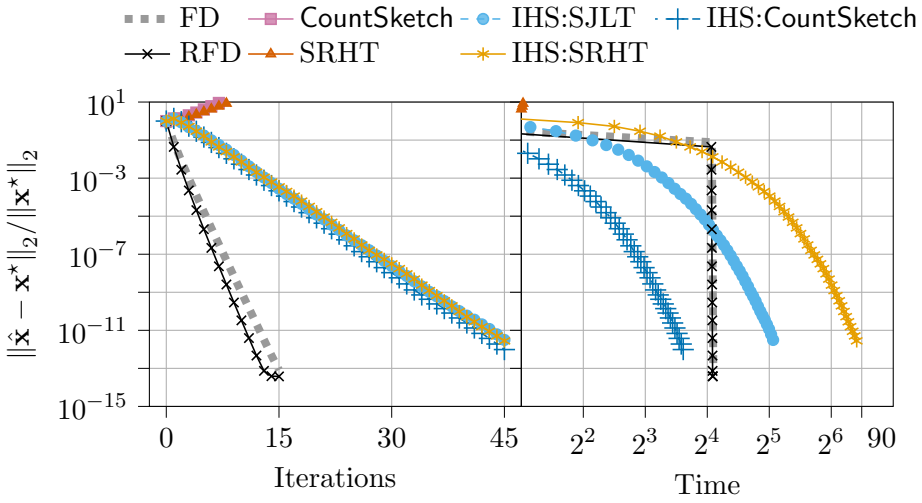


Figure 6.7: Error profile over time for w8a

Chapter 7

Discussion and Conclusion

7.1 Results Summary

The central theme of this thesis has been to study how one can find and use summaries of large tabular datasets. Tabular datasets may represent arbitrary arrays as used in chapter 3, or matrices over $\mathbb{R}^{n\times d}$ for linear algebraic problems as in chapters 4 - 6. These summaries are small space representations of the data that estimate certain characteristics of interest. In approaching this task, we have seen results split into three categories regarding the efficacy of such summaries. We briefly highlight these unifying themes and then discuss some future directions in which to take the work.

1. **Scalability**: each chapter has been motivated by "exact" or "optimal" analysis being too expensive in either space usage or running time. This necessitated finding small space summaries that can be obtained efficiently, considering both time and space usage.

2. **Accuracy**: each of the summaries we have developed or used has an associated guarantee on how well it estimates a property of interest.

3. **Hardness**: not all problems admit summaries that can be efficiently obtained *and* have a guarantee of $1 \pm \varepsilon$ relative error *in small space*. In chapters 3-5, we have shown that finding (or using) small space summaries that achieve overly optimistic error guarantees would result in expensive space consumption. Such results typically forced us to adapt the algorithmic guarantee that we sought or the type of computational model to study. In Chapters 3 and 4, these hardness reductions were shown via communication complexity by instantiating a clever hard instance over matrices derived from combintorial codes. However, in Chapter 5, we showed a spectral property of the CountSketch and appealed to an information theory argument of [PW16].

Chapter 3

In Chapter 3 the tabular data was an $n \times d$ binary array over which we wanted to estimate frequency-based problems restricted to only a subset of columns. The difficulty in this chapter was that the query set of columns was revealed after the data observation phase. An exact solution might calculate frequency statistics for each of the 2^d subsets of $[d]$. Realising the large length 2^d frequency vector is not scalable.

Our starting point for this work was to show that many of these projected frequency estimation problems are *hard* to estimate using small space. Returning an estimate within constant factors of the correct answer was shown to require exponential space of $2^{\Omega(d)}$. On the other hand, we relaxed our *accuracy* requirement from an error guarantee within constant factors to one within some small exponential factors related to the input dimensionality, the summary size, and the specific frequency estimation problem. The algorithmic idea that we used was to maintain a "net" of summaries for a much smaller set of column queries than the exponentially large set of all possible column queries; specifically, maintain $\mathcal{N} \subset \mathcal{P}([d])$ with $|\mathcal{N}| < |\mathcal{P}([d])|$. Although these guarantees still use space exponential in d, we showed that the net $\mathcal{N}$ can be smaller than maintaining a sketch for every possible query column subset and thus is more *scalable*. How much smaller the $\mathcal{N}$ is than $\mathcal{P}([d])$ is controlled through a parameter that also governs the error bounds in our estimation, thus admitting a space-approximation tradeoff.

Chapter 4

We investigated how to find deterministic summaries in arbitrary ℓ_p for $p \geq 1$. To obtain such a summary, one approach would be to compute a "global" well-conditioned basis $\mathbf{U} \in \mathbb{R}^{n \times d}$ for the input. The summary would contain all rows with high-leverage in $\mathbf{U}$. However, this is not scalable as finding $\mathbf{U}$ costs $O(nd^5)$ time and $\mathbf{U}$ cannot be stored in full for large n.

Rather than computing a single, large, well-conditioned basis, we proposed evaluating a sequence of "local" well-conditioned bases from which we pruned the important high-leverage rows of data. The pruned set of rows were repeatedly merged to collate the locally important information and then reduced to ensure the space usage remained bounded by poly (d). This algorithm is a more *scalable* approach as it operates on a data stream and uses space independent of the input size. We used the summary of high-leverage rows to obtain an additive-error guarantee for approximate ℓ_∞ regression. On the other hand, we also showed a negative result through a *hard instance* using a similar but simpler construction to those from Chapter 3. Specifically, improving the additive error guarantee to relative error is difficult, needing a summary to

grow with the size of the input, rather than having size independent of the data's size.

Chapter 5

In this chapter we separated analyses of the so-called "sketch-and-solve" and "Iterative Hessian Sketch" (IHS) frameworks for solving regression using sparse random projections such as the Sparse Johnson-Lindenstrauss Transform (SJLT) and CountSketch. Exact solutions require an SVD of the input matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ so have a $O(nd^2)$ time cost which is not scalable. We showed spectral and structural properties of the CountSketch to invoke an argument of Pilanci and Wainwright [PW16] showing that estimating the optimal weights in a single-pass yields high-error.

Nonetheless, we adopted the multi-round optimisation model, which is still typically seen in machine learning, by compromising the need of a single-pass algorithm for a multi-pass algorithm. In this setting, we showed that both the CountSketch and SJLT can return weights between 2-20 times much more quickly than dense random projections.

Chapter 6

For the particular case of ridge regression, we showed that Frequent Directions (FD) can be used in the multi-round optimisation model. FD has an increased time overhead compared to sparse projections but this deficiency should not immediately rule out its use. We gave high-quality error guarantees and used only a single sketch rather than a new one for each iteration. The examples we tested suggest that the larger time investment to obtain FD is worthwhile. Our approach used one third of the iteration rounds (or passes) and has no significant change in overall wall clock running time for the same accuracy. Thus, the iterative sketching with a FD can be considered a *scalable* and *accurate* method for returning regression weights.

7.2 Future Work

There are various directions in which future work could build upon the ideas of this thesis. These vary depending on the type of results we have given, as well as the computation models that we have studied.

Chapter 3

A good starting point for building upon this section would be to modify the computation model to allow for more positive results. We have assumed that the input array is read once and never seen again. Perhaps this could be

relaxed to reading the entire data once, and then accessing *a small portion* of the data once more. The aim here would be to build a summary during the initial observation and then use the information garnered during the next phase to return a more accurate estimate to the query. Characterising this requires care; if one can read the entire dataset again, this resembles a two-pass streaming model which would be trivial if the columns are revealed after the initial phase. Hence, the extra portion of data to be read should be considered a quantity to be minimised.

An alternative direction would be the study of linear algebraic functions over column projections. Here, perhaps more easily than for projected frequency analysis, interactions between pairs of columns can be better understood through the covariance matrix structure $\mathbf{e}_j^\top \mathbf{A}^\top \mathbf{A}\mathbf{e}_i = \langle \mathbf{A}\mathbf{e}_j, \mathbf{A}\mathbf{e}_i \rangle$. This relationship might permit a useful starting point. For instance, for any $S \subseteq [d]$, the covariance matrix on $\mathbf{A}$ restricted to S, denoted $\mathbf{A}^S$, is $\mathbf{X} = \mathbf{A}^{S^\top}\mathbf{A}^S$. If $\mathbf{A}^\top\mathbf{A}$ is stored in $O(d^2)$ space, then it is possible to represent $\mathbf{X}$ through all pairs of inner products $\mathbf{e}_j^\top \mathbf{A}^\top \mathbf{A}\mathbf{e}_i$ for $i, j \in S$. On the other hand, while applications for projected frequency analysis were quite natural, coming up with useful applications of projected linear algebraic problems is less clear. For example, what would be the purpose of a projected subspace embedding or projected leverage scores?

Chapter 4

A shortcoming of our approach in Chapter 4 is that we can find the rows of high leverage but we cannot use this to reason about a form of subspace embedding. This hinders the applicability of the high-leverage row summary beyond ℓ_∞-regression. Future work could study whether this summary can be used to find an 'additive' subspace embedding in a similar fashion to Frequent Directions. Specifically, for a summary $\mathbf{B} \in \mathbb{R}^{m \times d}$ of input $\mathbf{A} \in \mathbb{R}^{n \times d}$ can the summary be used to find:

$$\|\mathbf{A}\mathbf{x}\|_p - t\,\|\mathbf{x}\|_p \leq \|\mathbf{B}\mathbf{x}\|_p \leq \|\mathbf{A}\mathbf{x}\|_p$$

for a constant t that depends on the leverage score threshold and the size of the sketch.

Such a result might need a better understanding of the so-called "Lewis weights" of [CP15]. These weights resemble ℓ_p leverage scores of an augmented input $\mathbf{WA}$ through a reweighting matrix $\mathbf{W}$. Although [CP15] achieve an input sparsity time algorithm for ℓ_p subspace embeddings, finding the correct $\mathbf{W}$ requires $\log n$ accesses to the dataset. Hence, although their algorithm is input sparsity time, it is not single-pass. Future study should investigate if these weights can be approximated on streaming data.

Chapter 5

Recall that the *Gaussian Width* of a convex set $\mathcal{K}$ is denoted $\mathcal{W}_{\mathcal{K}}$. Preserving inner product and norms up to ε-error on $\mathcal{K}$ with dense Gaussian and Subsampled Randomised Hadamard Transform (SRHT) random projections can be done using $m = \tilde{O}(\mathcal{W}_{\mathcal{K}}^2/\varepsilon^2)$ projections [PW15, PW16]. However, no such result is known for the CountSketch. Due to [BDN15], the same property is known for Sparse Johnson-Lindenstrauss Transform, modulo some extra log factors in the $\tilde{O}(\cdot)$ term. This contribution is highly non-trivial and relies on the use of $s = \Omega(1/\varepsilon)$ nonzeros per column to ensure that rows with high-leverage do not often collide. Given the empirical advantages we saw when using the CountSketch for least squares problems and convex constrained variants in [CD19, LLW21], it would be helpful to better understand this behaviour. If the largest leverage score is at most $1/d$, then [BDN15] showed that the projection dimension for a CountSketch can be chosen as $O(d \log d/\varepsilon^2)$ provided that $\mathcal{K}$ is a linear subspace. This property is exploited in [LLW21] for iterative sketching. However, they use a "learned" CountSketch, which is slightly different from the "standard" CountSketch we have studied and no results are given relating the projection dimension to the Gaussian width of the constraint set.

Another direction would be to study "weak embeddings". We found a subspace embedding that preserves all d directions for every iteration. Can anything be said about solution estimation if only a $1 - \alpha$ fraction of the directions are preserved in every iteration?

Chapter 6

Further work on better exploiting Frequent Directions for fast and small space iterative solvers could be promising. In our experiments, we have shown that our algorithm can outperform randomised variants, achieving convergence in better wall clock time and fewer iterations/passes/rounds of communication. Although the sketch is expensive to obtain it is accurate enough to enable rapid descent to the optimum once the gradient steps begin to take place. There are two clear practical weaknesses to our results: the first is that ridge regression, while a useful foundational tool is not a modern machine learning so has been superceded in practice by those such as *kernel ridge regression*. Our approach may work in this setting but would not be the most scalable solution as an explicit feature map would need to be applied to every sample. Future work should consider which problems would benefit from such approximate-Newton methods with Frequent Directions. A good starting point might be ℓ_2 penalised logistic regression due to its similarity to ridge regression, followed by generalised linear models as in [PW17]. The second weakness is that our result gives a bound on the solution error $\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2$ in contrast to the quantity

$\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_{\mathbf{A}}$ of Iterative Hessian Sketch. We cannot obtain this guarantee as we do not have relative error control of inner products as in (5.20) under the transform $\mathbf{u} \mapsto \mathbf{Au}$. More precisely, random projections achieve

$$|\langle \mathbf{SAu}, \mathbf{SAv} \rangle - \mathbf{Au}, \mathbf{Av} \rangle| \leq \varepsilon \|\mathbf{Au}\|_2 \|\mathbf{Av}\|_2$$

while a comparable statement using the non-linear Frequent Directions sketch is not known.

Although we argued that the multi-pass computation model is widely used in machine learning, single-pass algorithms certainly have their place. With this in mind, one could try to extend the results of [SP21] to *matrix ridge regression* using deterministic sketches such as the *Co-Occurring Directions* of [MMG17]. Whereas FD seeks only to estimate $\mathbf{A}^\top \mathbf{A}$, Co-Occurring directions uses the same idea but replaces the SVD of $\mathbf{A}$ with QR decompositions of $\mathbf{A}$ and a *matrix* $\mathbf{M}$ to estimate[1] $\mathbf{A}^\top \mathbf{M}$. By applying this idea to ridge regression, it might be possible to return the deterministic sketches $\mathbf{B_A}, \mathbf{B_Y}$ of the data $\mathbf{A}$ and target variables $\mathbf{Y}$, resulting in a matrix ridge regression estimate of

$$\hat{\mathbf{X}} = \left(\mathbf{B_A}^\top \mathbf{B_A} + \gamma \mathbf{I}_d \right)^{-1} \mathbf{B_A}^\top \mathbf{B_Y}.$$

This is to be compared with the ridge estimate of [SP21] using $\mathbf{B} = \mathsf{FD}(\mathbf{A})$:

$$\hat{\mathbf{x}} = \left(\mathbf{B}^\top \mathbf{B} + \gamma \mathbf{I}_d \right)^{-1} \mathbf{A}^\top \mathbf{y}.$$

If this is possible, then it would be interesting to understand whether the following analogy is true for statistical guarantees (such as bias and variance [WGM17]):

Ridge regression with co-occurring directions
behaves as "classical sketch-and-solve"

in the same way that [SP21, Lemma 3] showed that

Ridge regression with Frequent Directions
behaves as "Hessian sketch-and-solve."

Another interesting direction would be to investigate whether approximate gradient steps can be taken with every batch update used for the FD sketch rather than needing to traverse the entire data once again. This is somewhat reminiscent of the experimental setup of [SP21], but is also close to a stochastic gradient-type approach that is common in machine learning. The hope here would be that we sacrifice on extremely high precision in estimating the optimal

[1] when $\mathbf{M} = \mathbf{A}$, the guarantee of [MMG17] is equivalent to the Frequent Directions guarantee.

weights for something that is more accurate than a single pass sketch-and-solve approach, but does not need repeated access to the data as our model has assumed.

Bibliography

[AC06] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, page 557–563, New York, NY, USA, 2006. Association for Computing Machinery.

[ACH+12] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '12, page 23–34, New York, NY, USA, 2012. Association for Computing Machinery.

[ACH+13] Pankaj K Agarwal, Graham Cormode, Zengfeng Huang, Jeff M Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Transactions on Database Systems (TODS)*, 38(4):1–28, 2013.

[ACW17] Haim Avron, Kenneth L. Clarkson, and David P. Woodruff. Sharper Bounds for Regularized Data Fitting. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh S. Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*, volume 81 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:22, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[AKLT16] Sepehr Assadi, Sanjeev Khanna, Yang Li, and Val Tannen. Algorithms for Provisioning Queries and Analytics. In *International Conference on Database Theory*, pages 18:1–18:18, 2016.

[Alm21] Virginia Vassilevska Alman, Josh; Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the thirty-second annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 2021.

[AM15] Ahmed Alaoui and Michael W Mahoney. Fast randomized kernel ridge regression with statistical guarantees. In C. Cortes,

N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[AMS99a] N. Alon, Y. Matias, and M. Szegedy. The Space Complexity of Approximating the Frequency Moments. *JCSS: Journal of Computer and System Sciences*, 58:137–147, 1999.

[AMS99b] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.

[AMT10] Haim Avron, Petar Maymounkov, and Sivan Toledo. Blendenpik: Supercharging lapack's least-squares solver. *SIAM Journal on Scientific Computing*, 32(3):1217–1236, 2010.

[AN07] Arthur Asuncion and David Newman. Uci machine learning repository, 2007.

[BBV04] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[BCI+17] Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P Woodruff. BPTree: An ℓ_2 heavy hitters algorithm sing constant memory. In *Proceedings of Principles of Database Systems*, pages 361–376. ACM, 2017.

[BDN15] Jean Bourgain, Sjoerd Dirksen, and Jelani Nelson. Toward a unified theory of sparse dimensionality reduction in euclidean space. *Geometric and Functional Analysis*, 25(4):1009–1088, 2015.

[BGD13] Austin R Benson, David F Gleich, and James Demmel. Direct qr factorizations for tall-and-skinny matrices in mapreduce architectures. In *2013 IEEE international conference on big data*, pages 264–272. IEEE, 2013.

[BGL+18] Vladimir Braverman, Elena Grigorescu, Harry Lang, David P. Woodruff, and Samson Zhou. Nearly Optimal Distinct Elements and Heavy Hitters on Sliding Windows. In *Approximation, Randomization, and Combinatorial Optimization Algorithms and Techniques (APPROX/RANDOM 2018)*, volume 116, pages 7:1–7:22, 2018.

[BJRL15] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[BJW19] Ainesh Bakshi, Rajesh Jayaram, and David P Woodruff. Learning two layer rectified neural networks in polynomial time. In Alina Beygelzimer and Daniel Hsu, editors, *Proceedings of the Thirty-Second Conference on Learning Theory*, volume 99 of *Proceedings of Machine Learning Research*, pages 195–268, Phoenix, USA, 25–28 Jun 2019. PMLR.

[BKY18] Vladimir Braverman, Robert Krauthgamer, and Lin F. Yang. Universal Streaming of Subset Norms. *CoRR*, abs/1812.00241, 2018.

[Bra02] Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. In *European Conference on Computer Vision*, pages 707–720. Springer, 2002.

[Bub15] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[CCDS20] Rachit Chhaya, Jayesh Choudhari, Anirban Dasgupta, and Supratim Shit. Streaming coresets for symmetric tensor factorization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1855–1865, Virtual, 13–18 Jul 2020. PMLR.

[CCFC02] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[CD19] Graham Cormode and Charlie Dickens. Iterative hessian sketch in input sparsity time. In *NeurIPS Workshop: Beyond First-Order Optimization Methods in Machine Learning*, 2019.

[CDMI+16] Kenneth L Clarkson, Petros Drineas, Malik Magdon-Ismail, Michael W Mahoney, Xiangrui Meng, and David P Woodruff. The fast cauchy transform and faster robust linear regression. *SIAM Journal on Computing*, 45(3):763–810, 2016.

[CDP+19] Pern Hui Chia, Damien Desfontaines, Irippuge Milinda Perera, Daniel Simmons-Marengo, Chao Li, Wei-Yen Day, Qiushi Wang,

and Miguel Guevara. Khyperloglog: Estimating Reidentifiability and Joinability of Large Data at Scale. In *IEEE Symposium on Security and Privacy (SP)*, pages 867–881, 2019.

[CDW18] Graham Cormode, Charlie Dickens, and David P. Woodruff. Leveraging well-conditioned bases: Streaming and distributed summaries in Minkowski p-norms. In *International Conference on Machine Learning*, 2018.

[CDW21] Graham Cormode, Charlie Dickens, and David P. Woodruff. Subspace exploration: Bounds on projected frequency estimation. In *ACM Principles of Database Systems (PODS)*. ACM, 2021.

[CH+86] Samprit Chatterjee, Ali S Hadi, et al. Influential observations, high leverage points, and outliers in linear regression. *Statistical science*, 1(3):379–393, 1986.

[CJN18] Michael B. Cohen, T.S. Jayram, and Jelani Nelson. Simple Analyses of the Sparse Johnson-Lindenstrauss Transform. In Raimund Seidel, editor, *1st Symposium on Simplicity in Algorithms (SOSA 2018)*, volume 61 of *OpenAccess Series in Informatics (OASIcs)*, pages 15:1–15:9, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[CL98] Lorrie Faith Cranor and Brian A LaMacchia. Spam! *Communications of the ACM*, 41(8):74–83, 1998.

[CL11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[CLM+15] Michael B Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 181–190, 2015.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[CM05] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[CMM17] Michael B Cohen, Cameron Musco, and Christopher Musco. Input sparsity time low-rank approximation via ridge leverage score sampling. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1758–1777. SIAM, 2017.

[CMP16] Michael B. Cohen, Cameron Musco, and Jakub Pachocki. Online Row Sampling. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*, volume 60 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:18, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[CNW16] Michael B. Cohen, Jelani Nelson, and David P. Woodruff. Optimal Approximate Matrix Product in Terms of Stable Rank. 55:11:1–11:14, 2016.

[Coq21] Guillaume Coqueret. *Machine Learning in Finance: From Theory to Practice: by Matthew F. Dixon, Igor Halperin, and Paul Bilokon, Springer (2020). ISBN 978-3-030-41067-4. Paperback.* Taylor & Francis, 2021.

[CP15] Michael B Cohen and Richard Peng. ℓ_p row sampling by lewis weights. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 183–192. ACM, 2015.

[CW09] Kenneth L Clarkson and David P Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 205–214, 2009.

[CW13] Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, page 81–90, New York, NY, USA, 2013. Association for Computing Machinery.

[CW17] Kenneth L Clarkson and David P Woodruff. Low-rank approximation and regression in input sparsity time. *Journal of the ACM (JACM)*, 63(6):1–45, 2017.

[CY20] Graham Cormode and Ke Yi. *Small Summaries for Big Data.* Cambridge University Press, 2020.

[CYD18] Agniva Chowdhury, Jiasen Yang, and Petros Drineas. An iterative, sketching-based framework for ridge regression. In *International Conference on Machine Learning*, pages 989–998, 2018.

[DDH+08] A. Dasgupta, P. Drineas, B. Harb, R. Kumar, and M. W. Mahoney. Sampling algorithms and coresets for l_p regression. In *Proc. of the 19th Annual SODA*, pages 932–941, 2008.

[DHS05] Chris Ding, Xiaofeng He, and Horst D Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proceedings of the 2005 SIAM international conference on data mining*, pages 606–610. SIAM, 2005.

[DK06] Etienne De Klerk. *Aspects of semidefinite programming: interior point algorithms and selected applications*, volume 65. Springer Science & Business Media, 2006.

[DKM20] Michal Derezinski, Rajiv Khanna, and Michael W Mahoney. Improved guarantees and a multiple-descent curve for column subset selection and the nystrom method. *Advances in Neural Information Processing Systems*, 33, 2020.

[DMIMW12] Petros Drineas, Malik Magdon-Ismail, Michael W Mahoney, and David P Woodruff. Fast approximation of matrix coherence and statistical leverage. *The Journal of Machine Learning Research*, 13(1):3475–3506, 2012.

[DMMS11] Petros Drineas, Michael W Mahoney, Shan Muthukrishnan, and Tamás Sarlós. Faster least squares approximation. *Numerische mathematik*, 117(2):219–249, 2011.

[Doe20] Benjamin Doerr. Probabilistic tools for the analysis of randomized optimization heuristics. In *Theory of Evolutionary Computation*, pages 1–87. Springer, 2020.

[EY36] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[FI11] Cédric Févotte and Jérôme Idier. Algorithms for nonnegative matrix factorization with the β-divergence. *Neural computation*, 23(9):2421–2456, 2011.

[FMS+06] Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Cliff Stein, and Zoya Svitkina. On the complexity of processing

massive, unordered, distributed data. Technical Report CoRR abs/cs/0611108, ArXiV, 2006.

[Gal14] David Galvin. Three tutorial lectures on entropy and counting. *arXiv preprint arXiv:1406.7872*, 2014.

[Gér] Aurélien Géron. Chapter 2 – End-to-end Machine Learning project. `https://github.com/ageron/handson-ml/blob/master/02_end_to_end_machine_learning_project.ipynb`. [Online; accessed 2020].

[Gha17] Mina Ghashami. *On Frequent Directions, A Streaming Matrix Sketching Algorithm.* PhD thesis, The University of Utah, 2017.

[GJC20] Tushaar Gangavarapu, CD Jaidhar, and Bhabesh Chanduka. Applicability of machine learning in spam and phishing email filtering: review and approaches. *Artificial Intelligence Review*, pages 1–63, 2020.

[GLP16] Mina Ghashami, Edo Liberty, and Jeff M Phillips. Efficient frequent directions algorithm for sparse matrices. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 845–854, 2016.

[GLPW16] Mina Ghashami, Edo Liberty, Jeff M Phillips, and David P Woodruff. Frequent directions: Simple and deterministic matrix sketching. *SIAM Journal on Computing*, 45(5):1762–1792, 2016.

[GM09] Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5):2044–2059, 2009.

[GM13] Alex Gittens and Michael Mahoney. Revisiting the nystrom method for improved large-scale machine learning. In *International Conference on Machine Learning*, pages 567–575. PMLR, 2013.

[GO21] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021.

[GP14] Mina Ghashami and Jeff M Phillips. Relative errors for deterministic low-rank matrix approximations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 707–717. SIAM, 2014.

[GRB+19] Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and João Gama. Machine learning for streaming data: State of the art, challenges, and opportunities. *SIGKDD Explor. Newsl.*, 21(2):6–22, November 2019.

[GV18] Nicolas Gillis and Stephen A Vavasis. On the complexity of robust pca and ℓ_1-norm low-rank matrix approximation. *Mathematics of Operations Research*, 43(4):1072–1084, 2018.

[GVL13] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU press, 2013.

[Hab16] Itamar Haber. Count Min Sketch: The Art and Science of Estimating Stuff. `https://redislabs.com/blog/count-min-sketch-the-art-and-science-of-estimating-stuff/`, 18th August 2016. [Online; accessed 11-May-2021].

[Han78] Michael Lawrence Hand. Aspects of linear regression estimation under the criterion of minimizing the maximum absolute residual. 1978.

[Har] Nick Harvey. Notes on symmetric matrices. `https://www.cs.ubc.ca/~nickhar/W12/NotesMatrices.pdf`. Accessed: 2020.

[HK70] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[HMT11] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[HNH13] Stefan Heule, Marc Nunkesser, and Alexander Hall. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 683–692, 2013.

[Hua18] Zengfeng Huang. Near optimal frequent directions for sketching dense and sparse matrices. In *International Conference on Machine Learning*, pages 2048–2057. PMLR, 2018.

[HW13] Moritz Hardt and David P. Woodruff. How robust are linear sketches to adaptive inputs? In *Proceedings of the Forty-fifth*

Annual ACM Symposium on Theory of Computing, STOC '13, pages 121–130, New York, NY, USA, 2013. ACM.

[IW03] Piotr Indyk and David Woodruff. Tight lower bounds for the distinct elements problem. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 283–288. IEEE, 2003.

[JEP+20] John Jumper, R Evans, A Pritzel, T Green, M Figurnov, K Tunyasuvunakool, O Ronneberger, R Bates, A Zidek, A Bridgland, et al. High accuracy protein structure prediction using deep learning. *Fourteenth Critical Assessment of Techniques for Protein Structure Prediction (Abstract Book)*, 22:24, 2020.

[JST11] Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for l_p samplers, finding duplicates in streams, and related problems. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, page 49–58, New York, NY, USA, 2011. Association for Computing Machinery.

[JW09] T. S. Jayram and D. P. Woodruff. The Data Stream Space Complexity of Cascaded Norms. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 765–774, Oct 2009.

[JW18] Rajesh Jayaram and David P. Woodruff. Perfect ℓ_p Sampling in a Data Stream. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 544–555, 2018.

[JW21] Rajesh Jayaram and David Woodruff. Perfect l_p sampling in a data stream. *SIAM Journal on Computing*, 50(2):382–439, 2021.

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[KMA+19] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[KMVX18] Branislav Kveton, S. Muthukrishnan, Hoa T. Vu, and Yikun Xian. Finding Subcube Heavy Hitters in Analytics Data Streams. In *Proceedings of the 2018 World Wide Web Conference*, pages 1705–1714, 2018.

[KN14] Daniel M. Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *J. ACM*, 61(1), January 2014.

[KNR99] Ilan Kremer, Noam Nisan, and Dana Ron. On Randomized One-Round Communication Complexity. *Computational Complexity*, 8(1):21–49, 1999.

[KNW10] Daniel M Kane, Jelani Nelson, and David P Woodruff. An Optimal Algorithm for the Distinct Elements Problem. In *Proceedings of Principles of database systems*, pages 41–52. ACM, 2010.

[Lib13] Edo Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–588, 2013.

[LLW21] Yi Li, Honghao Lin, and David P Woodruff. Learning-augmented sketches for hessians. *arXiv preprint arXiv:2102.12317*, 2021.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[LMP13] Mu Li, Gary L Miller, and Richard Peng. Iterative row sampling. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 127–136. IEEE, 2013.

[LNNT16] Kasper Green Larsen, Jelani Nelson, Huy L. Nguyen, and Mikkel Thorup. Heavy Hitters via Cluster-Preserving Clustering. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS*, pages 61–70, 2016.

[LP19] Jonathan Lacotte and Mert Pilanci. Faster least squares optimization. *arXiv preprint arXiv:1911.02675*, 2019.

[LS14] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in o (vrank) iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014.

[LW13] Yi Li and David P Woodruff. A tight lower bound for high frequency moment estimation with small error. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 623–638. Springer, 2013.

[LWW20] Yi Li, Ruosong Wang, and David P Woodruff. Tight bounds for the subspace sketch problem with applications. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1655–1674. SIAM, 2020.

[Mah11] Michael W. Mahoney. Randomized algorithms for matrices and data. *Found. Trends Mach. Learn.*, 3(2):123–224, February 2011.

[McC18] Shannon McCurdy. Ridge regression and provable deterministic ridge leverage score sampling. In *Advances in Neural Information Processing Systems*, pages 2463–2472, 2018.

[MD09] Michael W Mahoney and Petros Drineas. Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009.

[MM13] Xiangrui Meng and Michael W Mahoney. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 91–100, 2013.

[MM17] Cameron Musco and Christopher Musco. Recursive sampling for the nystrom method. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[MMG17] Youssef Mroueh, Etienne Marcheret, and Vaibahava Goel. Co-occurring directions sketching for approximate matrix multiply. In *Artificial Intelligence and Statistics*, pages 567–575. PMLR, 2017.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Approximate Counting*, page 306–334. Cambridge University Press, 1995.

[MW10] Morteza Monemizadeh and David P. Woodruff. 1-pass relative-error ¡i¿l¡/i¿¡sub¿¡i¿p¡/i¿¡/sub¿-sampling with applications. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, page 1143–1160, USA, 2010. Society for Industrial and Applied Mathematics.

[MW17] Cameron Musco and David Woodruff. Is input sparsity time possible for kernel low-rank approximation? In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[NN13] Jelani Nelson and Huy L. Nguyên. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, FOCS '13, page 117–126, USA, 2013. IEEE Computer Society.

[PHL04] Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace Clustering for High Dimensional Data: a review. *SIGKDD Explorations*, 6(1):90–105, 2004.

[PKB14a] Dimitris Papailiopoulos, Anastasios Kyrillidis, and Christos Boutsidis. Provable deterministic leverage score sampling. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 997–1006, 2014.

[PKB14b] Dimitris Papailiopoulos, Anastasios Kyrillidis, and Christos Boutsidis. Provable deterministic leverage score sampling. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 997–1006, New York, NY, USA, 2014. Association for Computing Machinery.

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[PW15] Mert Pilanci and Martin J Wainwright. Randomized sketches of convex programs with sharp guarantees. *IEEE Transactions on Information Theory*, 61(9):5096–5115, 2015.

[PW16] Mert Pilanci and Martin J Wainwright. Iterative hessian sketch: Fast and accurate solution approximation for constrained least-squares. *The Journal of Machine Learning Research*, 17(1):1842–1879, 2016.

[PW17] Mert Pilanci and Martin J Wainwright. Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence. *SIAM Journal on Optimization*, 27(1):205–245, 2017.

[Ren88] James Renegar. A polynomial-time algorithm, based on newton's method, for linear programming. *Mathematical programming*, 40(1-3):59–93, 1988.

[Sar06] Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 143–152. IEEE, 2006.

[Sch20] Drew Schmidt. A survey of singular value decomposition methods for distributed tall/skinny data. In *2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, pages 27–34. IEEE, 2020.

[SKIW17] Uthayasankar Sivarajah, Muhammad Mustafa Kamal, Zahir Irani, and Vishanth Weerakkody. Critical analysis of big data challenges and analytical methods. *Journal of Business Research*, 70:263–286, 2017.

[SP21] Benwei Shi and Jeff Phillips. A deterministic streaming sketch for ridge regression. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 586–594. PMLR, 13–15 Apr 2021.

[Spo76] VA Sposito. Minimizing the maximum absolute deviation. *ACM SIGMAP Bulletin*, (20):51–53, 1976.

[SRD+18] Philip Schmidt, Attila Reiss, Robert Duerichen, Claus Marberger, and Kristof Van Laerhoven. Introducing wesad, a multimodal dataset for wearable stress and affect detection. In *Proceedings of the 20th ACM international conference on multimodal interaction*, pages 400–408, 2018.

[SSH+14] Fumin Shen, Chunhua Shen, Rhys Hill, Anton van den Hengel, and Zhenmin Tang. Fast approximate l_∞ minimization: speeding up robust regression. *Computational Statistics & Data Analysis*, 77:25–37, 2014.

[Sub] Sublinear.info. Open problem 94. `https://sublinear.info/index.php?title=Open_Problems:94`.

[SW11] Christian Sohler and David P. Woodruff. Subspace embeddings for the l_1-norm with applications. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, page 755–764, New York, NY, USA, 2011. Association for Computing Machinery.

[SWZ17] Zhao Song, David P Woodruff, and Peilin Zhong. Low rank approximation with entrywise l1-norm error. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 688–701, 2017.

[Tea17] Differential Privacy Team. CLearning with Privacy at Scale. `https://machinelearning.apple.com/research/learning-with-privacy-at-scale`, December 2017. [Online; accessed 11-May-2021].

[Tib11] Robert Tibshirani. Regression shrinkage and selection via the lasso: a retrospective. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(3):273–282, 2011.

[Tur17] Chintan Turakhia. Engineering More Reliable Transportation with Machine Learning and AI at Uber. `https://eng.uber.com/machine-learning/`, 10th November 2017. [Online; accessed 11-May-2021].

[TW12] Srikanta Tirthapura and David P. Woodruff. A General Method for Estimating Correlated Aggregates over a Data Stream. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, pages 162–173, 2012.

[TYUC19] Joel A Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. Streaming low-rank matrix approximation with an application to scientific simulation. *SIAM Journal on Scientific Computing*, 41(4):A2430–A2463, 2019.

[TZ12] Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal on Computing*, 41(2):293–331, 2012.

[UT19] Madeleine Udell and Alex Townsend. Why are big data matrices approximately low rank? *SIAM Journal on Mathematics of Data Science*, 1(1):144–160, 2019.

[Vu18] Hoa Vu. *Data Stream Algorithms for Large Graphs and High Dimensional Data*. PhD thesis, U. Massachusetts at Amherst, 2018.

[VW81] Paul F Velleman and Roy E Welsch. Efficient computing of regression diagnostics. *The American Statistician*, 35(4):234–242, 1981.

[WBW+11] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[WGM17] Shusen Wang, Alex Gittens, and Michael W Mahoney. Sketched ridge regression: Optimization perspective, statistical perspective, and model averaging. *The Journal of Machine Learning Research*, 18(1):8039–8088, 2017.

[WLM+17] Jialei Wang, Jason Lee, Mehrdad Mahdavi, Mladen Kolar, and Nati Srebro. Sketching meets random projection in the dual: A provable recovery algorithm for big and high-dimensional data. In *Artificial Intelligence and Statistics*, pages 1150–1158. PMLR, 2017.

[Woo04] David Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, page 167–175, USA, 2004. Society for Industrial and Applied Mathematics.

[Woo14a] David Woodruff. Low rank approximation lower bounds in row-update streams. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[Woo14b] David Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1-2):1–157, 2014.

[WS01] Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2001.

[WW19] Ruosong Wang and David P Woodruff. Tight bounds for l_p oblivious subspace embeddings. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1825–1843. SIAM, 2019.

[WZ13] David Woodruff and Qin Zhang. Subspace embeddings and ℓ_p-regression using exponential random variables. In *Conference on Learning Theory*, pages 546–567, 2013.

[YCRM17] Jiyan Yang, Yin-Lam Chow, Christopher Ré, and Michael W Mahoney. Weighted sgd for ℓ_p regression with randomized preconditioning. *The Journal of Machine Learning Research*, 18(1):7811–7853, 2017.

[YMM13] Jiyan Yang, Xiangrui Meng, and Michael Mahoney. Quantile regression for large-scale applications. In *International Conference on Machine Learning*, pages 881–887, 2013.

[ZH05] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.

[ZMJ+13] Lijun Zhang, Mehrdad Mahdavi, Rong Jin, Tianbao Yang, and Shenghuo Zhu. Recovering the optimal solution by dual random projection. In *Conference on Learning Theory*, pages 135–157. PMLR, 2013.