Sparse solutions for linear prediction problems

by
Tyler Neylon

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Mathematics
New York University
May 2006

Dennis Shasha—Advisor

for my father

Acknowledgements

I would like to begin by thanking my first advisor, Professor Dennis Shasha. He has been by far the most helpful and influential person throughout the majority of my graduate career. He has provided an incredible amount of encouragement, advice, and research wisdom for which I am extremely grateful.

Next I would like to thank my second advisor, Professor Mehryar Mohri. Over the past year, he has been happy to discuss the fine technical points of machine learning for many hours at a time. I am extremely appreciative of his consistent generosity, good humor, and expert insight.

Having the opportunity to meet and work with these two extraordinary professors has been my favorite part of graduate school.

I would like to thank Professor Richard Cole for the many hours of attention and thought he devoted to the understanding and improvement of my thesis work.

I am also grateful for the time and expert time series advice of Professor Clifford Hurvich.

Thanks as well to Lee-Ad "Adi" Gottlieb, with whom I collaborated on some complexity results. Adi is a very bright fellow who is fun to work with.

I wish to mention with much appreciation the following people, each of whom provided technical knowledge or acumen along the way: Professors Sinan Gunturk, Michael Overton, Janos Pach and Percy Deift; and fellow students / researchers Ashish Rastogi, Cynthia Rudin, Martin Raphan, Rosa Figueras, and Xiaojian Zhao.

Finally, thank you to my family and friends for your kindness and support!

Sparse solutions for linear prediction problems

Author: Tyler Neylon

Advisors: Dennis Shasha and Mehryar Mohri

Abstract

Simplicity is the ultimate sophistication.
-Leonardo da Vinci

The simplicity of an idea has long been regarded as a sign of elegance and, when shown to coincide with accuracy, a hallmark of profundity.

In this thesis our ideas are vectors used as predictors, and sparsity is our measure of simplicity. A vector is sparse when it has few nonzero elements. We begin by asking the question: given a matrix of n time series (vectors which evolve in a "sliding" manner over time) as columns, what are the simplest linear identities among them? Under basic learning assumptions, we justify that such simple identities are likely to persist in the future. It is easily seen that our question is akin to finding sparse vectors in the null space of this matrix. Hence we are confronted with the problem of finding an optimally sparse basis for any vector space. This is a computationally challenging problem with many promising applications, such as iterative numerical optimization [18], [10], fast dimensionality reduction [31], graph algorithms on cycle spaces [33], and of course the time series work of this thesis.

In part I, we give a brief exposition of the questions to be addressed here: finding linear identities among time series, and asking how we may bound the generalization error by using sparse vectors as hypotheses in the machine learning versions of these problems. In part II, we focus on the theoretical justification for maximizing sparsity as a means of learning or prediction. We'll look at sample compression schemes as a means of correlating sparsity with the capacity of a hypothesis set, as well as examining learning error bounds which support sparsity. Finally, in part III, we'll illustrate an increasingly sophisticated toolkit of incremental algorithms for discovering sparse patterns among evolving time series.

Contents

Dedication iv
Acknowledgements v
List of Figures xii

I Introduction **1**

1 Linear Prediction Problems **2**
1.1 The correlation problem 4
1.2 Generalizing the correlation problem 6
1.3 Guarantees from learning theory 15

II Sparsity in theory: learning **21**

2 Elements of learning theory **24**
2.1 The PAC model 24

3 Learning as sample compression **35**
3.1 Labeled sample compression schemes 36
3.1.1 Basic properties of concept classes 37
3.1.2 Sauer's Lemma 39
3.1.3 Maximum concept classes 42
3.2 Unlabeled sample compression schemes 49

4 Sparsity for linear predictors **54**
4.1 A bound on the VC dimension 54
4.2 The regression case 60

III Sparsity in practice: Time series algorithms **64**

5 Introduction **67**
5.1 The predictive power of the null space 70

6 Theory of sparse matrices **76**
6.1 Sparsity inequalities 76
6.2 Gaussian elimination for sparsity 78
6.3 Most matrices are completely unsparsifiable 79
6.4 Equivalent conditions for sparsifiability 81

7 Exact matrix sparsification **88**
7.1 NP-hardness and other reductions 89
7.1.1 Sparsest Independent Vector $\in$ NP 94
7.1.2 Sparsest Vector is NP-hard 96
7.2 Algorithms 96
7.2.1 Previous algorithms 97
7.2.2 New incremental algorithms 107

8 Probabilistic and approximate sparsification **123**
8.1 Hardness of approximation 123
8.1.1 Equivalence and APX-hardness of Min-Unsatisfy 124
8.1.2 APX-hardness of Matrix Sparsification 128
8.1.3 An approximation algorithm in RP 133
8.1.4 L^1 minimization approximates L^0 minimization 140
8.2 Algorithms 142
8.2.1 Heuristics for solving Matrix Sparsification 142
8.2.2 Fast incremental algorithms with high sparsity 145
8.2.3 Idea of the $\epsilon-$space 149
8.2.4 The block power iteration 154
8.2.5 Time complexity 156
8.2.6 Experiments 158

9 Conclusion **165**
9.1 Summary 165
9.2 Future Work 168
9.3 Conclusion 169

Notation **170**

Linear Algebra Glossary **174**

List of Figures

7.1 One iteration of the partial history null track algorithm 115
7.2 Density of null space comparison on simulated data. Back substitution remains between 3–6% while our periodically recalibrated algorithm achieves between 3–7%. 120
7.3 Average number of nonzeros per column in stock price data. A full column could have contained up to 500 nonzeros. Back substitution alternated between 38–41 while our algorithm accomplished between 41–49. 121
7.4 Relative speedup factor between our algorithm and back substitution over partial history sliding windows of increasing width. Tests were run on 1000 streams of simulated data. The factor represents the time efficiency gained over one iteration. 122

8.1 Average relative test error for different k 160
8.2 Average percentage change of nonzero coefficients (of x) for different k 161
8.3 Histogram of classification quality 163
8.4 Time (in seconds) per iteration for different versions of our algorithm 164

Part I

Introduction

Chapter 1

Linear Prediction Problems

In this chapter we'll survey the core algorithmic problem addressed in this thesis: finding simple linear identities among time series. We'll motivate this problem by treating it as a natural generalization of correlated pairs of time series, and in later chapters we will argue that such identities tend to be good predictors of future behavior in a certain model of learning.

A time series is a stream of data which we think of as evolving over time. The data could represent live stock prices or returns, the blood pressure of a patient being monitored, or sound data from a song being played. In our case, we are interested in large sets of time series from which we will derive patterns, such as temperature readings from multiple locations at once, or live stock prices of many symbols (e.g. AAPL, GM, IBM,...). We will also focus on *discrete* time series, whose entries may be indexed by integers corresponding to atomic time units (such as seconds) — this is in contrast with a *continuous* time series, whose entries may be considered as indexed by an arbitrary real value $t \in \mathbb{R}$.

Mathematically, a set of (discrete) time series may be considered as a matrix. At any time $t \in \mathbb{N}$, we have the $t \times n$ matrix $A_t = (a_{ij})$, in which each column contains all the data collected thus far from a single source. In this model, we assume that the time series have been synchronized so that each row corresponds to a single time step. We have n data sources, one for each column — entry a_{ij} is the data collected from stream/column j at time/row i. Algorithms which operate on a set of time series would be wise to function in an incremental fashion, taking advantage of computations on A_{t-1}, which form the first $t-1$ rows of the next matrix, A_t at time t.

In some applications, we are concerned only with data from the last m time

steps — here, m is the length of our sliding window. Restricting our attention to this limitation of past data may be considered as a type of Markovian assumption on the dependency between past and present; in particular, we assume that the patterns we are interested in (to be outlined below) will be equally evident even if we ignore data more than m time units old. In this case, instead of using the usually larger matrix A_t, which contains *a*ll our data, we'll work with matrix S_t which is restricted to this *s*liding window of data. Thus, S_t is of some fixed size $m \times n$ and is composed of the last m rows of matrix A_{t+m}. (We use A_{t+m} instead of A_t since otherwise there would not be enough rows in A_t for small t to complete an entire sliding window of size m.)

1.1 The correlation problem

In this section, we consider the problem of finding all pairs, among a given set of vectors, which have a high correlation coefficient.

Noticing certain simple patterns which have been consistently true over the past m time units can often help us form reasonable expectations of future behavior. For now we consider which patterns may be interesting to discover; we will discuss future behavior further in §1.3. One of the simplest such patterns is the idea of a correlated pair.

Given two column vectors s_i and s_j from matrix S_t, we'd like to measure how close they are to fitting a linear equation of the form $as_i + bs_j = c\vec{\mathbf{1}}$, where $\vec{\mathbf{1}}$ is the column vector of all ones. The traditional measure of "closeness" here is the **Pearson correlation** r_{ij} (or just 'the correlation,' [50], page 567), which can be defined in terms of the slopes given by using the least-squares method to approximate the linear coefficients.

In particular, we may define

$$r_{ij} := \frac{\sum_k (s_{ki} - \bar{s}_i)(s_{kj} - \bar{s}_j)}{\sqrt{\sum_k (s_{ki} - \bar{s}_i)^2 \sum_k (s_{kj} - \bar{s}_j)^2}}, \tag{1.1}$$

where $\bar{s}_i$ is the average of the values in s_i, and s_{ki} is the kth value in vector s_i. Throughout this thesis, we will consistently use s_k to denote a column vector (such as the kth column of a matrix S), s^k for a row, and s_{ij} for the ith value in column j, or equivalently the jth value in row i; this internal consistency will help to avoid confusion about how our vectors relate to our matrices.

By defining the zero-mean vectors z_ℓ via $z_{k\ell} := s_{k\ell} - \bar{s}_\ell$ for $\ell = i, j$, we can rewrite (1.1) as

$$r_{ij} = \frac{\langle z_i, z_j \rangle}{||z_i|| \cdot ||z_j||} = \cos\theta_{ij}, \tag{1.2}$$

where $\theta_{ij} \in [0, \pi/2]$ is the angle between z_i and z_j. Our 'default' norm, denoted simply by $||\cdot||$, is always the Euclidean norm. If these zero-mean vectors fit a linear least-squares model, it will now be of the form $az_i + bz_j = 0$. Hence measuring the linear correlation of z_i and z_j corresponds with measuring their linear dependence. Given two vectors, the angle between the two is a fair quantification of their linear dependence. Thus, pairs with low correlation have an angle θ_{ij} close to $\pi/2$ and a coefficient r_{ij} close in magnitude to 0; while those pairs with higher correlation have θ_{ij} near zero and hence r_{ij} close in magnitude to 1. Notice that by the Cauchy-Schwarz inequality, we always have $|r_{ij}| \leq 1$ (many important linear algebra concepts, including the Cauchy-Schwarz inequality, are breifly reviewed in the Linear Algebra Glossary at the end of this thesis).

From this point on, we will treat each of s_i and s_j as normalized data so that $\bar{s}_i = \bar{s}_j = 0$ and $||s_i|| = ||s_j|| = 1$. Such normalization is relatively easy to achieve in practice by setting $s_i = z_i/||z_i||$, where the z_i are the zero-mean versions of the raw data. This normalization will not be used in our incremental algorithms presented later; it is simply a means of conveniently presenting the motivating material in this introduction. In this simplified case, (1.2) becomes

$$r_{ij} = \langle s_i, s_j \rangle.$$

That is, r_{ij} is simply the dot product between s_i and s_j. Thus we have justified that, when the columns of matrix S_t are normalized, setting $R := S_t^T S_t$ truly results in a **correlation matrix** since the entry r_{ij} in R is exactly the correlation coefficient (of the same name) above. Moreover,

$$||s_i - s_j||^2 = \langle s_i - s_j, s_i - s_j \rangle = ||s_i||^2 + ||s_j||^2 - 2\langle s_i, s_j \rangle = 2 - 2r_{ij},$$

so that the pairs with highest correlation coefficient r_{ij} are exactly those which minimize the distance $||s_i - s_j||$.

Thus we have reduced the problem of finding all pairs with a high correlation coefficient to

Closest Pairs Given $m \times n$ matrix S and threshold $\epsilon > 0$, find all pairs $\{i, j\} \subset [n]$ with $||s_i - s_j|| < \epsilon$.

Here, as above, s_i denotes the ith column of S, and $[n] := \{1, 2, \ldots, n\}$.

1.2 Generalizing the correlation problem

In this section, we propose the idea of small linear identites, or near-identities, as a generalization of correlated pairs. We have considered vectors s_1 and s_2 to be closely correlated if they nearly fit a linear equation of the form $v_0\vec{\mathbf{1}} + v_1 s_1 + v_2 s_2 \approx 0$ for scalar values v_0, v_1, v_2, which we could state more precisely as

$$||v_0\vec{\mathbf{1}} + v_1 s_1 + v_2 s_2|| < \epsilon$$

for some given $\epsilon > 0$.

Hence, given n vectors $s_1, \ldots, s_n$, it seems natural to ask if there exists a column vector v of coefficients $v_0, \ldots v_n$ so that

$$||v_0\vec{\mathbf{1}} + v_1 s_1 + v_2 s_2 + \ldots + v_n s_n|| < \epsilon.$$

If matrix S has an all-ones column $\vec{\mathbf{1}}$ as its first column, followed by $s_1, \ldots, s_n$, then we see that $Sv = v_0\vec{\mathbf{1}} + v_1 s_1 + v_2 s_2 + \ldots + v_n s_n$, allowing us to restate the inequality as

$$||Sv|| < \epsilon. \tag{1.3}$$

If only finitely many vectors v satisfy this inequality, then we would like to discover those.

However, there are infinitely many vectors v which satisfy this equation. Indeed, if v is any vector whatsoever (of the appropriate dimension), then we may choose a scalar μ with $|\mu| < \epsilon/||Sv||$ so that vector μv then satisfies equation (1.3).

As a first step, we will require that v be a unit vector; that is, we require $||v|| = 1$ to eliminate the above source of an abundance of solutions v. However, there will still be many matrices S for which infinitely many vectors v satisfy (1.3). For instance, if $\vec{u}, \vec{v}$ are orthogonal unit vectors so that both $||S\vec{u}||$ and $||S\vec{v}||$ are bounded above by $\epsilon/\sqrt{2}$, then, for any pair of scalars x, y with $x^2 + y^2 = 1$, we have

$$||S(x\vec{u} + y\vec{v})|| \leq |x| \cdot ||S\vec{u}|| + |y| \cdot ||S\vec{v}|| \leq \epsilon$$

since $|x| + |y| \leq \sqrt{2}$; while

$$||x\vec{u} + y\vec{v}||^2 = x^2 + y^2 = 1,$$

since $\langle \vec{u}, \vec{v} \rangle = 0$. In other words, based on the two vectors $\vec{u}$ and $\vec{v}$, we have created an infinite set of vectors $\{x\vec{u} + y\vec{v} : x^2 + y^2 = 1\}$ which all satisfy equation (1.3).

Thus, we must further specify which coefficient vectors v are of interest to us. Again drawing our motivation as a generalization of the Closest Pairs problem, we will focus on those solutions to (1.3) which involve the *smallest number of columns of* S. We will restate this idea in terms of sparsity: if a particular coefficient $v_i = 0$, then column s_i essentially plays no role in equation (1.3). Thus those coefficient vectors v with the most zeros — the *sparsest* v's — are exactly those which involve the smallest number of data streams (columns).

In order to formally define our problem, we will now define the **support** of a vector v as the set of coordinates which are nonzero; specifically

$$\text{supp}(v) := \{i : v_i \neq 0\}.$$

Within a set of vectors $V \subset \mathbb{R}^n$, a particular vector $v \in V$ has **minimal support** when there is no $y \in V$ such that $\text{supp}(y) \subsetneq \text{supp}(v)$. A vector with minimal support is sparse in the sense that its support cannot be reduced within V.

We may now pose the problem

Minimal Linear Near-Identities Given matrix S, find all unit vectors v satisfying $\|\|Sv\|\| < \epsilon$ with minimal support.

Rather than rigorously defining the term *sparsity* in this thesis, we will restrict this idea to the partial ordering of vectors based on their number of nonzeros. In particular, we will say that vector x is **sparser** than y iff $\#\text{supp}(x) < \#\text{supp}(y)$. Thus the solutions to Minimal Linear Near-Identities certainly include what we would like to indicate as "the sparsest" vectors v satisfying (1.3).

Notice that if we had not specified v to be nonzero, then $v = 0$ would be our only (non-informative) solution.

At this point, we will examine some very similar problems which lend themselves more readily to an algorithmic analysis.

Our first variation will be to examine the non-approximation version of Minimal Linear Near-Identities, which is quite simply

Minimal Linear Identities Given matrix S, find all nonzero vectors v satisfying $Sv = 0$ with minimal support.

Here we may omit the condition that v be a unit vector: in the approximation problem Minimal Linear Near-Identities any vector v could be scaled down by some factor μ so that $||S\mu v||$ would be small enough, but in this exact version, if $Sv \neq 0$, then no nonzero scalar μ will cause $S(\mu v) = 0$.

A **circuit** among a set of vectors is a nonempty linearly dependent subset of minimal size. The reader familiar with matroids [37] will recognize that the Minimal Linear Identities problem is nothing more than discovering all the *circuits* among the columns of S.

This last problem formulation brings to focus the null space of matrix S, which we may denote by $\mathsf{null}(S)$: by definition, any vector $v \in \mathsf{null}(S)$ satisfies $Sv = 0$. We may wonder, intuitively, how Minimal Linear Identities relates to the problem of finding an "optimally sparse" basis to the null space. Let us define some terms in order to formally pose this new problem.

A **null matrix** N for A obeys $AN = 0$. A **full null matrix** N for A is a null matrix whose columns form a basis for $\mathsf{null}(A)$. Equivalently, the $n \times c$ matrix N is a full null matrix for A iff

- $AN = 0$, and
- $\mathsf{rank}(N) = c \leq n$, where
- $c = \dim(\mathsf{null}(A))$.

The dimension c of $\mathsf{null}(A)$ is also referred to as the **corank** of A, written $\mathsf{corank}(A)$.

A matrix N is **optimally sparse** iff, for any invertible matrix T, matrix NT has at least as many nonzeros as N. When the columns of N are linearly independent, the following statements are equivalent:

- $M = NT$ for some invertible matrix T;
- $\mathsf{col}(M) = \mathsf{col}(N)$ and M is of the same dimensions as N.

This follows from the fact that the columns of M and N are both bases of the same space $\mathsf{col}(N) = \mathsf{col}(M)$. Thus any optimally sparse matrix N with linearly independent columns satisfies, for all M of the same dimensions as N,

$$\mathsf{col}(M) = \mathsf{col}(N) \implies M \text{ has at least as many nonzeros as } N.$$

We may now pose our new problem, which, in various flavors, will become the core problem for the rest of this thesis:

Sparse Null Space Given matrix A, find an optimally sparse full null matrix N for A.

Suppose T is an invertible matrix. Notice that NT is a full null matrix for A iff N is. This follows since $\mathsf{rank}(NT) = \mathsf{rank}(N)$ and $\mathsf{col}(NT) = \mathsf{col}(N)$. Hence, by requiring that N be optimally sparse in this problem, we are simultaneously requiring that the columns of N have the least number of nonzeros among all full null matrices for A.

At this point it may be helpful to give an example:

Example 1 *Suppose our data is represented by the* 3×6 *matrix*

$$A = \begin{pmatrix} 2 & 5 & 4 & 4 & 1 & 5 \\ 1 & 1 & 1 & 1 & 1 & 3 \\ 0 & 5 & 4 & 4 & 1 & 1 \end{pmatrix}.$$

Then the set $\mathcal{M}$ *of minimal linear identities is given by*

$$\mathcal{M} = \left\{ \begin{pmatrix} 2 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ 3 \\ -4 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ -1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 3 \\ 0 \\ -4 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -2 \\ 3 \\ -4 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -2 \\ 3 \\ 0 \\ -4 \\ 0 \\ 1 \end{pmatrix} \right\}.$$

On the other hand, here is an optimally sparse full null matrix:

$$N = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & -4 & -1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}.$$

Notice that the three columns of N *are exactly the first three linear identities in* $\mathcal{M}$*. The last three identities in* $\mathcal{M}$ *are various linear combinations of the first three (and therefore of columns of* N*).*

We could have formed other full null matrices out of the vectors in $\mathcal{M}$*; but notice that our optimally sparse* N *will necessarily avoid the last two vectors in* $\mathcal{M}$*, which, although having minimal support, are less sparse than the rest. We'll look more closely at the relationship between columns of* N *and vectors in* $\mathcal{M}$ *in property 1 below.*

This example highlights the question: How is the problem sparse null matrix related to minimal linear identities? We will see that every column of N represents a circuit of A. In fact, we will see even more than this:

Property 1 *Suppose N is an optimally sparse full null matrix for A. Let $\mathcal{C}_i$ denote the set of circuits containing column a_i of A. If $\mathcal{C}_i$ is nonempty, then there is a column n_j in N so that*

1. *supp$(n_j) \in \mathcal{C}_i$, and*
2. *supp(n_j) is of minimal size within $\mathcal{C}_i$.*

In other words, the sets given by $\text{supp}(n_j)$ for each column of N represent a circuit of minimal size. What's more, we don't need to worry about some column of A missing from the supports of columns in N; this can happen only when that column is in no circuits at all.

For the proof it will be useful to introduce the so-called "**zero-norm**" on vectors, defined on $x \in \mathbb{R}^n$ by

$$||x||_0 := |\text{supp}(x)| = \#\{i : x_i \neq 0\}.$$

Technically, this is not actually a norm since it fails to scale — that is, $||\lambda x||_0 \neq |\lambda| \cdot ||x||_0$ for any scalar $\lambda \notin \{0, 1\}$. Unlike true norms on $\mathbb{R}^n$, it is neither convex nor even continuous. However, it is the case that $||x||_0 \geq 0$ with equality exactly when $x = 0$. And the triangle inequality holds: for the zero-norm,

$$||\alpha + \beta||_0 \leq ||\alpha||_0 + ||\beta||_0$$

means exactly that

$$|A \cup B| \leq |A| + |B|$$

for $A = \text{supp}(\alpha), B = \text{supp}(\beta)$; and this inequality is clearly true for any two sets A and B.

We are now ready to give the

Proof of Property 1.

As above, $\mathcal{C}_i$ denotes the set of circuits among columns of A which include the ith column a_i.

Suppose the set of circuits $\mathcal{C}_i$ is nonempty. Technically, circuits are just sets, so we'll identify each $c \in \mathcal{C}_i$ with a vector $\vec{c}$ so that $\text{supp}(\vec{c}) = c$ and $A\vec{c} = 0$.

Choose a minimally-sized $s \in \mathcal{C}_i$ (s is *s*mall). Since N is a full null matrix, $\vec{s} \in \mathsf{col}(N) = \mathsf{null}(A)$, so that we may find coefficients x satisfying $Nx = \vec{s}$.

If the ith row of N were all zero, then we would also have $\vec{s}_i = 0$, which is not the case. Hence there must be some column n_j with $j \in \text{supp}(x)$ and $n_{ij} \neq 0$; we claim that $\text{supp}(n_j) \in \mathcal{C}_i$ and $||n_j||_0 = ||\vec{s}||_0$, so that n_j meets both conditions (1) and (2) of the property.

Indeed, suppose $||n_j||_0 > ||\vec{s}||_0$, and let matrix M be identical to N except that column n_j is replaced by $\vec{s}$. Notice that $n_j \in \mathsf{col}(M)$ since $j \in \text{supp}(x)$ and $\vec{s} = Nx$ gives us a linear equation with nonzero coefficients between $\vec{s}$ and n_j, which may be solved for n_j — that is, we may turn the equation $\vec{s} = Nx$ into $n_j = Mz$ for some z.

Thus we would have $\mathsf{col}(M) = \mathsf{col}(N)$, but M is strictly sparser than N, since $||\vec{s}||_0 < ||n_j||_0$. By our choice of N, this is impossible; so it must be the case that $||n_j||_0 = ||\vec{s}||_0$, confirming part (2).

If $\text{supp}(n_j) \notin \mathcal{C}_i$, then, since $\text{supp}(n_j)$ indexes a linearly dependent subset of the columns of A including a_i, there must be some subset $\tau \subsetneq \text{supp}(n_j)$ which is in $\mathcal{C}_i$. But then $|\tau| < ||n_j||_0 = |s|$, contradicting our choice of s. Hence $\text{supp}(n_j) \in \mathcal{C}_i$, confirming part (1) of the property.

This completes the proof. □

In this section, we have been introduced to the core problem Sparse Null Space; the majority of this thesis is devoted to how and why to solve different versions of this problem. Property 1 shows us that any solution to Sparse Null Space is a useful subset of those solutions to Minimal Linear Identities, which, along with Minimal Linear Near-Identities, we have posed as a natural generalization of finding highly correlated pairs among a set of time series.

1.3 Guarantees from learning theory

The previous section posed the Sparse Null Space problem as a potential answer to *which* patterns among time series may be interesting to us. In this chapter we will mathematically formulate the question: *why* should the vectors in Sparse Null Space be useful to us? In particular, we recognize that our ultimate goal is not always retrospective subset selection, but often to discover some information about the time series which will persist *in the future*. Thus, we should ask: Is sparsity really a useful heuristic for predictive power?

In order to pose our answer to this question, as we will do in §4 below, we must first describe the idea of *structural risk minimization* outlined by

Vladimir Vapnik in [49].

When attempting to learn a pattern based on a set of training data, one often arrives at the problem of *overfitting*. Intuitively, overfitting occurs when the pattern learned is given too many degrees of freedom. A natural example is that of choosing a polynomial to approximate a set of points in the plane. It may be the case that these points were in fact generated by a very low-degree polynomial but our data is noisy. Given 20 noisy points approximately on a parabola, we may unwisely find a degree 19 polynomial which perfectly fits each point. However, it is well-known that such an exactly fitting polynomial is very likely to exhibit erratic behavior outside the range of the training data (see, for example, lecture 11 in [44]).

Thus arises the question: when learning a pattern based on training data, how can we avoid overfitting? The answer given by structural risk minimization is to restrict our learned pattern to a set with only a limited degree of freedom. In the example of polynomials, this would correspond with choosing a curve of limited degree — we could, say, choose the *parabola* which best fits the noisy data, which would then give better predictions of extrapolated values on the curve.

We will postpone rigorous definitions until chapter 2, but for now let us roughly state the objective of our approach to learning patterns: given sets of patterns $P_1, P_2, \ldots$ such that set P_i has, in some sense, i degrees of freedom, and a set of training data T, we would like to choose a set P_i a pattern $p \in P_i$ which will minimize some balance between the error on the training data $error(p, T)$ and the degree of freedom i allowed within set P_i. Thus we are attempting to avoid overfitting by reducing the degrees of freedom allowed to our learned pattern p — this is the intuitive idea behind structural risk minimization. (The machine learning reader may anticipate that we will eventually formalize the idea of "degrees of freedom" as the VC dimension.)

How can we apply these ideas to our problem Sparse Null Space? We will pose the learning problem in terms of a single-vector version of Sparse Null Space:

Approx_P$_0$ Given a matrix A, column vector b, and an error tolerance ϵ, find x which solves

$$\begin{cases} \min & ||x||_0 \\ \text{s.t.} & ||Ax - b|| < \epsilon. \end{cases}$$

If x is a solution to $\mathsf{Approx_P}_0$, then we may define matrix $C = (A\ b)$, and vector $y = \binom{x}{-1}$ to see that

$$||Cy|| = ||Ax - b|| < \epsilon,$$

so that y is an approximating vector to the null space of C. Moreover, this problem is attempting to sparsify this vector by minimizing its number of nonzeros. Thus $\mathsf{Approx_P}_0$ may be considered an approximation version of $\mathsf{Sparse\ Null\ Space}$ which focuses on a particular column or time series b of our data.

At the same time, we may think of $\mathsf{Approx_P}_0$ as attempting to learn a pattern x which gives a relationship between the columns of A and column b. Since we consider these columns as time series, there will be a new row α added to A and a new coordinate β to b. Our goal in learning x is to hope that

$$\langle \alpha, x \rangle \approx \beta,$$

which corresponds to our approximation $Ax \approx b$ remaining (approximately) true at the next time step.

In this formulation, we specify the error tolerance ϵ as an input to the problem. Thus, in terms of structural risk minimization, our theoretical goal is to show that *a set of sparse vectors x has a low "degree of freedom."* If we can show this, then solving $\mathsf{Approx_P}_0$ as a learning problem does indeed help minimize a balance between the error of the pattern and the "degrees of freedom" of our set of patterns.

We state this as

Question 2 *Is it the case that, by restricting our choice to sparse vectors x among those for which $Ax \approx b$, we effectively reduce our "degrees of freedom" for the sake of structural risk minimization?*

Why should we expect this to be true? Intuitively, one could argue that a sparse vector is a pattern based on only a small subset of our data. So it becomes natural to ask: how does sparsity in learning, in general, relate to effective learning algorithms? If *linear* sparse patterns are a useful goal because of their sparsity, then perhaps any pattern based on only a small subset of the data is also helpful. Thus we will generalize question 2 as

Question 3 *Suppose a learning algorithm operates on a set of training data T by first finding a small subset $S \subset T$ to focus on, and then, "forgetting" the*

rest of T, learns a pattern based entirely on subset S. Can this technique be an effective approach to learning?

An affirmative answer here would provide even more justification to the idea of seeking sparse solutions to linear prediction problems, as those are an interesting specific case of the above idea. This last question is examined more carefully in §3 below.

The questions presented here give our primary motivations for the ideas to be discussed in part II below.

In this chapter, we have become acquainted with the key ideas behind this thesis: the problem Sparse Null Space and variations thereof (such as Approx_P_0) serve as interesting generalizations to correlated pairs. In the work to follow, we explore first *why* solving these problems is likely to help in predicting future behavior among time series, and then *how* we may go about efficiently discovering solutions on real data.

Part II

Sparsity in theory: learning

In this part, we'll focus on justifying why we expect sparsity-based predictions to be accurate. In particular, we will use the *P*robably *A*pproximately *C*orrect model (the PAC model) of learning formulated by Leslie Valiant [47] and built upon the work of Vladimir Vapnik and Alexey Chervonenkis [48].

The PAC model is named for providing a prediction guarantee of the form

$$P(error > \epsilon) < \delta,$$

where $\delta, \epsilon > 0$ are (small) inputs which determine how hard the algorithm must work to achieve this guarantee. We can think of this inequality as assuring us that, with high confidence $(1 - \delta)$, the general prediction of the algorithm will be very accurate (within ϵ) — it is probably approximately correct.

The fundamental assumption of this model is that the data points used for training are drawn in an i.i.d. fashion according to the *same* distribution by which future points (or *test sample* points) are received. The model has the advantage of assuming nothing at all about this distribution, and is flexible enough to accommodate a vast range of anticipated patterns in the data. However, only certain types of patterns, or *concept classes*, are provably learnable in this model.

First (in chapter 2), we will briefly review the elements of learning theory surrounding the PAC model. Next (in chapter 3), we will consider compression schemes in learning as an abstract application of sparsity in learning. Finally (in chapter 4), we will concentrate on demonstrating the utility of sparsity in discovering predictors of linear patterns.

Chapter 2

Elements of learning theory

In this chapter, we will present a brief overview of the PAC model as framework for learning. We will see how a particular learning problem can be stated mathematically in this setting. We will then formally introduce the crucial idea of the VC dimension and proceed to show (via theorem 4) why we might expect this model to be a useful approach for handling prediction problems.

2.1 The PAC model

The PAC model of learning is built around the idea that each piece of data consists of an "unlabeled" element x of some set X, and a label y from a label set L. In this thesis, we will assume there is a function $c : X \to L$ which dictates how label $y = c(x)$ is assigned to element $x \in X$. We will also assume there is a probability distribution P on X, according to which we may draw a random set of labeled sample points; our goal will be to approximate the function c based on these few labeled points.

As a simple example, suppose we have a fixed deterministic computer program z which plays chess. Then x could be another deterministic chess program, and $y \in L = \{\text{yes}, \text{no}\}$ could be whether or not this player x will beat player z in a game of chess. We may try to learn the relationship between x and y by studying past games and their outcomes, and use this to try to predict whether or not a newcomer will win (and perhaps indirectly compute the rating of player z!).

In more detail, we assume that the relationship we are trying to learn is a function, or **concept**, $c : X \to L$ which is a member of a set C. Whichever set

$C \subset L^X$ we choose is referred to as our **concept class**. We may also choose another set $\mathcal{H} \subset L^X$, whose members we refer to as **hypotheses**; the output of our algorithm will be a member of this set. A PAC algorithm is built around a specific concept class C; it assumes the data complies with some $c \in C$ and chooses a particular hypothesis $h \in \mathcal{H}$ based on labeled training data (x, y) randomly chosen from $X \times L$ according to probability distribution P. Between each pair $c \in C$ and $h \in \mathcal{H}$, we define a penalty function $error(c, h)$, which is usually a distance function based on probability distribution P. We will use

$$error(c, h) := P_x\big(c(x) \neq h(x)\big),$$

which is the probability of choosing a point x on which hypothesis $h(x)$ supplies the wrong label. This particular error function makes the most sense when L is a small set of labels. When $|L| = 2$, we say that we are considering a **classification learning problem**. When $L = \mathbb{R}$ or is otherwise infinite, we are considering a **regression learning problem**. In the latter case, when $L = \mathbb{R}$, a common choice of error function is

$$error(c, h) := \left(\int |c(x) - h(x)|^2 dP(x) \right)^{1/2},$$

although we will be dealing primarily with the classification case.

An algorithm is built to learn a *target concept* $c \in C$, and operates on input δ and ϵ by extracting as many pieces of labeled data as it needs until it can return a hypothesis $h \in \mathcal{H}$ such that

$$P(error > \epsilon) < \delta, \tag{2.1}$$

where $error = error(c, h)$ indicates how badly h is as an estimator of target c. Note that the learned hypothesis h is not necessarily a member of C. If concept class C allows such an algorithm, then we say that C is **PAC learnable**.

In many cases, such an algorithm may exist but run very slowly. We are interested in those concept classes which allow for reasonably fast learning. We say that concept class C is **poly-time PAC learnable** (as a single concept class) when it is PAC learnable in time polynomial in $1/\epsilon$ and $1/\delta$.

To be more abstract, we would like to allow for a capacity-increasing sequence of classes $C_1, C_2, \ldots$ so that a general algorithm may also effectively choose a concept from any of these classes. It is important that our dependence on this new size parameter is also polynomial. Traditionally, this parameter is

always the *VC dimension* d of class C_d, which has many natural properties as a measure of the capacity or size of a concept class; we will define and discuss this dimension momentarily. But first, we are ready to define a sequence of concept classes $C_1, C_2, \ldots$ with $VCdim(C_d) = d$ as being **poly-time PAC learnable** (as a sequence of concept classes) when each class C_d is PAC learnable in time polynomial in $1/\epsilon, 1/\delta, d$, and $rep(c)$, the size of a minimal representation of concept $c \in C$.

VC dimension

The *Vapnik-Chervonenkis dimension*, or simply *VC dimension* of a concept class is an integer measuring the capacity of the class. Intuitively, the VC dimension attempts to capture the capacity of the concepts in the class to model different patterns of the data. In order to formally define the VC dimension, we will first introduce some specific notation for *classification problems*.

Many learning algorithms focus on the case where we have but two labels: $|L| = 2$. We may choose $L = \{0, 1\}, \{\text{yes}, \text{no}\}, \{+, -\}$ or $\{-1, +1\}$, all of which are essentially the same for the purpose of learning — we will use $\{+, -\}$ in this thesis for its intuitive appeal. In this case, every concept c may also be considered a subset of X; i.e., $c_{\text{subset}} = \{x \in X |\ c(x) = +\}$. Accordingly, we may think of a concept class as a subset $C \subset 2^X$, where 2^X denotes the power set $2^X := \{Y \subset X\}$. As mentioned above, a *classification problem* is the challenge of learning a target concept from concept class C when $|L| = 2$.

Given a subset $Y \subset X$ and a concept class $C \subset 2^X$, we use $C|Y := \{c \cap Y | c \in C\}$ to denote the *restriction of C to Y*. Notice that $C|Y \subset 2^Y$, and hence could be considered as a concept class on Y. We say that C **shatters** Y when $C|Y = 2^Y$. A set of unlabeled data Y is thus shattered exactly when *any* choice of labels on the data is consistent with some concept in C.

Example of shattering Suppose we are working with ground set $X = \{x_1, x_2, x_3, x_4\}$ and concept class $C = \{c_1, c_2, c_3, c_4\}$, where $c_1 = \{x_3\}, c_2 = \{x_2, x_4\}, c_3 = \{x_2, x_3, x_4\}$, and $c_4 = \{x_1\}$. We can illustrate these sets with the following diagram of indicator functions:

X	$=$	x_1	x_2	x_3	x_4
c_1	$=$	$\{0$	0	1	$0\}$
c_2	$=$	$\{0$	1	0	$1\}$
c_3	$=$	$\{0$	1	1	$1\}$
c_4	$=$	$\{1$	0	0	$0\}$

In this setup, the set $Y = \{x_2, x_3\} \subset X$ is shattered since $C|Y = 2^Y$. More specifically,

$$C|Y = \{c_1 \cap Y, c_2 \cap Y, c_3 \cap Y, c_4 \cap Y\} = \{01, 10, 11, 00\} = 2^Y,$$

again using an indicator-function notation for the sets. Looking at the table, we can quickly see that this set $Y = \{x_2, x_3\}$ is shattered since the middle two columns, corresponding to Y, contain all possible combinations 00, 01, 10, and 11 among its rows. Similarly, we can also see that $\{x_3, x_4\}$ is also shattered by C.

On the other hand, the set $Y' = \{x_1, x_2\}$ is *not* shattered, since, from the two left columns, we can see that $C|Y' = \{00, 01, 10\}$, so that $\{11\}$ is missing; that is, $Y' \notin C|Y'$.

Also notice that it is impossible here for any set Y with $|Y| > 2$ to be shattered since there are simply not enough elements in C. If $|Y| = m$ is shattered, then it must be the case that

$$size(C) \geq size(C|Y) = 2^m.$$

□

The **VC dimension** $d = VCdim(C)$ of concept class C is defined as the size $d = |Y|$ of any largest subset $Y \subset X$ which is shattered by C. It follows that one must take the following two steps in order to prove that $VCdim(C) = d$ for any particular concept class C:

1. show the existence of a subset $Y \subset X$ of size d which is shattered by C, and

2. show that *any* Y of size $d + 1$ is not shattered by C.

Example of learning with VC dimension = 1 To illustrate some of these ideas, we present a simple example. Suppose we are passively observing

a monkey playing with a straight branch of nonuniform density. The monkey repeatedly places the branch on a narrow point, attempting to balance the branch. Each time he does so, the branch falls down. We observe whether it falls to the right or the left, and we would like to predict in the future which way the branch will fall.

To model this situation, let $X = [0, 1]$, so that any unlabeled data point $x \in X$ represents a point on the branch at which the monkey will attempt to balance it; if the branch is 1 meter long, then x indicates how far from the left end the point is (so 0 is the left end of the branch and 1 is the right). Our concept class will be given by $C = \{c | c = [0, \theta] \text{for some } \theta \in X\}$. Any particular concept $c = [0, \theta]$ represents the prediction that the branch will fall to the right if $x \in c$ — that is, concept c predicts that the branch falls to the right iff $x \leq \theta$.

What is the VC dimension of C? Well, the singleton $\{1/2\}$ is shattered since $c_1 = [0, 1]$ contains it while $c_2 = [0, 1/3]$ does not. However, for any doubleton $\{x, y\}$ with $x < y$, we see that it's impossible for $y \in c$ while $x \notin c$ by the nature of our concept class. Hence $VCdim(C) = 1$. This makes intuitive sense since we are learning a single parameter, which we may think of as the center of gravity of the branch.

Is this concept class poly-time PAC learnable? We claim that it is. We will use the following algorithm: after watching the monkey perform m trials (m for *m*onkey trials), we assume that θ is exactly the rightmost point from any trial in which the branch fell to the right.

Let θ_t represent the true value; and θ_g is our guess. By our algorithm, it must be the case that $\theta_g \leq \theta_t$. This is an i.i.d. monkey which always chooses $x \in X$ independently according to probability distribution P on X. Choose θ_ℓ as the leftmost point for which $P([\theta_\ell, \theta_t]) \leq \epsilon$. Notice that

$$P([\theta_g, \theta_t]) > \epsilon \implies \theta_g < \theta_\ell \implies \forall i, 1 \leq i \leq m, x_i \notin [\theta_\ell, \theta_t],$$

where x_i is the ith point at which the monkey attempted (and failed) to balance the branch. If this last event occurs, we'll say that *region* $[\theta_\ell, \theta_t]$ *is missed*. Since

$$P([\theta_\ell, \theta_t] \textit{ is missed}) \leq (1 - \epsilon)^m \leq e^{-\epsilon m},$$

we can conclude that

$$P(error > \epsilon) \leq P([\theta_\ell, \theta_t] \textit{ is missed}) \leq e^{-\epsilon m}, \tag{2.2}$$

where $error := P([\theta_g, \theta_t])$ is the probability of guessing incorrectly on a future trial.

By choosing $m > (1/\epsilon)\log(1/\delta)$, we can be sure that $e^{-\epsilon m} < \delta$, so that (2.2) becomes

$$P(error > \epsilon) < \delta,$$

which suffices to show that this problem is eventually PAC learnable. If our monkey is diligent, and performs trials at a regular pace, then the number of observations we need, m, determines our time complexity. Since m is certainly polynomial in $1/\epsilon$ and $1/\delta$, we see that our concept class C is also poly-time PAC learnable.

(In this example, it was important that a monkey be performing the trials rather than a human being: if we ourselves were performing the experiments, then we could control which unlabeled elements x we sampled, and hence perform a more systematic search for θ.)

□

Generalization error bounds

Thus far we have defined the key terms and explained the essential methodology behind PAC learning. Now we will connect these ideas with the general strategy of *structural risk minimization* — the idea that learning a predictive hypothesis is a combination of accuracy on the training data along with a limit on the amount of freedom we have in choosing a hypothesis. In particular, we will present a fundamental inequality due to Vapnik [49] in the context of classification problems.

Our data points $x \in X$ are drawn according to probability distribution P, and label $y \in \{+1, -1\}$ is associated with point x according to the target concept $c \subset X$ via

$$y = \triangle_c(x) := \begin{cases} +1 & \text{if } x \in c \\ -1 & \text{otherwise.} \end{cases}$$

In this setting, our goal is to learn a hypothesis $h \in \mathcal{H}$ which provides a small error term in comparison with target concept c. Here we will refer to our error term, defined above by

$$error(h) := P\big(c(x) \neq h(x)\big),$$

as the **test error** or **generalization error** of hypothesis h. So, $error(h)$ is the probability of receiving a random point x which is misclassified by hypothesis h.

Recall that a learning algorithm, which is ignorant of the distribution P and target concept c, is given labeled training data $T \subset X \times \{\pm 1\}$ drawn according to distribution P and labeled according to concept c. We need just one more definition: define the **training error** of our learned hypothesis h by

$$\widehat{error}(h) := \frac{\#\{(x,y) \in T : y \neq \triangle_h(x)\}}{|T|}.$$

Thus the training error is simply the fraction of training data which are misclassified by our learned hypothesis h. Of course, if the points in T are consistent with some hypothesis $h \in \mathcal{H}$, then $\widehat{error}(h) = 0$.

We are ready to state:

Theorem 4 (Vapnik) *Suppose we have drawn the training data points $T \subset X \times \{\pm 1\}$ according to distribution P on X, and chosen a hypothesis $h \in \mathcal{H}$. Let $d = VCdim(\mathcal{H})$ and $m = |T|$. Then, with probability at least $1 - \delta$,*

$$error(h) \leq \widehat{error}(h) + \sqrt{\frac{1}{m}\left(d \log\left(\frac{2em}{d}\right) + \log\left(\frac{4}{\delta}\right)\right)}.$$

This bound gives us a specific relationship between the key terms $error(h)$, $\widehat{error}(h)$, and $VCdim(\mathcal{H})$ in the mold of structural risk minimization. In particular, we see that we achieve a better bound from theorem 4 when our VC dimension can be decreased without sacrificing too much training error.

The next two chapters consider the relationship between sparsity, small VC dimensions, and better learning guarantees.

Chapter 3

Learning as sample compression

In this section we examine some very general evidence supporting the idea that learning from a very small number of examples, chosen carefully, is often a good idea.

It is easy to notice that many concept classes intuitively promote learning from only a sparse subset of the training data. In the monkey example above, it was only one particular point x which we used to determine our guess θ_g for the center of gravity representing our hypothesis. In the case of support vector machines, only a few pieces of training data are ultimately used to construct the final hypothesis.

This general idea is referred to as *sample compression*. In particular, a *sample compression scheme* is a pair of functions, say *compress* and *decompress*. Given training data $T \subset X \times L$, *compress* chooses a subset τ of T so that the hypothesis that is output by *decompress* on τ is consistent with all of T. We add the restriction that the set output by *compress* may not exceed a fixed integer k, which we call the *size of the compression scheme*. We'll give more formal definitions of these ideas (in two different flavors) below.

Work demonstrating that the best sample compression size of a concept class correlates very closely with its VC dimension has been lead by Manfred Warmuth and co-authors Sally Floyd [21] and Dima Kuzmin [36]. In 1995, Warmuth and Floyd proved that any class of VC dimension d allowed a labeled sample compression scheme of size d. In a *labeled* sample compression scheme, the *compress* function is allowed to preserve the labels of the data it keeps. On the other hand, an *unlabeled* sample compression scheme may only keep a subset of X, not of $X \times L$, as the output of its *compress* function. This more challenging type of scheme was the subject of Warmuth and Kuzmin's 2006

paper, in which they proved such schemes of size $d = VCdim(C)$ always exist when C is a finite class and satisfies the property of being of maximum size with respect to its VC dimension. Such a class is called *maximum*, and must abide by many analytically useful properties.

3.1 Labeled sample compression schemes

In this section, we'll show that every concept class of finite VC dimension d has a labeled compression scheme of size at most d. In order to do so, we first introduce some very useful ideas, and corresponding notation, for finding particular "subclasses" of a concept class. We next present Warmuth and Floyd's general compression scheme.

Third, we'll explain how any compression scheme corresponds with a learning algorithm, and mention some generalization error bounds based on sparsity due to Thore Graepel, Ralf Herbrich, and John Shawe-Taylor [27]. As we'll see later, these bounds also support the idea of optimizing a linear separator based on a direct minimization of the number of support vectors used.

Finally, we'll notice that labeled compression schemes are a factor of about 4.4 away from achieving optimal compression, although an *unlabeled* compression scheme (explored in the next section), would be exactly optimal.

3.1.1 Basic properties of concept classes

Given a base set X, recall that a *concept class* C is any fixed subset $X \subset 2^X$.

Given $A \subset X$, we will define

$$\begin{aligned} C - A &:= \{c - A | c \in C\}, \quad \textit{and} \\ C \ominus A &:= \{c - A | \forall a \subset A, c \cup a \in C\}. \end{aligned}$$

Thus $C - A$ is exactly $C|(X - A)$; and the concepts $c \in C \ominus A$ are those members of $C - A$ which may be extended in any way a on A while keeping $c \cup a$ in the concept class C. Thus any single member of C^A may be extended to a subset of C which shatters A. Both $C - A$ and $C \ominus A$ are concept classes on the slightly smaller set $X - A$. When $A = \{x\}$ is a singleton, we'll write $C \ominus x$ for $C \ominus \{x\}$ and $C - x$ for $C - \{x\}$.

The following fact will be useful to know:

Property 5 *If* $VCdim(C) = d$ *then*

- $|A| \leq d \implies VCdim(C \ominus A) \leq d - |A|$, *and*
- $|A| > d \implies C \ominus A = \emptyset$.

Proof. If $C \ominus A$ is nonempty and shatters $B \subset X - A$, then C must shatter $A \cup B$, by definition of $C \ominus A$. Since A, B are disjoint,

$$d = VCdim(C) \geq |A| + |B| = |A| + VCdim(C \ominus A) \tag{3.1}$$

$$\implies VCdim(C \ominus A) \leq d - |A|.$$

Thus if $|A| > d$ and $C \ominus A$ were nonempty, then (3.1) would dictate that $d \geq |A|$, which is a contradiction here. Hence whenever $|A| > d$, it must be that $C \ominus A$ is empty.

□

It is relatively trivial to observe that

$$VCdim(C|A) \leq VCdim(C) \tag{3.2}$$

for any $A \subset X$, $C \subset 2^X$.

3.1.2 Sauer's Lemma

One might wonder if there are any size restrictions which follow from having a finite VC dimension. In fact, a sharp upper bound has been discovered independently by Saharon Shelah [40], Norbert Sauer [39], and Vapnik and Chervonenkis [48] (Sauer's name is most regularly associated with the fact). We'll present a few key observations leading up to this bound.

First we'll see a decomposition which is a generally useful way to work inductively with concept classes. We'll introduce the notation $1C \ominus x := \{c \cup \{x\} : c \in C \ominus x\}$. Usually, we think of $C \ominus x$ as a subset of $X - \{x\}$, so we'll use the notation $0C \ominus x := C \ominus x$, and "think of" $0C \ominus x \subset 2^X$ instead of just $2^{X-\{x\}}$. Finally, we'll define

$$\mathsf{tail}_x(C) := \{c \in C : c - x \notin C \ominus x\}.$$

For any $c \in C$, let $c' = c \,\Delta\, x$. Here we are using the *symmetric difference* operator on sets, which is defined for $A, B \subset X$ as

$$A \,\Delta\, B \quad := \quad (A - B) \cup (B - A).$$

Thus

$$c \,\Delta\, x \quad = \quad \begin{cases} c - \{x\} & \text{if } x \in c \\ c \cup \{x\} & \text{otherwise.} \end{cases}$$

If $c' \in C$, then $c - x \in C \ominus x$. Otherwise, $c \in \mathsf{tail}_x(C)$. Thus we arrive at the following decomposition:

Property 6 *For any concept class $C \subset 2^X$,*

$$C = 0C \ominus x \;\uplus\; 1C \ominus x \;\uplus\; \mathsf{tail}_x(C).$$

Here, the "dot union" notation emphasizes that the sets being unioned are all disjoint.

It is clear that the different versions of $C \ominus x$ are all identical in size:

$$|C \ominus x| = |0C \ominus x| = |1C \ominus x|.$$

Furthermore, we have $C - x = 0C \ominus x \uplus \mathsf{tail}_x(C)$. Thus we arrive at the useful

Corollary 7 *For any concept class $C \subset 2^X$,*

$$|C| = |C \ominus x| + |C - x|.$$

Now we're ready to state

Lemma 8 (Sauer's Lemma) *If X is finite and $C \subset 2^X$ has VC dimension d, then*

$$|C| \leq \binom{|X|}{\leq d}.$$

Here we are using the notation $\binom{|X|}{\leq d} = \sum_{i=0}^{d} \binom{|X|}{i}$.

Proof.

We'll use induction on $|X|$. The bound clearly holds in the base case $|X| = 0$.

Property 5 tells us that $VCdim(C \ominus x) \leq d - 1$; we also have that $VCdim(C - x) \leq d$ (which matches (3.2)). Thus, by corollary 7 and induction,

$$\begin{aligned} |C| &= |C \ominus x| + |C - x| \\ &\leq \binom{|X| - 1}{\leq d - 1} + \binom{|X| - 1}{\leq d} \\ &= \binom{|X|}{\leq d}, \end{aligned}$$

the last equality following by simple properties of the binomial coefficients. $\square$

Is this bound tight? We'll show that it is indeed. In order to do so, let's define, for any set X and integer $d \geq 0$,

$$\left\{ \begin{matrix} X \\ k \end{matrix} \right\} \quad := \quad \{i \subset X : |i| = k\},$$

and

$$\left\{ \begin{matrix} X \\ \leq k \end{matrix} \right\} \quad := \quad \{i \subset X : |i| \leq k\}.$$

Now let $C_d := \left\{ \begin{matrix} X \\ \leq d \end{matrix} \right\}$, the set of all subsets of X of size at most d. Choose any $i_k \in \left\{ \begin{matrix} X \\ k \end{matrix} \right\}$. Then clearly any i_d is shattered in C_d, while no i_{d+1} can be shattered since the set $i_{d+1} \notin C_d$. Thus $VCdim(C_d) = d$, and clearly $|C_d| = \binom{|X|}{\leq d}$, exactly the upper bound given by Sauer's lemma.

3.1.3 Maximum concept classes

It turns out that those classes which render Sauer's lemma an equality have many useful properties conducive to analysis. We call a set $C \subset 2^X$ a **maximum concept class** of VC dimension d if $VCdim(C) = d$ and, for any finite $Y \subset X$, we have $size(C|Y) = \binom{|Y|}{\leq d}$. Thus a maximum concept class is one which has as many concepts as possible without exceeding VC dimension d.

There are two immediate questions any good mathematician should ask when first encountering this idea:

Question 9 *Is it the case that any $C \subset 2^X$ of finite VC dimension is a subset of some maximum class of the same VC dimension?*

Question 10 *Are maximum concept classes interesting? That is, do there exist many maximum concept classes besides the trivial example $\left\{ \begin{matrix} X \\ \leq d \end{matrix} \right\}$?*

We will see that the answers are "no" and "yes," respectively.

In the first question, an affirmative answer would allow us to restrict our attention to maximum concept classes and still derive generally-applicable results when the property in question can be extended to subsets. Unfortunately, there do in fact exist many concept classes which are not a subset of a maximum class of the same VC dimension. We'll see an example in a moment, and later on give some insight into these other classes which avoid a maximum extension.

The second question also results in the "interesting" answer — that is, there are indeed many nontrivial maximum concept classes. In fact, even when we allow for a certain natural type of isomorphism between concept classes, we'll see that there is still variety of structure.

There are many nice characterizations of finite maximum concept classes. Let's use the notation $C \in \mathsf{Mm}(X, d)$ to denote that $C \subset 2^X$ is a maximum concept class of VC dimension d.

We'll begin with perhaps the most surprising characterization, which Warmuth credits to Emo Welzl (unpublished, mentioned in [21] and [36]).

Property 11 *For finite X,*

$$C \in \mathsf{Mm}(X, d) \quad \Leftrightarrow \quad VCdim(C) = d \ \ and \ \ size(C) = \binom{|X|}{\leq d}.$$

We will prove this last property and the next one together.

Property 12 *Suppose $C \in \mathsf{Mm}(X, d)$ for some finite X, and $A \subset X$.*

1. *If $|A| \leq d$, then $C \ominus A \in \mathsf{Mm}(X - A, d - |A|)$.*
2. *If $|A| \geq d$, then $C|A = C - \bar{A} \in \mathsf{Mm}(A, d)$.*

Here, $\bar{A}$ denotes $X - A$.

Thus $C \ominus A$ is a singleton whenever $|A| = d$.

Proof. We will prove both of these last two properties simultaneously using induction on $|X|$. If $|X| = 1$, all the properties are trivial (note that, by convention, $VCdim(\emptyset) = -1$, which keeps things consistent here).

Suppose we have shown all of the above statements for any base set Y with $|Y| < |X|$, and now we are examining $C \in \mathsf{Mm}(X, d)$. To show property 12, it will suffice to demonstrate it when $A = \{x\}$ is a singleton (larger A are covered by the inductive assumption).

It is clear that $VCdim(C - x) \leq d$, and by property 5, $VCdim(C \ominus x) \leq d - 1$. Now we can use corollary 7 along with Sauer's lemma to see that

$$|C| = |C \ominus x| + |C - x| \leq \binom{|X| - 1}{\leq d - 1} + \binom{|X| - 1}{\leq d} = \binom{|X|}{\leq d}.$$

Yet by definition of maximum, we know that $|C| = \binom{|X|}{\leq d}$, so that either $|C \ominus x| < \binom{|X|-1}{\leq d-1}$ or $|C - x| < \binom{|X|-1}{\leq d}$ would lead to a contradiction. Thus by our inductive assumption on property 11, we know that $C \ominus x$ and $C - x$ are maximum, confirming property 12.

We must also confirm property 11 on X. Let us suppose only that $C \subset 2^X$ has $VCdim(C) = d$ and $size(C) = \binom{|X|}{\leq d}$. Then by property 12, for any $Y \subset X$ with $|Y| \geq d$, we see that $size(C|Y) = \binom{|Y|}{\leq d}$, as required for C to be maximum. If $Y \subset Z$ with $|Z| = d$, then $\binom{|Z|}{\leq d} = 2^|Z|$, so that $C|Z$ is exactly 2^Z, and then clearly $C|Y = (C|Z)|Y$ must also be shattered, and the equality $C|Y = \binom{|Y|}{\leq d}$ is still satisfied. Thus C meets the definition of being maximum, confirming property 11. □

Property 12 can also be extended to infinite X.

Property 13 *Suppose $C \in \mathsf{Mm}(X, d)$ and $A \subset X$.*

1. *If $|A| \leq d$, then $C \ominus A \in \mathsf{Mm}(X - A, d - |A|)$.*
2. *If $|A| \geq d$, then $C|A = C - \bar{A} \in \mathsf{Mm}(A, d)$.*

Proof. We begin by verifying part *2*. If $|A| \geq d$, and we have some finite set $Y \subset A$, then clearly

$$size((C|A)|Y) = size(C|Y) = \binom{|Y|}{\leq d},$$

since C is itself maximum. Thus $C|A$ also meets the definition of being in $\mathsf{Mm}(A, d)$.

Part *1*. involves just a little more work. We make two observations. First, that

$$\begin{aligned} \binom{n-1}{\leq k-1} + \binom{n-1}{\leq k} &= \binom{n}{\leq k} \\ \implies 2\binom{n-1}{\leq k-1} &\leq \binom{n}{\leq k} \\ \implies 2^m\binom{n-m}{\leq k-m} &\leq \binom{n}{\leq k}. \end{aligned} \tag{3.3}$$

The last implication can be shown using induction on m.

Now suppose $|A| \leq d$ and $Y \subset X - A$ is finite. Our second observation is that

$$size(C|(Y \uplus A)) \geq size(C \ominus A|Y) \cdot 2^{|A|}, \tag{3.4}$$

which follows since each $c \in (C \ominus A)|Y$ corresponds to $2^{|A|}$ concepts in $C|(Y \uplus A)$.

So if there exists a finite set $Y \subset X - A$ with $size((C \ominus A)|Y) < \binom{|Y|}{\leq d-|A|}$, then by these observations it follows that

$$size(C|(Y \uplus A)) < 2^{|A|}\binom{|Y|}{\leq d - |A|} \leq \binom{|Y| + |A|}{\leq d},$$

which contradicts that $C \in \mathsf{Mm}(X, d)$. Hence it must be that $C \ominus A$ is also maximum of VC dimension $d - |A|$, as claimed. $\square$

Another very simple characterization of maximum concept classes can be derived using the following general result proven independently by Noga Alon [1] and Peter Frankl [22]. We will use the idea of a **hereditary set** $B \subset 2^X$ which by definition satisfies

$$a \subset b \ \wedge \ b \in B \quad \Longrightarrow \quad a \in B.$$

Theorem 14 (Alon-Frankl) *If $C \subset 2^X$ for some finite set X, then there exists a hereditary set $B \subset 2^X$ such that $|A| = |B|$ and, for any $Y \subset X$, $size(B|Y) \leq size(C|Y)$.*

In particular, notice that $VCdim(B) \leq VCdim(C)$.

We are now ready to prove

Theorem 15 *For finite X,*

$$C \in \mathsf{Mm}(X, d) \ \ \Leftrightarrow \ \ VCdim(C) = d \ \& \ \forall i \in \left\{ {X \atop d} \right\}, i \text{ is shattered in } C.$$

Proof. One direction is easy: if C is maximum, then certainly everything in $\left\{ {X \atop d} \right\}$ must be shattered.

For the other direction, suppose that $VCdim(C) = d$ and that everything in $\left\{ {X \atop d} \right\}$ is shattered. By theorem 14, we can find a hereditary B with $|B| = |C|$ and which does not increase the VC dimension. Since B is hereditary, it must be the case that $B \supset \left\{ {X \atop \leq d} \right\}$. If there is some $b \in B$ with $|b| > d$, then

that set would be shattered and $VCdim(B) > d$. Hence $B = \left\{ {X \atop \leq d} \right\}$, so that $|C| = |B| = \binom{|X|}{\leq d}$, and by property 11, it must be that $C \in \mathsf{Mm}(X, d)$. $\square$

A curious result follows easily from this last, as was first noticed by Sally Floyd in her thesis [20].

Theorem 16 *For finite X,*

$$C \in \mathsf{Mm}(X, d) \quad \Leftrightarrow \quad \bar{C} \in \mathsf{Mm}(X, |X| - d - 1).$$

Here $\bar{C} := 2^X - C$, not to be confused with the set of complements of the individual elements of C.

The essential idea behind this fact is the idea of a **forbidden label** of class C, which is a pair (L, S) with $L \subset S \subset X$ such that $L \notin C|S$. Set L is the label itself, while S is the set on which this label resides.

The proof of theorem 16 will be much easier if we first prove

Lemma 17 *For any $C \subset 2^X$ and $\ell_J \subset J \subset X$,*

$$(\ell_J, J) \text{ is a forbidden label of } C \quad \Leftrightarrow \quad \ell_J \in \bar{C} \ominus \bar{J}.$$

Proof of Lemma 17.

$$\begin{aligned}
&(\ell_J, J) \text{ is a forbidden label of } C \\
&\quad \Leftrightarrow \quad \ell_J \notin C|J \\
&\quad \Leftrightarrow \quad \forall K \subset \bar{J},\ \ell_J \uplus K \notin C \\
&\quad \Leftrightarrow \quad \forall K \subset \bar{J},\ \ell_J \uplus K \in \bar{C} \\
&\quad \Leftrightarrow \quad \ell_J \in \bar{C} \ominus \bar{J}.
\end{aligned}$$

$\square$

From here it is easy to see that

Corollary 18 *If $C \in \mathsf{Mm}(X, d)$ and $J \in \left\{ {X \atop d+1} \right\}$, then the forbidden label (ℓ_J, J) is unique and $\bar{C} \ominus \bar{J}$ is a singleton.*

The uniqueness follows directly from a cardinality argument:

$$size(C|J) = \binom{d+1}{\leq d} = 2^d - 1,$$

and there is exactly one subset of J, specifically ℓ_J, missing from $C|J$.

Proof of Theorem 16. It will suffice to prove the $\Longrightarrow$ direction; the other follows by symmetry. We will use theorem 15.

First let's see that $VCdim(\bar{C}) < |X| - d$. Consider an arbitrary set $I \in \left\{ {X \atop |X|-d} \right\}$. Since $C \ominus \bar{I}$ is nonempty, there is (by the lemma) a forbidden label on I for $\bar{C}$. Thus no set of size $|X| - d$ is shattered by $\bar{C}$.

Next we'll check that every set of size $|X| - d - 1$ is shattered by $\bar{C}$. By the corollary, every $J \in \left\{ {X \atop d+1} \right\}$ has a unique forbidden label ℓ_J. Then $\bar{C} \ominus \bar{J}$ is nonempty, so that $\bar{J}$ is shattered by $\bar{C}$. Applying theorem 15, this completes the proof. $\square$

3.2 Unlabeled sample compression schemes

Above, we have studied the idea of a *labeled* sample compression scheme, in which the compressed training data was allowed to retain their labels. At the same time, we noticed that the size of a maximum concept class was *exactly* the same as the number of possible "compressions" of size at most d — this curious size equivalence prompts the question: what if, in the compression phase of our sample compression, we were to discard the labels of our data?

Hence we arrive at the idea of an *unlabeled* sample compression scheme. As above, we must find a pair (f, g) of functions — one (f) for compressing an arbitrary training set, and the other (g) for decompressing this small subset into a hypothesis for our original training data. The difference is that $f : \mathcal{P}(X \times L) \to \mathcal{P}(X)$ now has a range of subsets of unlabeled data, i.e., subsets of X, rather than a labeled subset of $X \times L$ (here, as above, L is our set of labels).

In [36], Dima Kuzmin and Manfred Warmuth conjecture that any concept class of VC dimension d allows an unlabeled sample compression scheme of size d. They support this claim by proving it for finite maximum concept classes.

However, in their definition of an unlabeled sample compression scheme, they *do not require* the reconstructed hypothesis, $h = g(\text{compressed data})$, to be a member of the original concept class. The curious reader may wonder if this is not a spurious exclusion from the definition. Doesn't it seem most natural to always learn a concept within our given concept class? We will defend their original "unfaithful" compression scheme definition by showing

that a more restrictive definition would certainly disallow the existence of a compression scheme for some concept classes.

In particular, we will define a **faithful unlabeled sample compression scheme** for concept class C as a pair of functions (f, g) such that (f, g) is an unlabeled sample compression scheme for C, *and* the range of the reconstruction function g is restricted to concept class C.

We will present an infinite concept class of VC dimension 2 which does not allow a faithful unlabeled sample compression scheme.

Our concept class is the set C of *positive halfspaces* in the plane. Concept $c \subset \mathbb{R}^2$ is a **positive halfspace** iff $c = \{(x_1, x_2) : x_2 \geq mx_1 + b\}$ for some pair $(m, b) \in \mathbb{R}^2$. It is clear that any pair in $\mathbb{R}^2$ can be shattered by C. We also claim that no triple is shattered, so that the VC dimension of C is in fact 2. Given any point x we will denote its separate coordinates by (x_1, x_2). Suppose we have any three points $\{x, y, z\}$ with $x_1 \leq y_1 \leq z_1$. If y is on or above the line between x and z, then no positive halfspace h will have $x, z \in h$ but not $y \in h$. On the other hand, if y is below the line between x and z, then no h will exclude both x and z while including y. That is, no triple in the plane is shattered by C, and its VC dimension is exactly 2.

Theorem 19 *The set of positive halfspaces in the plane does not allow a faithful unlabeled sample compression scheme of size 2.*

Proof. We present a proof by contradiction. Begin by assuming that there is a faithful unlabeled sample compression scheme for C on $\mathbb{R}^2$. Then we have a map $g : \mathcal{P}(\mathbb{R}^2) \to C$ which associates a positive halfspace with each subset of $\mathbb{R}^2$ of size at most 2; this is the "decompression" map from the compression scheme. For any $A \subset \mathbb{R}^2, |A| \leq 2$, we may define $\ell : \mathcal{P}(A) \to \{+, -\}^A$ as the labeling on A induced via g by a particular compression (=subset) of A. That is, for any $\alpha \subset A$, $\ell(\alpha)$ assigns label $+$ to point $a \in A$ if $a \in g(\alpha)$; or $-$ otherwise.

Given any set $A \subset \mathbb{R}^2$ of at most two points, notice that f must be injective on $\mathcal{P}(A)$; otherwise it cannot be onto, and not all labelings of this sample will be reconstructable. We will show that there must exist a pair A such that ℓ is not injective on $\mathcal{P}(A)$ — this will be our contradiction.

Given a hypothesis $h \in C$, we will write $h(x) = +$ and $h(x) = -$ to indicate that $x \in h$ and $x \notin h$, respectively. Given any $x \in \mathbb{R}^2$, let h_x denote $g(\{x\})$, the hypothesis represented by the singleton $\{x\}$ in the compression scheme.

Let $h_0 = g(\emptyset)$. Then

$$\forall x, \quad h_x(x) \neq h_0(x). \tag{3.5}$$

This must be the case by the injectivity of ℓ on the power set of $\{x\}$.

Choose point u above the boundary of h_0 so that there exists $\epsilon_1 > 0$:

$$||u - x|| < \epsilon_1 \implies h_0(x) = +. \tag{3.6}$$

Since $h_u(u) = -$ and we're dealing with positive halfspaces, there must be some $\epsilon_2 > 0$ so that

$$||u - x|| < \epsilon_2 \implies h_u(x) = -. \tag{3.7}$$

Let $\epsilon_3 = \min(\epsilon_1, \epsilon_2)$. If there is a point v with $||u - v|| < \epsilon_3$ and $h_v(u) = -$, then both h_u and h_v give all-negative labelings to the set $\{u, v\}$, contradicting the injectivity of ℓ on $\mathcal{P}\{u, v\}$.

Hence it must be that

$$||u - x|| < \epsilon_3 \implies h_x(u) = +. \tag{3.8}$$

It is a property of any halfspace h that if $a_1 < b_1 < c_1$, $a_2 = b_2 = c_2$, $h(a) = +$ and $h(b) = -$, then $h(c) = -$ as well. We can apply this general fact in our setting to notice that

$$(u_1 < v_1 < w_1) \;\&\; (u_2 = v_2 = w_2) \;\&\; (||u - v|| < \epsilon_3) \quad \implies \quad h_v(w) = - \tag{3.9}$$

since (3.8) gives $h_v(u) = +$ and (3.5) and (3.6) give $h_v(v) = -$.

Thus we may choose t with $u_1 < t_1 < u_1 + \epsilon_3$ and $u_2 = t_2$. Again using (3.5) and (3.6), we can see that $h_t(t) = -$, so there must be a neighborhood around t in which h_t is always $-$. From this neighborhood choose ϵ_4 such that

$$||t - x|| < \epsilon_4 \implies h_t(x) = -. \tag{3.10}$$

And from here we may choose s with $u_1 < s_1 < t_1$, $||t - s|| < \epsilon_4$, and $u_2 = s_2 = t_2$. Then (3.9) tells us that $h_s(t) = - = h_s(s)$ while (3.10) tells us that $h_t(s) = - = h_t(t)$. That is, we have a set $\{s, t\}$ of size two on whose power set ℓ is not injective, which is the aforementioned contradiction. □

In this chapter, we have explored the question of how much sparsity in learning may be possible in a very general setting. We reviewed the properties of maximum concept classes, whose surprisingly stringent structure reveals to

us a method of sparse learning via sample compression schemes. We have also seen that unlabeled sample compression schemes, although possible for any *finite* maximum concept class, are not always possible in the infinite case when required to be *faithful*.

We are now ready to look more specifically at the role of sparsity in the more concrete setting of linear classifiers.

Chapter 4

Sparsity for linear predictors

In this chapter we will provide strong evidence that sparsity is a good indicator of prediction quality. Specifically, we will give an upper bound on the VC dimension of C_k, the concept class of linear classifiers with at most k nonzeros (we define this class C_k formally in a moment). As reviewed after the proof, this bound will allow us to achieve better bounds on the generalization error of a linear hypothesis.

4.1 A bound on the VC dimension

The main result of this chapter, and indeed one of the primary results of this thesis, is

Theorem 20 *If* $3 \leq k < \frac{9}{20}\sqrt{n}$*, then*

$$VCdim(C_k) < 2k \lg(n).$$

Here, $C_k = \{c_u : ||u||_0 \leq k, u \in \mathbb{R}^n\}$ where $c_u = \{x \in \mathbb{R}^n : x \cdot u \geq 0\}$. Also, $\lg(x)$ is defined as $\log_2(x)$, the base 2 logarithm.

The heart of the proof is captured by the following

Lemma 21 *If* $d = VCdim(C_k)$ *and* $k \geq 3$*, then*

$$d - k \lg(d) < k \lg(n).$$

In order to prove this lemma, we will use a clever application of Sauer's lemma. Define

$$\delta_{\geq 0}(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ 0 & \text{otherwise} \end{cases}$$

on scalars x, which may be extended coordinate-wise to the mapping $\delta_{\geq 0} : \mathbb{R}^n \to \{0,1\}^n$ on vectors as well. Now we'll introduce the notation $SP(X) = \{\delta_{\geq 0}(x) | x \in X\}$ for any subset $X \subset \mathbb{R}^n$; these are the *sign patterns* taken on by set X.

Lemma 22 *If A is an $m \times k$ matrix, then*

$$\#SP(\mathsf{col}(A)) \leq \binom{m}{\leq k} \leq \left(\frac{em}{k}\right)^k.$$

Here, $\#SP(X)$ denotes the size of the set $SP(X)$.

Proof of lemma 22.

Let concept class C be the set of concepts c_u, where $c_u = \{x \in \mathbb{R}^k : x \cdot u \geq 0\}$ (as before), except that we allow u to be any vector in $\mathbb{R}^k$ now. It is well-known that $VCdim(C) = k$. (Those readers used to linear classifiers with an added constant — of the form $\{x : x \cdot u + b \geq 0\}$ — might expect that $VCdim(C) = k+1$, but since we are excluding the '$+b$' from our concepts, we effectively achieve this slightly smaller VC dimension.)

Now let $R \subset \mathbb{R}^k$ denote the set of rows of A, and let

$$D = C|R = \{c_u \cap R | r \in \mathbb{R}^k\}.$$

Then by Sauer's lemma (stated as lemma 8 above),

$$|D| \leq \binom{|R|}{\leq VCdim(D)} \leq \binom{m}{\leq k}.$$

Finally, we note that there is a bijection between the set D and $SP(\mathsf{col}(A))$ since each element $\delta_{\geq 0}(Au)$ of $SP(\mathsf{col}(A))$ is the indicator function for $c_u \cap R \in D$, and vice versa. Id est, $\#SP(\mathsf{col}(A)) = |D| \leq \binom{m}{\leq k}$, as claimed.

To conclude, we notice that, when $k \leq m$,

$$\left(\frac{k}{m}\right)^k \binom{m}{\leq k} \leq \sum_{i=0}^{k} \left(\frac{k}{m}\right)^i \binom{m}{i} \leq \sum_{i=0}^{m} \left(\frac{k}{m}\right)^i \binom{m}{i}$$

$$= \left(1 + \frac{k}{m}\right)^m \leq e^{m(k/m)} = e^k.$$

This gives us our final inequality: that

$$\binom{m}{\leq k} \leq \left(\frac{em}{k}\right)^k.$$

This completes lemma 22. □

Note that we may interpret this last lemma in the following interesting geometric way: any $k-$dimensional subspace of $\mathbb{R}^m$ may pass through at most $\binom{m}{\leq k}$ quadrants, where "passing through a quadrant" is defined to correspond with our particular type of sign patterns. This result actually provides a special case of the Milnor-Thom or Warren's theorem for linear functions which is stronger than any version the author is aware of. For example, this bound is a factor of 4^k times better than the version of Warren's theorem stated in the very well-written paper [38].

We are now ready for the

Proof of lemma 21.

Consider k as fixed. Suppose we are given m points in $\mathbb{R}^n$, which we will label by concepts in C_k. We can compile these points as rows in the $m \times n$ matrix A.

For each $i \in \left\{ {n \atop k} \right\}$, let $U_i = \{u \in \mathbb{R}^n : \text{supp}(u) \subset i\}$. Notice that if $u \in U_i$, then $\delta_{\geq 0}(Au)$ is the vector of labels assigned to these points by classifier c_u — equivalently, $\delta_{\geq 0}(Au)$ is the indicator function of concept c_u on the rows of A. Also notice that when $u \in U_i$, we have $Au = A_i u_i$, where A_i is the $m \times k$ submatrix of A keeping only the $i-$columns (we could write $A_i = A(:, i)$), and $u_i \in \mathbb{R}^k$ are the $i-$coordinates of u.

Then, for any fixed i,

$$\#SP(Au : u \in U_i) = \#SP(A_i u_i : u \in U_i) \leq \left(\frac{em}{k}\right)^k.$$

Thus, since $\{u : ||u||_0 \leq k\} = \cup_{i \in \left\{ {n \atop k} \right\}} U_i$, we have

$$\#\text{labelings of the rows of } A \text{ by } C_k = \#SP(Au : ||u||_0 \leq k)$$

$$\leq \sum_{i \in \left\{ {n \atop k} \right\}} \left(\frac{em}{k}\right)^k = \binom{n}{k} \left(\frac{em}{k}\right)^k \leq \left(\frac{emn}{k}\right)^k.$$

So if $2^m > (emn/k)^k$, then no set of size m can be shattered by C_k since there cannot possibly be enough labelings / sign patterns on these points. Hence, if $d = VCdim(C_k)$, then

$$2^d \leq \left(\frac{edn}{k}\right)^k \quad \Longrightarrow \quad d \leq k \lg \left(\frac{edn}{k}\right) \quad \Longrightarrow$$

$$d - k\lg(d) \leq k\lg(en/k) < k\lg(n)$$

when $k > e$. This confirms the lemma. □

At last we may now give the **Proof of theorem 20.**

This proof is essentially a series of balanced inequalities which connect from lemma 21 to the statement of the theorem.

Let us begin by noting that, for any $b > 0$,

$$\frac{b}{2} + \frac{1}{11} > \lg(b),$$

which we will use in the form $b - \lg(b) > b/2 - 1/11$.

Now

$$2k < \frac{9}{10}\sqrt{n} \implies \lg(2k) < \lg(9/10) + \frac{1}{2}\lg(n).$$

Let $b = \lg(n)$, and observe that $\lg(9/10) < -1/11$ so that

$$\lg(2k) < \frac{b}{2} - \frac{1}{11} < b - \lg(b)$$

$$\implies \lg(2bk) < b \implies$$

$$k < \frac{bk}{\lg(2bk)}. \tag{4.1}$$

Let $z = d - kb$. Lemma 21 tells that $z \leq k\lg(d) = k\lg(z + kb)$, which we can rewrite as

$$k \geq \frac{z}{\lg(kb + z)}.$$

It is easy to check that $f(z) = z/\lg(kb + z)$ is increasing for $z > 0$ whenever $kb > 1$. Thus, if $z \geq kb$, then

$$k \geq \frac{z}{\lg(kb + z)} \geq \frac{kb}{\lg(2kb)},$$

which would directly contradict (4.1). So it must be the case that $z < kb$, which means that $d = kb + z < 2kb = 2k\lg(n)$, as claimed. □

Now that we have formally shown our bound on the VC dimension of sparse linear classifiers, we would like to remind the reader of the connection with the probability of misclassification on future data points.

In particular, let us recall the inequality given in theorem 4 above:

$$error(c) \leq \widehat{error}(c) + \sqrt{\frac{1}{m}\left(d\log\left(\frac{2em}{d}\right) + \log\left(\frac{4}{\delta}\right)\right)}, \tag{4.2}$$

which holds with probability at least $1-\delta$, where m is the number of training points used, c is our learned concept in class C and $d = VCdim(C)$.

If we let C_n denote the concept class of all linear classifiers of the form $c_u = \{x \in \mathbb{R}^n : x \cdot u \geq 0\}$ in $\mathbb{R}^n$, then it is not too hard to see that $VCdim(C_n) = n$. Thus we have achieved a significant asymptotic improvement in showing that $VCdim(C_k) \leq 2k\lg(n)$. We therefore can best utilize the generalization error bound of theorem 4 precisely when we can find a concept c_ℓ in a class of *maximal sparsity* which does not incur too high a cost of training error.

4.2 The regression case

So far we have been focused on the learning scenario in which $|L| = 2$; that is, we have been *classifying* our data points into exactly two possible categories. However, as we will see in the chapters to come, the algorithms presented in this thesis are most naturally considered in the *regression* case — that is, when $L = \mathbb{R}$, and each data point x is labeled by an arbitrary real value y.

The initial elements of our learning model presented in §2.1 remain the same. Indeed, we still assume that there is a particular concept $c : X \to L = \mathbb{R}$ relating each data point x to its label $y = c(x)$. We also assume that there is a probability distribution P on X by which our training and test points are drawn. And our goal is still to achieve a bound of the form

$$P(error > \epsilon) < \delta,$$

in time polynomial in $1/\epsilon$ and $1/\delta$.

The first key difference we will note is that it is no longer clear how to associate a VC dimension with a concept class. Furthermore, the generalization bound (4.2) does not necessarily hold any longer — indeed, our classification-based definition of the *error* terms don't even make sense in the regression case!

So we begin by defining the generalization error between a target concept $c_0 : X \to \mathbb{R}$ and a hypothesis $h : X \to \mathbb{R}$ as

$$error(h) := \int (c_0(x) - h(x))^2 dP(x).$$

Then the natural empirical version of this, our regression version of training error on the points $(x, y) \in T \subset X \times \mathbb{R}$, is

$$\widehat{error}(h) := \sum_{(x,y)\in T} (h(x) - y)^2.$$

We can also extend the definition of the VC dimension: given a set C of functions $X \to \mathbb{R}$, let

$$I(C) := \left\{ \{(x, y) : \big(c(x) - y\big)^2 \geq \alpha\} : c \in C, \alpha \geq 0 \right\}.$$

We can treat $I(C)$ as a concept class of $X \times \mathbb{R}$; thus we will treat $d = VCdim(I(C))$ as the dimension of C itself.

Now that we have established the necessary terminology, we can state a regression-case generalization error bound (again due to Vapnik). We must make the additional assumption that there exist real numbers a and b such that, for target function $c_0 : X \to \mathbb{R}$, we have $a \leq (c(x) - c_0(x))^2 \leq b$ for any $x \in X, c \in C$. Then for any $c \in C$, with probability at least $1 - \delta$,

$$error(c) \leq \widehat{error}(c) + (b - a)\sqrt{\frac{1}{m}\left(d \log\left(\frac{2em}{d}\right) + \log\left(\frac{4}{\delta}\right)\right)}. \tag{4.3}$$

How does this bound relate to the linear regression algorithms presented later in this thesis? We first observe that, if F_n denotes the set of linear regression functions on $\mathbb{R}^n$, then there is a constant κ such that $d = VCdim(I(F_n)) \leq \kappa n$ for any $n \in \mathbb{N}$. Replacing this bound in our proof of theorem 20, we see that $VCdim(I(F_k)) \leq 2\kappa k \lg(n)$, where F_k is now the set of linear functions with support of size at most k. These are precisely the sparse linear prediction functions which will be found by our algorithms.

When we restrict our base set X to a reasonably bounded domain, such as $X = \{x \in \mathbb{R}^n : ||x|| < A\}$ for suitable A, then many time series will permit finite parameters a, b so that we may utilize inequality (4.3). As in the classification case, our generalization bound here is optimized when we can maximize sparsity without sacrificing too much accuracy to training error.

In this chapter we have demonstrated a new upper bound on the VC dimension of sparse linear classifiers. We have also reviewed the connection between lower VC dimensions and better generalization error bounds; this motivates the efforts of the following chapters, in which we explore algorithmic techniques of discovering and tracking sparse linear patterns among a set of evolving time series.

Part III

Sparsity in practice: Time series algorithms

In this part, we'll see how sparsity can be used to find patterns in time series which are likely to act as accurate predictors.

In his paper "Just Relax," Joel Tropp [46] offers three fundamental motivations of the search for sparsity:

1. It is known or assumed that the underlying data has a sparse linear representation.

2. Many representations are viable, but density is expensive — this includes such cases as compression or building a system whose time complexity is a function of density.

3. As we have seen in the previous chapter, sparsity can help us prove better bounds on the predictive power of our discovered patterns. We could state this as the principle of *Occam's razor* (compare [8] and [15]).

If the last chapter explored *why* we might want to search for sparse linear patterns, this chapter then focuses on *how* we may discover these patterns.

In §6, we examine questions about when and to what degree we might expect a matrix to be sparsifiable. The main result of this section states that almost every matrix is *completely unsparsifiable* — that is, it cannot be sparsified at all beyond what is achieved by simple Gaussian elimination. To counter this negative result, we continue to show that poorly-conditioned matrices can be easily sparsified to a relatively high degree if we allow some bounded error in the process.

In §7 we explore the problem of combinatorially sparsifying an arbitrary matrix, whereas in §8, we consider probabilistic and approximate attacks. We'll see that even approximating the optimal solution is NP-hard. At the same time, we'll present a series of algorithms which take advantage of the incremental nature of time series to efficiently update a sparse null space structure. These algorithms take advantage of the fact that L^1- minimization problems are very likely to result in sparse solutions.

Chapter 5

Introduction

As in previous sections of this paper, we will consider m values of a time series as a column vector a_i in $\mathbb{R}^m$, and we'll identify a set of n time series with the columns of an $m \times n$ matrix A. Some columns in A allow a linear dependence relation iff there is a nontrivial identity of the form

$$\lambda_1 a_1 + \lambda_2 a_2 + \ldots + \lambda_n a_n = 0$$

between them, for some coefficient column vector $\lambda \neq 0$. We may write this succinctly as

$$A\lambda = 0.$$

Clearly, we are interested in exactly those vectors λ in the null space of A. Suppose we have a basis for $N(A)$, the null space of A. Here are two natural questions we may ask at this point:

1. What information is still left "unknown" to us?

2. What predictive power have we gained by our null space basis?

Let's briefly give one answer to (1), after which most of this chapter is devoted to exploring question (2). The following elementary fact gives us a concise view of what is "left out" when we extract the null space:

Property 23 *If $m \times n$ matrix A has $n \times c$ full null matrix N with orthonormal columns, then we may decompose*

$$A = LV^T$$

where L is $m \times r$, full rank, and lower triangular, with $r = \mathsf{rank}(A)$; and V is $n \times r$ so that $(V\ N)$ is unitary. (More specifically, we may ensure that L is in lower column echelon form.)

A full null matrix, as we originally defined in §1.2, has full rank equal to c, the corank of A.

We may interpret property 23 in many ways. One may think of the columns of V as the independent time stamps which are linearly combined to produce each row of A. From this perspective, each row of L acts as a coefficient vector indicating how to combine the columns of V into a row of A. Since L is lower triangular, we may interpret the nth column of V as the nth independent component of the timestamps.

Another point of view would be to think of the rows (in $\mathbb{R}^r$) of V as a compressed representation for each time series (column) in A. In this case, the columns of L may be considered as a basis for all the time series. Further, since L is lower column echelon, it is straightforward to convert any column of A into its compressed form as a row of V. It is also interesting to note that if the rank of A does not increase, then matrix V may persist while only L evolves over time.

Hence one method of prediction might be to compute L and V, and then predict future values of the new rows of L, from which we may extract the future rows of A as well. We'll take a preliminary look at answering question (2) first, and then compare these two approaches. As the astute reader may anticipate, we will argue that using the decomposition in property 23 is less desirable than simply tracking the null space of A or an approximation thereof.

Before moving on, we pause a moment to give the

Proof of property 23. It is well known that any matrix has a QR decomposition. In our case, we write

$$A^T = QR,$$

where Q is $n \times n$ and unitary; and R is $n \times m$ and upper row echelon (slightly stronger than being upper triangular). Then $A = R^T Q^T$. As above, let $r = \mathsf{rank}(A) = \mathsf{rank}(R)$. Since R^T is in lower column echelon form, we know that the first r columns are linearly independent, while the remainder must be zero columns. Thus we can define L as the first r columns of R^T, and, correspondingly, V as the first r columns of Q. The result is that

$$A = LV^T,$$

as desired. In addition it must be the case that $\mathsf{col}(N) = \mathsf{null}(A)$, where N are the last $n - r$ columns of Q. $\square$

5.1 The predictive power of the null space

We now begin to build some motivation for the predictive power of the null space of matrix A as a set of columns of time series. Thus we are addressing question (2) above.

Let us rephrase the questions as:

Question 24 *What is the minimal amount of information we need about the future to reconstruct other time series values?*

More specifically, suppose we are allowed to "peek" at a subset of the coordinates of a, a future row of matrix A. Thus, we are given measurements from a subset of the time series, and we would like to predict the rest of the values based on this.

We will propose two answers. To give the first, we'd like to define a *generating set*. A set of time series $G \subset [n]$ (where $[n] = \{1, 2, \ldots n\}$ as above), forms a **generating set** with respect to null matrix N when, knowing the values of row a on coordinates G, we can reconstruct the full row a using the fact that $aN = 0$. Notice that a generating set might contain some redundancy. For example, setting $G = [n]$ trivially gives a generating set, but this case is not interesting. We really care about those generating sets which minimize redundancy. To capture this idea, we'll say that $G \subset [n]$ is a **free generating set** when any choice of values for $a(G) \in \mathbb{R}^{|G|}$ is consistent with exactly one row a such that $aN = 0$. Here, we use the notation $a(G)$ to indicate the values of row a on the coordinates indicated by G. Thus a *free generating set* is one which imposes no constraint whatsoever on our choice of $a(G)$, and hence contains no redundancy, yet still allows us to reconstruct all of the data points in row a from this subset.

We are now ready to state

Theorem 25 *Given $n \times c$ null matrix N so that $aN = 0$, $G \subset [n]$ is a generating set iff rows $N(\bar{G}, :)$ are linearly independent, where $\bar{G} = [n] - G$. Furthermore, G is a free generating set iff rows $N(\bar{G}, :)$ form a basis for $\mathsf{row}(N)$, the row space of N.*

Those familiar with the Matlab software package will immediately understand the $N(\bar{G},:) = N([n]-G,:)$ notation. For the rest of us: suppose N is an $n \times c$ matrix. Then, given $rows \subset [n]$ and $cols \subset [c]$, we write $N(rows, cols)$ to denote the submatrix of N from rows $rows$ and columns $cols$. The colon shorthand, as in $N(rows,:)$ indicates that $cols = [c]$; we are including all the columns.

Proof. These statements follow from elementary facts about solving linear systems via Gaussian elimination.

As a very brief reminder, we recall for the reader one expression of Gaussian elimination as a method of solving for vector x in the matrix equation $Ax = b$. In particular, we may use the decomposition $A = LU$, in which L is invertible, and U is psychologically in reduced row echelon form. This means that the column space of U is spanned by a subset, the *pivot columns*, which are independent columns from an identity matrix. Thus $Ax = b \Leftrightarrow Ux = L^{-1}b$, and the latter system is straightforward to solve by assigning arbitrary values to the *free coordinates* of x (those not corresponding to pivot columns in U), and then solving for the *basic coordinates* (of course, those which do correspond to pivot columns in U).

What we care about in this process is the fact that a subset of columns in A may be chosen as (a subset of the) pivot columns iff they are linearly independent. Moreover, these columns will be exactly the set of pivot columns (and not a strict subset) iff they are a basis for the column space.

Thus solving $aN = 0$ for row vector a, equivalent to solving $N^T a^T = 0$, can always be done given consistent coordinates $a(G)$ when columns $N^T(:,\bar{G})$ = rows $N(\bar{G},:)$ are linearly independent. Moreover, the coordinates $a(G)$ may be chosen arbitrarily exactly when rows $N(\bar{G},:)$ form a basis for the row space of N, as claimed. $\square$

Theorem 25 tells us exactly which sets are needed to reconstruct all of row a. Notice that if $|G| < n - c$ then $|\bar{G}| > c$, in which case $N(\bar{G},:)$ cannot possibly be linearly independent, so that G is not a generating set. In other words, *every generating set must have size at least* $n - c$. If $c \ll n$, then we need to be given most of the information in order to determine the rest.

Is there any way we could utilize an even smaller subset $G \subset [n]$? This goal leads us to our second answer to question 24. In order to see this answer, let's define a **generating set for coordinate** $i \in [n]$ as any subset $G \subset [n]$ such that, given $a(G)$, there is a unique value for a_i such that $a(G \cup \{i\})$ can be extended to a row a with $aN = 0$. We do not require uniqueness in the

other coordinates of a — just the ith.

Example 2 *Suppose we are given matrix*

$$A = \begin{pmatrix} 9 & -35 & 9 & 3 & 18 & 0 & 3 \\ 6 & -39 & 12 & 6 & 0 & 3 & 0 \\ 9 & -81 & 15 & 18 & 6 & 3 & 15 \\ -6 & -80 & 18 & 18 & 15 & 12 & 9 \\ 9 & -52 & 15 & 6 & 0 & 3 & 18 \end{pmatrix}.$$

From this, we can extract the null space (written in transpose here to save space)

$$N^T = \begin{pmatrix} 3 & 3 & 4 & 7 & 1 & 3 & 1 \\ 4 & 3 & 3 & 7 & 1 & 5 & 1 \end{pmatrix}.$$

Then, $G = \{1, 2, 3, 4, 5\}$ *is a free generating set since* $N(\bar{G}, :) = \begin{pmatrix} 3 & 5 \\ 1 & 1 \end{pmatrix}$ *is a basis for* $\mathsf{row}(N)$*; further, there is no smaller generating set (although there are others of the same size).*

However, letting $i = 3$*, we can also see that* $G = \{1, 6\}$ *is a generating subset for coordinate* i*. Indeed, if we let* $n^T = (1, 0, -1, 0, 0, 2, 0)$*, then* $n \in \mathsf{col}(N)$ *so that* $An = 0$ *and, in general, any row* a *must follow* $an = 0$*. (Also notice that, writing* a^i *for the* i^{th} *column of* A*, the identity* $a^3 = a^1 + 2a^6$ *is equivalent to* $An = 0$*.) Thus, if we are given the values of row* a *on only coordinates 1 and 6, we can uniquely determine the value of the third coordinate. This is interesting since the size of* G *here is much smaller than that of any generating set.*

□

This example gives us a hint toward characterizing minimally-sized generating sets for a particular coordinates: If we choose column vector n which satisfies

$$\begin{cases} \min & ||n||_0 \\ \text{s.t.} & An = 0 \ \& \ n_i \neq 0, \end{cases}$$

then the set $G_i = \text{supp}(n) - \{i\}$ is a minimally sized generating set for coordinate i.

Example 3 *We pause for a moment to note that the set* G_i *is not necessarily unique.*

For example, suppose we are given matrix $A = (1\ 1\ 1)$ *and we will receive another row* a *with the same nullspace. This is equivalent to saying that* a *is in the row span of* A*, so that all the coordinates of* a *will also be the same. Then clearly both singletons* $G = \{1\}$ *and* $H = \{2\}$ *are minimally-sized generating sets for coordinate 3.*

□

Chapter 6

Theory of sparse matrices

In this section we'll examine some theoretical results about matrices and their sparsity properties. It will be useful for later algorithmic work to justify the intuition that the product of sparse matrices is likely to also be sparse; this is quickly mentioned in §6.1. In §6.2, we give a simple lower bound for the sparsifiability of any matrix. In §6.3, we outline the main result of this section — the complete unsparsifiability of "most" matrices in a certain probabilistic sense. Finally, this last theorem is proven rigorously in §6.4.

6.1 Sparsity inequalities

The following series of inequalities captures a variety of levels of sparsity preservation in taking the product of two sparse matrices. We'll use the notation $||A||_{\tilde{0}}$ to indicate the number of nonzero entries in matrix A. The tilde, as in $\tilde{0}$, is used to avoid confusion with the *induced matrix almost-norm*,

$$(||A||_{\tilde{0}} \quad \neq) \qquad ||A||_0 = \sup_{||x||_2=1} (||Ax||_0) \, .$$

In this thesis, we will consistently work with $|| \cdot ||_{\tilde{0}}$ instead.

Theorem 26 *For any matrices A and B,*

$$||AB||_{\tilde{0}} \leq c(A)r(B) \leq ||c(A)|| \cdot ||r(B)|| \leq ||A||_{\tilde{0}} \cdot ||B||_{\tilde{0}}$$

where row vector $c = c(A)$ is given by $c_j = ||a^j||_0$ and column vector $r = r(B)$ is given by $r_i = ||b_i||_0$.

Proof. We'll demonstrate the inequalities from left to right.

For the first inequality, start by observing that the rank-one matrix $a^i b_i$ has exactly $c_i r_i$ nonzeros, since each nonzero entry in $a^i b_i$ corresponds in a bijective manner to a pair of nonzeros in a^i and b_i. Now recall the column-row expansion formula for matrix multiplication:

$$AB = \sum_i a^i b_i.$$

Since $||\cdot||_{\tilde{0}}$ obeys the triangle inequality, it follows that

$$||AB||_{\tilde{0}} \leq \sum_i ||a^i b_i||_{\tilde{0}} = \sum_i c_i r_i = c(A)r(B).$$

The second inequality is an instance of the Cauchy-Schwartz inequality.

For the last inequality, we will show that in fact $||c(A)|| \leq ||A||_{\tilde{0}}$. The analogous inequality also holds for $r(B)$ of course, and this suffices.

The claimed inequality is equivalent to

$$\sum_i c_i^2 \leq \big(\sum_i c_i\big)^2,$$

but this is obvious for any sequence (c_i) of nonnegative integers. $\square$

6.2 Gaussian elimination for sparsity

Gaussian elimination is usually considered as a process to be performed on the *rows* of a given matrix. Of course the same procedure may also be performed on columns; id est, we can perform traditional Gaussian elimination on the rows of A^T. We'll refer to this as the *column reduction* of A.

Hence for any $m \times n$ matrix A with $m \geq n$, we can always guarantee at least as many zeros as would be given by an $n \times n$ identity matrix. This is a sort of lower bound on the sparsifiability of an arbitrary matrix. The surprising result of the next section is that this bound is in fact tight!

Let us formally state the minimum sparsifiability guaranteed by column reduction. We will write $A \overset{c}{\sim} B$ to indicate that the equally-sized matrices A and B are column equivalent. Specifically, $A \overset{c}{\sim} B$ iff there exists an invertible matrix C so that $A = BC$.

Property 27 *Any $m \times n$ matrix A with rank r can be column reduced to a matrix $B \overset{c}{\sim} A$ with $||B||_{\tilde{0}} \leq (m-r+1)r \leq \frac{1}{4}(m+1)^2$.*

This follows since any rank r matrix B in reduced column echelon form (achievable by Gaussian elimination on the columns) includes an embedded $r \times r$ identity matrix I_r, which has $r(r-1)$ zeros. In addition, there are $n-r$ all-zero columns.

$$B = \begin{array}{cc} & \longleftarrow n \longrightarrow \\ \begin{array}{c} \uparrow \\ m \\ \downarrow \end{array} & \begin{pmatrix} I_r & 0 \\ X & 0 \end{pmatrix} \end{array}$$

Hence $||B||_{\tilde{0}} \leq mn - m(n-r) - r(r-1) = (m-r+1)r$. For any fixed m, this function is maximized when $r = (m+1)/2$, giving us our second bound (which is independent of r).

6.3 Most matrices are completely unsparsifiable

How tight is property 27? In many cases, it turns out to be completely optimal: For any positive integers m, n and $r \leq \min(m, n)$, there exist many $m \times n$ matrices A of rank r such that any $B \overset{c}{\sim} A$ has $||B||_{\tilde{0}} \geq (m-r+1)r$.

In fact, we will actually prove the more specific result. We'll call a rank r matrix A **completely unsparsifiable** iff, for any $B \overset{c}{\sim} A$, $||B||_{\tilde{0}} \geq (m-r+1)r$. (We include "completely" since we find it useful to leave the adjective pair "sparsifiable / unsparsifiable" open to interpretation, as is the word "sparse" for now.) We state and briefly discuss our result here, and give the proof in the next section after introduction some useful definitions and lemmas.

Theorem 28 *If every subsquare of $m \times n$ matrix A is nonsingular, where $m \geq n$, then A has rank n and is completely unsparsifiable. Moreover, the matrix $\binom{I}{A}$ is optimally sparse, where I is the $n \times n$ identity matrix.*

A submatrix $A(R, C)$ is a **subsquare** of A iff $|R| = |C|$.

We can now support the idea that "many" real matrices are completely unsparsifiable. Indeed, suppose that $P : \mathcal{B} \to [0, 1]$ is a probability distribution on the Borel-measurable subsets $\mathcal{B}$ of $\mathbb{R}$ such that $P(\{x\}) = 0$ for any $x \in \mathbb{R}$ (in this case P is sometimes referred as being *nonatomic*). Then if we choose

each element of A in an i.i.d. manner according to P, with probability 1, matrix A will be full rank, each subsquare will be nonsingular, and A will be completely unsparsifiable. Thus, almost every (in this probabilistic sense) matrix is completely unsparsifiable.

Of course, in practice, there may be underlying reasons to expect *a priori* that a matrix is sparsifiable; in general, we would need to justify any probability distribution over our matrices before concluding that they could not be sparsified. In addition, as we will explore below, we may often achieve further sparsity by allowing for some small error in the column space.

6.4 Equivalent conditions for sparsifiability

In this section we will begin by giving several useful lemmas and definitions which help characterize the condition of a matrix being optimally sparse. We then use this foundation to give a proof of theorem 28.

The ruminative reader will readily recall that a matrix A is called *optimally sparse* iff $\forall$ invertible matrices C, $||AC||_{\tilde{0}} \geq ||A||_{\tilde{0}}$. We could equivalently state this as $\forall D \overset{c}{\sim} A$, $||D||_{\tilde{0}} \geq ||A||_{\tilde{0}}$.

Let's start with

Lemma 29 *Matrix A is optimally sparse iff $\forall x \neq 0$,*

$$||Ax||_0 \geq \max_{i \in supp(x)} ||a_i||_0,$$

where a_i is the ith column of A.

This lemma essentially states that no linear combination of columns from A can increase the sparsity of any column used in the combination.

Proof. If there exists an x with $||Ax||_0 < ||a_i||_0$ and $x_i \neq 0$, then we may replace a_i with Ax to achieve a sparser column equivalent version of A. This justifies the $\Rightarrow$ half of the proof.

If A is not optimally sparse, then there exist matrices B, C so that $B = AC$, C is invertible, and $||B||_{\tilde{0}} < ||A||_{\tilde{0}}$. Let n be the number of columns in A. Then there must exist a permutation $\sigma : [n] \to [n]$ so that $\forall i$, entry $c_{i\sigma(i)} \neq 0$ in C; otherwise $\det(C) = 0$ by the permutation definition of the determinant. Hence we can apply the appropriate permutation matrix P_σ to C to arrive at $\tilde{C} = CP$ and $\tilde{B} = A\tilde{C}$ where all the diagonal entries of $\tilde{C}$ are nonzero.

Since $||\tilde{B}||_{\tilde{0}} = ||B||_{\tilde{0}} < ||A||_{\tilde{0}}$, there must exist a column b_i of $\tilde{B}$ which is sparser than its counterpart a_i of A. Let x be the ith column of $\tilde{C}$ so that $||b_i||_0 = ||Ax||_0 < ||a_i||_0$, and $x_i \neq 0$ since it is the ith diagonal entry of $\tilde{C}$. This justifies the $\Leftarrow$ half of the proof. $\square$

We are ready to define a particular type of submatrix which is useful in considering the sparsifiability of a matrix. Recall that, if A is an $m \times n$ matrix and $R \subset [m]$, $C \subset [n]$, then $A(R, C)$ denotes the submatrix on rows R and columns C. We will say that $A(R, C)$ is **row-inclusive** in A iff $\forall r \in [m]$, $A(r, C) \in \mathsf{row}(A(R, C)) \Rightarrow r \in R$; in other words, $A(R, C)$ contains all the rows of $A([m], C)$ in its own row span.

Recall that a set of columns form a *circuit* exactly when they are linearly dependent and every proper subset of them is linearly independent. We will say that $A(R, C)$ is a **candidate submatrix**, written $A(R, C) \lhd A$, iff the columns of $A(R, C)$ form a circuit and $A(R, C)$ is row-inclusive. The name "candidate submatrix" is inspired by the following facts, which reveal that there is a correspondence between each candidate submatrix and each column combination which might further sparsify a matrix.

Property 30 *For any $m \times n$ matrix A:*

1. *If $\exists\, x$ with $i \in supp(x)$ and $||Ax||_0 = m - p$, then $\exists\, A(R, C) \lhd A$ with $|R| = p$ and $i \in C \subset supp(x)$.*

2. *If $\exists\, A(R, C) \lhd A$, then $\exists\, x$ with $supp(x) = C$ and $||Ax||_0 = m - |R|$.*

In order to gain an intuitive foothold on the meaning of this property, we first present

Corollary 31 *The $m \times n$ matrix A is optimally sparse iff there is no candidate submatrix $A(R, C) \lhd A$ with $m - |R| < ||a_c||_0$ for some $c \in C$.*

Proof of the corollary. First suppose A is not optimally sparse. Then by lemma 29, there is an x with $i \in \text{supp}(x)$ and $||Ax||_0 < ||a_i||_0$. Let $p = m - ||Ax||_0$. By part 1 of the property we immediately have a candidate submatrix $A(R, C)$ with $|R| = p$ so that $m - |R| = m - p = ||Ax||_0 < ||a_i||_0$ with $i \in C$.

Now suppose we have $A(R, C) \lhd A$ with $m - |R| < ||a_i||_0$ and $i \in C$. By part 2 of the property, we can find a nonzero x with $||Ax||_0 = m - |R| < ||a_i||_0$

and $i \in \text{supp}(x)$. This means (by lemma 29) that A is not optimally sparse. $\square$

Now we are ready for the

Proof of property 30.

(Part 1) We have a vector x with $i \in \text{supp}(x)$ and $||Ax||_0 = m - p$. Let $b = Ax$ and $R = [m] - \text{supp}(b)$, so that $r \in R$ iff $b_r = 0$, and $||Ax||_0 = |\text{supp}(b)| = m - |R|$. Since its null space is nonempty, the columns of matrix $A(R, \text{supp}(x))$ are linearly dependent. It is an easy fact that if $i \in X$ and $A(R, X)$ is a linearly dependent set of columns, then there is a subset $C \subset X$ so that the columns of $A(R, C)$ form a circuit, and $i \in C$. Choose such a subset $C \subset \text{supp}(x)$. Let $\tilde{A} = A(R, C)$; we claim that $\tilde{A}$ is then the desired candidate submatrix — we already see that its columns form a circuit, so we simply need to show that it is row-inclusive.

By our choice of C, $\mathsf{rank}(\tilde{A}) = |C| - 1$ and we can choose a nonzero vector $\tilde{x} \in \mathsf{null}(\tilde{A})$. Then $\dim(\mathsf{null}(\tilde{x}^T)) = |C| - 1 = \dim(\mathsf{col}(\tilde{A}^T))$ and $\mathsf{col}(\tilde{A}^T) \subset \mathsf{null}(\tilde{x}^T)$ together imply that $\mathsf{col}(\tilde{A}^T) = \mathsf{null}(\tilde{x}^T)$. So if any row $\alpha = A(r, C)$ has $\alpha \in \mathsf{row}(\tilde{A})$, then $\alpha^T \in \mathsf{null}(\tilde{x}^T)$ so that $\alpha\tilde{x} = 0 \Rightarrow b_r = 0 \Rightarrow r \in R$; and thus $\tilde{A}$ is row-inclusive. So indeed $A(R, C) \lhd A$.

Our choice of R provides that $|R| = p$, and we have chosen C so that $i \in C \subset \text{supp}(x)$. So we have confirmed part 1 of the property.

(Part 2) Now suppose we are given $\tilde{A} = A(R, C) \lhd A$.

Since $\dim(\mathsf{null}(\tilde{A})) = 1$, we can choose a nonzero x with $\text{supp}(x) \subset C$ and $A(R, [n])x = 0$. The columns of $A(R, \text{supp}(x))$ are linearly dependent, so that by the minimal-dependency of the columns of $A(R, C)$, it must be that $\text{supp}(x) = C$.

If $\tilde{x} = x(C)$ so that no coordinate of $\tilde{x}$ is zero. As above, $\dim \mathsf{null}(\tilde{x}^T)) = |C| - 1 = \dim(\mathsf{col}(\tilde{A}^T))$ and $\mathsf{col}(\tilde{A}^T) \subset \mathsf{null}(\tilde{x}^T)$ together give us that $\mathsf{col}(\tilde{A}^T) = \mathsf{null}(\tilde{x}^T)$.

If row $\alpha = A(r, C) \notin \mathsf{row}(\tilde{A})$, then by the row-inclusiveness of $\tilde{A}$, $\alpha^T \notin \mathsf{col}(\tilde{A}^T) = \mathsf{null}(\tilde{x}^T)$, so that $\alpha\tilde{x} \neq 0$. That is, $A(r, C)\tilde{x} = 0$ iff $r \in R$ so that $\text{supp}(A([m], C)\tilde{x}) = [m] - R$. Then

$$||Ax||_0 = |\text{supp}(Ax)| = |\text{supp}(A([m], C)\tilde{x})| = m - |R|.$$

Thus x is indeed the vector guaranteed to exist by part 2. $\square$

We are now ready for the

Proof of theorem 28.

We will see that any $m \times n$ matrix A, with $m \geq n$ and no singular subsquares is of rank n, and, for any $C \overset{c}{\sim} A$, $||C||_{\tilde{0}} \geq (m-n+1)n$; that is, A is completely unsparsifiable.

We will show this by working with the matrix

$$B = \begin{pmatrix} I \\ A \end{pmatrix},$$

where I is the $n \times n$ identity matrix. We will see that B is optimally sparse. Notice that $||A||_{\tilde{0}} = mn$ since any zero entry would be a singular subsquare (of size 1). Thus $||B||_{\tilde{0}} = (m+1)n$. If there existed an invertible matrix T with $||AT||_{\tilde{0}} < (m-n+1)n$, then

$$||BT||_{\tilde{0}} = \left|\left| \begin{pmatrix} T \\ AT \end{pmatrix} \right|\right|_0 < n^2 + (m-n+1)n = (m+1)n,$$

contradicting that B is (as we have yet to show) optimally sparse. In other words, showing that B is optimally sparse suffices to verify that A itself is completely unsparsifiable.

First, it is easy to see that $\mathsf{rank}(A) = n$; otherwise any $n \times n$ subsquare of A would be singular.

Now, we will see that any candidate submatrix only preserves the number of nonzeros in B; none can reduce it. To proceed, let us suppose we have a candidate submatrix $B(R, C) \lhd B$. By property 30, we also have a vector x with $\text{supp}(x) = C$ and $||Bx||_0 = m + n - |R|$.

Let $R_I = R \cap [n]$; these are the rows in R which are in the identity matrix part of B. Let $R_A = R - R_I$; these are the rows in the "A part" of B.

Since the columns of $B(R_I, C)$ are dependent, it must be the case that $B(R_I, C) = 0$. This, along with the fact that the columns of $B(R, C)$ form a circuit, ensure that the columns of $B(R_A, C)$ also form a circuit. Thus $\mathsf{rank}(B(R_A, C)) = |C| - 1$.

We now claim that $|R_A| = |C| - 1$. If $|R_A| < |C| - 1$, then $\mathsf{rank}(B(R_A, C)) < |C| - 1$, which is not the case. On the other hand, if $|R_A| \geq |C|$, then any $|C| \times |C|$ subsquare of $B(R_A, C)$ must be singular, since it is rank-deficient — and this would contradict that every subsquare of A is nonsingular. The only consistent case is $|R_A| = |C| - 1$.

We saw above that that $R_I \subset \{r | B(r, C) = 0\}$. By the row-inclusiveness of any candidate submatrix, we also have that $\{r | B(r, C) = 0\} \subset R_I$, so that these two sets are in fact equivalent. And the largest all-zero submatrix in I with $|C|$ columns has exactly $n - |C|$ rows. That is, $|R_I| = n - |C|$.

Then

$$\begin{aligned} ||Bx||_0 &= m + n - |R| = m + n - (|R_I| + |R_A|) \\ &= m + n - (n - |C| + |C| - 1) = m + 1. \end{aligned}$$

In other words, we have seen that *no* candidate submatrix can strictly reduce the number of nonzeros in any column of B; by corollary 31, this suffices to demonstrate that B is optimally sparse. This completes the proof. $\square$

Chapter 7

Exact matrix sparsification

In this section, we'll focus on the problem of exact sparsification, as opposed to approximate or probabilistic attempts, which will be considered next (§8).

First we'll highlight some of the key reductions showing the NP-hardness of matrix sparsification, sparse null space, and related problems. These reductions will be expanded in the next section. We will then outline a pair of incremental algorithms to track the null space of an evolving matrix of time series. Although these algorithms in themselves can only provide weak guarantees of *generating* some sparsity, they can *maintain* sparsity well, which we will take advantage of in a modified version examined in the next section.

7.1 NP-hardness and other reductions

Here we'll look at the four problems Sparse Null Space (first defined in §1.2), Matrix Sparsification, Sparsest Vector, and Sparsest Independent Vector; we'll formally define each of these shortly. We'll see that the first two are equivalent, and that all of them are NP-hard.

The following figure depicts with solid arrows the reductions we will constructively show. The dashed arrow indicates an implied (though not explicitly described) reduction. All reductions are polynomial-time.

$$\begin{tikzcd}
 & & \text{Sparsest Independent Vector} \in \text{NP} \arrow[dd, dashed] \\
\text{Sparse Null Space} \arrow[r, leftrightarrow] & \text{Matrix Sparsification} \arrow[ur] & \\
 & & \text{Sparsest Vector} \arrow[ul]
\end{tikzcd}$$

Let's review the definition of our fundamental problem:

Sparse Null Space Given a matrix A, find an optimally sparse full null matrix N for A.

Our first pair of reductions will be between Sparse Null Space and

Matrix Sparsification Given a matrix A, find a matrix B which solves

$$\begin{cases} \min & ||B||_{\tilde{0}} \\ \text{s.t.} & B \overset{c}{\sim} A. \end{cases}$$

Sparse Null Space **reduces to** Matrix Sparsification

Suppose we want to find a sparse full null matrix for A. We can easily compute an arbitrary (non-sparse) full null matrix N' for A. For example, we may column reduce $\binom{A}{I}$ as follows:

$$\begin{pmatrix} A \\ I \end{pmatrix} \to \begin{pmatrix} U & 0 \\ V & N' \end{pmatrix},$$

where U has all nonzero columns (and is psychologically lower-triangular). Then it can be seen that N' is a full null matrix for A. More carefully: we may decompose $A^T = L\tilde{U}^T$, where L is invertible and $\tilde{U}^T$ is psychologically (up to a column permutation) in reduced row echelon form. Then $A = \tilde{U}L^T$, so that $\binom{A}{I}L^{-T} = \binom{\tilde{U}}{L^{-T}}$. From here it is clear that the rightmost $\mathsf{corank}(A)$ columns of $\tilde{U}$ are all zero, and that the corresponding columns of L^{-T} form a full null matrix N' for A.

From here, simply apply Matrix Sparsification to N' to arrive at output N, which will be an optimally sparse full null matrix for A.

Matrix Sparsification **reduces to** Sparse Null Space

If we want to instead sparsify an arbitrary matrix B, then we can begin by choosing an $m \times n$ matrix $\tilde{B}$ whose columns are a subset of those of B with $\mathsf{col}(\tilde{B}) = \mathsf{col}(B)$ and $\mathsf{rank}(\tilde{B}) = \mathsf{rank}(B) = n$; in other words, we throw out the redundant columns of B since these will become zero columns in the optimally sparse output.

Now we may, in polynomial time, compute an arbitrary (non-sparse) full null matrix A^T for $\tilde{B}^T$, so that $\tilde{B}^T A^T = 0$. Then A is $(m-n) \times m$ with $\mathsf{rank}(A) = m - n \Rightarrow \mathsf{corank}(A) = n = \mathsf{rank}(\tilde{B})$. Since $A\tilde{B} = 0$, we can conclude that $\tilde{B}$ is a full null matrix of A.

Hence if we solve Sparse Null Space on A, we will simultaneously be solving Matrix Sparsification on $\tilde{B}$. To derive the final answer for our original matrix B, we need only augment the correct number of zero columns.

Sparsest Vector **reduces to** Matrix Sparsification

This reduction and the next have both been pointed out earlier in [12]. First let's formally state the problem

Sparsest Vector Given a matrix A, find a nonzero vector $v \in \mathsf{col}(A)$ which minimizes $\|\|v\|\|_0$.

To show the reduction, suppose we have a matrix A and we want to find a nonzero vector $v \in \mathsf{col}(A)$ which minimizes $||v||_0$.

Let B be the optimally sparse matrix we get from solving Matrix Sparsification on A. If there is a nonzero vector $v \in \mathsf{col}(A) = \mathsf{col}(B)$ with $||v||_0 < ||b_i||_0$ for all columns b_i of B, then we could replace one of the columns of B (choosing carefully which column to replace to avoid shrinking $\mathsf{col}(B)$) to arrive at an even sparser matrix $B' \overset{c}{\sim} A$. But this contradicts optimal sparsity. Hence the sparsest column of B also solves Sparsest Vector.

Matrix Sparsification **reduces to** Sparsest Independent Vector

The stronger version of Sparsest Vector is

Sparsest Independent Vector Given a matrix A with independent columns $\{a_1, \ldots, a_n\}$, and an integer $0 \leq j < n$, find a nonzero vector $v \in \mathsf{col}(A)$ which minimizes $\|\|v\|\|_0$ while keeping the set $\{a_1, \ldots, a_j, v\}$ linearly independent.

If $j = 0$, this is exactly Sparsest Vector.

The following greedy algorithm solves Matrix Sparsification using Sparsest Independent Vector. It assumes A is of full column rank — if not, we can simply run Matrix Sparsification on $\tilde{A}$, a full rank subset of the columns of A with $\mathsf{col}(\tilde{A}) = \mathsf{col}(A)$; the other columns will become all-zero in the sparsified output. In the pseudocode below, the function $\mathsf{SIV}(A, j)$ returns a vector which solves Sparsest Independent Vector on inputs A and j; and b_j denotes the jth column of B.

Greedy Matrix Sparsification
(using Sparsest Independent Vector)

```
Let B := A
For j := 1 to n
    Let b_j :=SIV(B, j − 1)
Next j
Return B
```

In general, it is not clear that greedy algorithms ever guarantee finding a global optimum — however, in the case of matroids, this has been proven, as we shall recall in theorem 32 below.

Remember that a matroid can be characterized by its *ground set* X and a set $\mathcal{I} \subset 2^X$ of *independent sets* (whose intuition is analogous to that of linearly independent sets of vectors) which must obey the following rules:

- $\emptyset \in \mathcal{I}$,
- (hereditary property) $A \in \mathcal{I}$ & $B \subset A \implies B \in \mathcal{I}$, and
- (exchange property) $A, B \in \mathcal{I}$ with $|A| > |B| \implies \exists a \in A - B : B \cup \{a\} \in \mathcal{I}$.

Suppose that we would like to find a maximally-sized independent set $B \in \mathcal{I}$ for which $f(B)$ is maximized. Also suppose that $f(B) := \sum_{b \in B} f(b)$, so that f is characterized by its values on the ground set X. Let's call such a function an **additive weight function**.

Finally, let's formalize the **greedy algorithm** on a matroid $(X, \mathcal{I})$ as an incremental algorithm which at each step begins with set $B \in \mathcal{I}$ and augments

it to form a new independent set $B \cup \{x\}$ by choosing a particular $x \in X - B$. This element x is chosen simply so that it maximizes the value $f(x)$ over all elements of $X - B$ which maintain the independence of $B \cup \{x\}$. The algorithm terminates when the size of B is maximal.

Theorem 32 *Any run of the greedy algorithm on a matroid using an additive weight function produces a maximally-sized independent set B which satisfies $f(B) \geq f(C)$ for all other maximally-sized sets $C \in \mathcal{I}$.*

A proof (and other related information) can be found in §1.8 of [37].

In our case, we simply have to notice that the column space of any matrix A can form the ground set of a matroid in which the independent sets are exactly the linearly independent subsets of vectors. Clearly, the number of nonzero entries in our subset is an additive weight function, and our problem Sparsest Independent Vector lends itself immediately to the greedy algorithm.

7.1.1 Sparsest Independent Vector $\in$ NP

We will show that Sparsest Independent Vector can be solved in terms of the following bounded version of the problem:

$k-$Sparse Independent Vector Given matrix A with independent columns $\{a_1, \ldots, a_n\}$, integer j with $0 \leq j < n$, and $k \in \mathbb{N}$, find a nonzero vector $v \in \mathsf{col}(A)$ so that $\|v\|_0 \leq k$ and $\{a_1, \ldots, a_j, v\}$ is linearly independent. If no such v exists, return "none."

Clearly we can solve Sparsest Independent Vector on an $m \times n$ matrix A with at most m calls to $k-$Sparse Independent Vector (just start with $k = 1$ and proceed until we don't get a "none" reply).

Let's find an NP algorithm to solve $k-$Sparse Independent Vector on an $m \times n$ matrix A with $\mathsf{rank}(A) = n$. First, nondeterministically choose an arbitrary submatrix $\tilde{A} = A(R, C)$ of A. We want to check the following properties of $\tilde{A}$ — if any property is not met, this branch of the nondeterministic algorithm terminates (returns "none"):

1. Check that $C \not\subset [j]$
2. Check that $m - |R| \leq k$
3. Check that $\tilde{A} \lhd A$

We can check each property in polynomial time. If $C \not\subset [j]$, then we are guaranteed that any linear combination involving all the vectors in C will be independent of the first j vectors $\{a_1, \ldots, a_j\}$; otherwise we would have a linear dependence among columns.

If the other two conditions are also met, then part 2 of property 30 ensures us of the existence of our desired vector v. To compute the actual vector, we can simply solve $\tilde{A}\tilde{x} = 0$ for some nonzero $\tilde{x}$ and return $v = A([m], C)\tilde{x}$.

Certainly any returned v will satisfy the requirements. But how do we know this algorithm is guaranteed to find such a v if one exists? This time refer to part 1 of property 30 to see that any existing v which depends on some a_i with $i > j$ has a candidate submatrix $\tilde{A} = A(R, C) \lhd A$ with $i \in C$ which will allow us to compute a vector as sparse as v and still depending on a_i.

7.1.2 Sparsest Vector is NP-hard

Since Sparsest Independent Vector $\in$ NP, we need only show that Sparsest Vector is NP-hard in order to (non-constructively) prove that the former is reducible to the latter in polynomial time.

Larry Stockmeyer is given credit for the original proof of this fact, although he himself never published it. A version of his proof can be found in [23].

7.2 Algorithms

In this section, we'll consider a series of algorithms designed to solve Sparse Null Space. First in §7.2.1 we enumerate and briefly analyze three previous algorithms: *back substitution*, the *turnback algorithm*, and an algorithm which assumes the input matrix satisfies the *Haar condition*, given by Alan Hoffman and Tom McCormick in [30]. Then in §7.2.2 we will present some new ideas focused on taking advantage of the incrementally changing nature of time series data.

7.2.1 Previous algorithms

Back Substitution and Column Reduction

Perhaps the most naive approach to solving Sparse Null Space is to simply find a full null matrix and column reduce it. Another simple method, which we will refer to as *back substitution,* would be to use Gaussian elimination to find

a set of independent null vectors, trying to induce sparsity by including many zeros in the free variables. We will see here how much sparsity can be expected from these routines (a minimal amount) and that these two techniques are in fact different perspectives of essentially the same approach.

The linear algebraic reader is likely to be very familiar the content of the next few paragraphs, in which we carefully describe the classical method for solving the standard matrix equation $Ax = b$. We justify this brief exposition by using it to show the equivalence between back substitution and column reduction, as well as quantifying the relatively weak guarantees which can be provided by either of these methods.

Let's describe the back substitution technique in some detail. We are interested in finding a sparse full null matrix for the $m \times n$ matrix A. If $A = LU$, where L is invertible and U is in reduced row echelon form, then x solves $Ux = 0$ iff it solves $Ax = 0$. Recall that a matrix U is in reduced row echelon form iff

$$\left.\begin{array}{l} \bullet \text{ there is an increasing sequence } P = \{p_1, \ldots, p_r\} \subset [n] \text{ of} \\ \quad \text{column indices so that column } u_{p_i} = e_i, \text{ the } i^{\text{th}} \text{ column of} \\ \quad \text{the identity matrix; and} \\ \bullet \text{ for } i \leq r,\ j < p_i \implies u_{ij} = 0; \text{ and} \\ \bullet \text{ for all } i > r,\ u_{ij} = 0. \end{array}\right\} \quad (7.1)$$

The set of columns indexed by P are the *pivot columns*, which correspond to the *basic variables* in x, while the other variables are referred to as *free variables*. Notice that $|P| = r = \mathsf{rank}(U)$ since the pivot columns span $\mathsf{col}(U)$.

The back substitution method consists of using the equation $Ux = 0$ to find a sequence $\eta_1, \ldots, \eta_c$ of independent vectors with $U\eta_k = 0\ \forall k$. We find columns η_k in such a way that forces them to each contain at least $c - 1$ zeros.

Specifically, we assign $\eta_k(F) = e_k$, where $F = [n] - P$ are the coordinates of our free variables, and e_k denotes the kth column of the identity matrix. We may then use the equation $u^i x = 0$ to solve for coordinate p_i of x for $1 \leq i \leq r$, where u^i is the ith row of U. This is possible since $u_{ij} = 0$ whenever $j < p_i$, so that we need only know the values x_j for $j > p_i$ in order to solve for x_{p_i} itself.

In the following pseudocode, the notation $(u^i - j^{\text{th}})$ represents the ith row of U with the jth coordinate removed. We use this notation to indicate that the product $(u^i - j^{\text{th}})x$ is well defined even if x has not yet been defined on coordinates j or j' for which $u_{ij'} = 0$.

Back Substitution

Decompose $m \times n$ matrix $A = LU$
Choose $F := \{f_1, \ldots, f_c\}$ ($c = \mathsf{corank}(A)$)
    so that $f_i \in F$ corresponds to the ith free variable in U
And $P := \{p_1, \ldots, p_r\}$ ($r = \mathsf{rank}(A) = n - c$)
    so that $p_i \in P$ corresponds to the ith pivot column in U
For $k := 1$ to c
    Let $x(F) := e_k$
    For $i := r$ down to 1
        Let $x_{p_i} := -(u^i - {p_i}^{\mathsf{th}})x$
    Next i
    Let $\eta_k := x$
Next k
Return $N := \left(\eta_1 \,\middle|\, \eta_2 \,\middle|\, \cdots \,\middle|\, \eta_c \right)$

One thing which becomes clear in this algorithm is that x_{p_i} is defined only in terms of x_j for $j > p_i$; this follows since $u_{ij} = 0$ for $j < p_i$. This means that

$$(x_j = 0 \;\forall\; j > p_i) \implies x_{p_i} = 0,$$

so that, when $x = \eta_k$, $x(f_\ell) = 0 \;\forall\; \ell > k$, and

$$x_j = \eta_{jk} = 0 \;\forall\; j > f_k. \tag{7.2}$$

We will now use this observation to show

Property 33 *Suppose the $n \times c$ matrix $N = (\eta_{ij})$ is the output of a run of* back substitution *on the $m \times n$ matrix U. Define matrix $\hat{N} = (\hat{\eta}_{ij})$ by setting $\hat{\eta}_{ij} = \eta_{n-i,j}$; in other words, we are just swapping each pair $\{i, n-i\}$ of rows in N to arrive at $\hat{N}$.*

Then $\hat{N}$ is in reduced column echelon form.

Proof. Recall that matrix $\hat{N}$ is in reduced column echelon form iff $\hat{N}^T$ is in reduced row echelon form. This translates, from definition (7.1), to

- $\exists$ increasing sequence $\{p_1, \ldots, p_c\} \subset [n] :\; \forall j$, row $\eta^{p_j} = e_j^T$,

- $i < p_j \implies \hat{\eta}_{ij} = 0$, and
- $j > c \implies \hat{\eta}_{ij} = 0$.

First we claim that letting $\hat{p}_j = f_j$ suffices for the first bullet, where the increasing sequence $F = \{f_1, \ldots, f_c\}$ are the free variables of U. In order to verify this, let's also define $p_j = n - \hat{p}_j$. By the above algorithm, we can clearly see that column $\eta_k(F) = e_k$, so that the submatrix $N(F, :)$ is the $c \times c$ identity, and thus row

$$\hat{\eta}^{\hat{p}_j} = \eta^{p_j} = \eta^{f_j} = e_j^T,$$

as claimed.

For the second bullet, we can use observation (7.2) to see that

$$i < \hat{p}_j \implies n - i > n - \hat{p}_j = f_j \overset{(7.2)}{\implies} \eta_{(n-i)j} = 0 \implies \hat{\eta}_{ij} = 0,$$

as required.

Finally, the third bullet is trivial since there no columns in N with $j > c$. Thus we have confirmed that $\hat{N}$ is indeed in reduced column echelon form. □

We are now ready to evaluate the sparsity achieved by these methods.

Property 34 *If an $n \times c$ full null matrix N is found by either back substitution or column reduction, then*

$$||N||_{\tilde{0}} \leq (n - c + 1)c.$$

This property follows from the fact that any N generated in this way must contain a $c \times c$ identity matrix, which has $c(c-1)$ zeros, so that $||N||_{\tilde{0}} \leq nc - c(c-1) = (n - c + 1)c$. (Note that this bound is just a special case of property 27.)

A slightly more careful analysis allows us to see

Theorem 35 *If $n \times c$ matrix $\hat{N}$ of rank c is in reduced column echelon form with pivot rows $\hat{p}_1, \ldots, \hat{p}_c$, then*

$$||\hat{N}||_{\tilde{0}} \leq \left(\sum_i p_i \right) - \binom{c}{2},$$

where $p_i = n - \hat{p}_i$.

Retaining the notation from the proof of property 33, this means that back substitution on U gives

$$||N||_{\tilde{0}} \leq \left(\sum_i f_i\right) - \binom{c}{2},$$

where f_i is the ith free variable in U.

How useful is this bound? Unless we can provide some bounds on the values for $p_1, \ldots, p_c$, it is no better than property 34. In particular, if $p_i = n - c + i$ then $\sum p_i = (n-c)c + \binom{c+1}{2}$ so that

$$\sum p_i = (n-c)c + \binom{c+1}{2} - \binom{c}{2} = (n-c+1)c.$$

How efficient are these bounds? They are tight – it is not too hard to find matrices which achieve the worst case. In the case of back substitution, any matrix of the form $U = (I\ X)$ works, where I is the $m \times m$ identity and X is any $m \times (n-m)$ matrix with no zeros; that is, performing back substitution on such a U will result in an N with $(m+1)(n-m)$ nonzeros, which is equal to $(n-c+1)c$ here since $c = \mathsf{corank}(U) = n - m$. In the case of column reduction, we may simply notice that $N = \binom{I}{X}$ is in reduced column echelon form, where I is the $c \times c$ identity and $(n-c) \times c$ matrix X has no zeros.

We will consider these techniques and bounds as a minimum level of sparsity which future algorithms should strive to improve upon.

The Turnback Algorithm

The Turnback Algorithm is a slightly modified version of back substitution in which we are sure to achieve some kind of band structure in the resulting null matrix. The original idea was proposed by A. Topcu in his 1979 doctoral dissertation [43], and has since been improved a bit. The version we present here was introduced by M.W. Berry et al in 1985 as the Refined Turnback Algorithm [6].

If A is an $m \times n$ matrix, then any $m+1$ columns must contain a linear dependency. The turnback algorithm takes advantage of this fact to achieve its banded structure. Intuitively, the algorithm row reduces A, finds its free variables J, and then for each free variable finds a linear dependency from a set of at most $m+1$ previous columns.

In the following pseudocode, b_i is the ith column of matrix B, and the expression $X\langle p\rangle$ denotes the first p vectors from X, whether X is a sequence of vectors or a matrix (in which case take the first p columns).

Turnback Algorithm

Row reduce $m \times n$ matrix $A \to U$
Choose $F := \{f_1, \ldots, f_c\}$ ($c = \mathsf{corank}(A)$)
    so that $f_i \in F$ corresponds to the ith free variable in U
For $i := 0$ to $f_1 - 1$
    Let $\beta_i := f_1 - i$
Next i
For $i := 1$ to c
    Let $E_i := [b_{\beta_1}|b_{\beta_2}|\cdots|b_{\beta_t}]$
        where $t := \min\{|\beta|, m+1\}$
    Row reduce $E_i \to \tilde{E}_i$
    Let $u_i :=$ index of first nonpivot column in $\tilde{E}_i$
    Let $\hat{E}_i := \tilde{E}_i\langle u_i\rangle$
    Find nonzero $\hat{\eta}_i$ so that $\hat{E}_i\hat{\eta}_i = 0$
    Let $\eta_i := 0$; then set $\eta_i(\beta\langle u_i\rangle) := \hat{\eta}_i$
        so that $U\eta_i = 0$
    Remove β_{u_i} from β
    Prepend $(f_{i+1}, f_{i+1} - 1, \ldots, f_i + 1)$ to β
            (so that $\beta_1 = f_{i+1}$)
Next i
Return $N := \left(\begin{array}{c|c|c|c} \eta_1 & \eta_2 & \cdots & \eta_c \end{array}\right)$

See [6] for a proof of the correctness of this algorithm.

Each column n_i of N has its support $\mathrm{supp}(n_i)$ contained in the column indices of $\hat{E}_i$. This means that each column has at most $m+1$ nonzero entries, so that $||N||_{\tilde{0}} \leq (m+1)k$. Solving Sparse Null Space on full rank matrices only makes sense when $m \leq n$, in which case $\mathsf{corank}(A) = k = n - m$. Then $(m+1)k = (n-k+1)k$, so that this analysis of the turnback algorithm reveals no further sparsification when compared to back substitution via property 34.

As stated above, the real advantage comes in the band structure of N.

However, it is difficult to give a general bound on the size of this band — partially because it depends on the unpredictable values of the j_i's and u_i's. The authors in [6] do not even attempt such an inequality, although they do provide experimental evidence supporting the intuition that a banded structure in A assists in maintaining the banded structure in N when using this algorithm.

It is not too difficult to verify that the worst-case example given for back substitution also demonstrates the potentially poor performance of the turnback algorithm. In particular, suppose we let the $m \times n$ matrix $A = (I\ X)$, where I is the $m \times m$ identity matrix and X is any $m \times c$ matrix without zeros, and $c = \mathsf{corank}(A) = n - m$. Then $\hat{\eta}_i$, the possibly nonzero components of column η_i of output N from the turnback algorithm, are in the nullspace of $(I\ x_i)$ up to a column permutation, where x_i is the ith column of X. This clearly means that $||\eta_i||_0 = m + 1$, so that each column of N has $m + 1$ nonzeros, and

$$||N||_{\tilde{0}} = (m + 1)c = (n - c + 1)c,$$

which is the worst case for any $n \times c$ matrix such as N.

Notice that the above worst-case example still holds even when A has a very sparse null space — for example, we could let X have all ones as its entries. In this case, all but one of the vectors in a basis of the null space may be of the form $e_i - e_{i+1}$, so that

$$||N||_{\tilde{0}} = m + 1 + 2(c - 1) = n + c - 1 \prec (n - c + 1)c$$

for $n \succeq kc$ for some $k > 1$ as $n \to \infty$.

The Haar Condition

A matrix A satisfies the **Haar condition** when every square submatrix of A either has no psychological diagonal with all nonzero elements, or is nonsingular. A psychological diagonal of an $n \times n$ matrix B is a subset $\{b_{i,\pi(i)}\}$ of its entries described by a permutation $\pi : [n] \to [n]$. Intuitively, a psychological diagonal is a set of entries which, under the proper row- or column-swaps, form the diagonal of the matrix. We may also give intuition for matrix A satisfying the Haar condition as being exactly when every square submatrix which appears-by-zero-nonzero-structure to be invertible actually is invertible.

Hoffman et al, in [30], have given a polynomial-time algorithm which optimally sparsifies this class of matrices. However, as we have seen in the idea

of candidate submatrices (such as in theorem 28 and corollary 31), a matrix will offer little sparsification unless we can find square submatrices which have many nonzeros yet are singular.

7.2.2 New incremental algorithms

In this section, we will outline two incremental versions of a general algorithm, which we refer to as *null track*, for tracking the exact nullspace of a set of time series considered as a matrix. These methods, originally published by the author in [51], have the advantage of both speed and flexibility. In addition, we will see experimental evidence that they provide levels of sparsity comparable with those of the previous, non-incremental algorithms.

Throughout this section, suppose we are given n time series of length t each; we model this as a sequence of matrices $A_1, A_2, \ldots, A_t$ such that matrix A_t is $t \times n$ in size and each is a "super-matrix" of the previous, in the sense that

$$A_t = \begin{pmatrix} A_{t-1} \\ a^t \end{pmatrix},$$

where a^t is the row vector representing a single datum from each time series at time t.

Tracking full history time series

The full history version of null track operates on a set of time series by maintaining a full null matrix N_t for A_t. Since A_t is constantly growing over time, the sequence of null spaces $\mathsf{col}(N_t)$ can only decrease in dimension over time. Depending on the data, it is very conceivable that $\mathsf{col}(N_t) \to \emptyset$ quickly. However, we will still find it useful to know how to track N_t since it is an essential component of the more stable partial history sliding window version described in the next section.

The algorithm here is very simple to describe, although we will have to explain (in a moment) why it truly maintains a full null matrix, as well as why it encourages sparsity.

Null track: full history

// incremental: assume N_{t-1} is a sparse full null matrix for A_{t-1}

Let $\eta_t :=$ nmat$(a^t N_{t-1})$
Let $N_t := N_{t-1}\eta_t$

Note that, above, η_t is a matrix, whereas in earlier pseudocode it had been simply a column vector. Here we have used the subroutine:

nmat(r): **null space for a row vector**

// exactly solve Sparse Null Space for the nonzero single-row matrix r
// without loss of generality, $r_n \neq 0$

For $i := 1$ to $n-1$
    Let $\eta_i := r_n e_i - r_i e_n$
Next i
Return $N := \left(\eta_1 \,\middle|\, \eta_2 \,\middle|\, \cdots \,\middle|\, \eta_{n-1} \right)$

Although, Sparse Null Space is NP-hard in general, we can in fact solve it exactly in linear time when the input is a row vector, as nmat does. We'll discuss this subroutine in more detail momentarily — first, let's observe the key fact behind this version of null track:

Theorem 36 *Suppose that $A = \binom{A_1}{A_2}$. Also suppose N_1 is a full null matrix for A_1, and N_2 for $A_2 N_1$. Then $N_1 N_2$ is a full null matrix for A.*

We could abbreviate this fact by writing:

$$\mathsf{col}(N_1) = \mathsf{null}(A_1) \;\; \textit{and} \;\; \mathsf{col}(N_2) = \mathsf{null}(A_2 N_1) \implies \mathsf{col}(N_1 N_2) = \mathsf{null}\begin{pmatrix} A_1 \\ A_2 \end{pmatrix}.$$

This fact precisely justifies that the above algorithm for null track maintains exactly a full null matrix N_t for A_t.

Proof. First we'll check that $\eta \in \mathsf{col}(N_1 N_2) \implies A\eta = 0$.

Notice that $A\eta = 0$ iff both $A_1\eta = 0$ and $A_2\eta = 0$. Since $\eta \in \mathsf{col}(N_1) = \mathsf{null}(A_1)$, it immediately follows that $A_1\eta = 0$. At the same time, if ν_i is the ith column of N_2, then $A_2 N_1 \nu_i = 0$ by the definition of N_2. If η_i is the ith column of $N_1 N_2$, then $\eta_i = N_1 \nu_i$ so that $A_2 \eta_i = A_2 N_1 \nu_i = 0$. That is, for any column η of $N_1 N_2$, we have $A_1\eta = A_2\eta = 0 \implies A\eta = 0$.

It's almost as easy to verify the other direction: that $A\eta = 0 \implies \eta \in \mathsf{col}(N_1 N_2)$.

Accordingly, suppose $\eta \in \mathsf{null}(A)$. Then $\eta \in \mathsf{null}(A_1) \implies \eta \in \mathsf{col}(N_1)$, so that there is a vector y with $\eta = N_1 y$. We also have $\eta \in \mathsf{null}(A_2)$ so it must be that $y \in \mathsf{null}(A_2 N_1) = \mathsf{col}(N_2)$. Then there must be a vector z with $y = N_2 z$, and $\eta = N_1 N_2 z$, so that in fact $\eta \in \mathsf{col}(N_1 N_2)$ as claimed. $\square$

Sparsity of full history null track Let's consider the subroutine nmat — we'll confirm that its output N is in fact a full null matrix for input r, and further that N is optimally sparse.

Since the length n row vector r is nonzero, it must have some nonzero entry. If $r_n = 0$, we can permute its coordinates to arrive at $\hat{r}$ with $\hat{r}_n \neq 0$; then the columns of $\hat{N} = \mathsf{nmat}(\hat{r})$ may be unpermuted to arrive at full null matrix N with $rN = 0$, and which is optimally sparse iff $\hat{N}$ is. This is why we can assume $r_n \neq 0$ without loss of generality.

It is easy to see that, if $\eta_i = r_n e_i - r_i e_n$, then $r\eta_i = r_n r_i - r_i r_n = 0$, for any $i \in [n-1]$; that is, N is indeed a null matrix for r. At the same time, if $i < n$, then the ith row of N has exactly one nonzero in the ith column. Thus the columns of N are independent, so the $n \times (n-1)$ matrix N has rank $n - 1 = \mathsf{corank}(r)$; this verifies that N is a full null matrix.

Now for the sparsity: if $||x||_0 \geq 2$, then $||Nx||_0 \geq 2$ since each column η_i of N has the only nonzero in the ith row. Yet we already know that each column of N has at most two nonzeros. So by lemma 29, matrix N must be optimally sparse.

Next we can use theorem 26 to justify that this sparsity is maintained to some degree at each iteration. In particular, the algorithm sets $\eta_t = \mathsf{nmat}(a^t N_{t-1})$ and $N_t = N_{t-1}\eta_t$ so that

$$\begin{aligned} ||N_t||_{\tilde{0}} &\leq c(N_{t-1}) r(\eta_t) \\ &\leq c_1 + c_2 + \ldots + c_{i-1} + ||a^t||_0 c_i + c_{i+1} + \ldots + c_k \\ &\leq ||c(N_{t-1})||_1 + ||a^t||_0 c_i = ||N_{t-1}||_{\tilde{0}} + ||a^t||_0 c_i, \end{aligned}$$

where k is the number of columns in N_{t-1}, c_i denotes the number of nonzeros in the ith column of N_{t-1}, and i is the pivot used by nmat to build η_t. Thus we would be wise to ask nmat to choose its actual pivot $i \in [n]$ as the element of $\text{supp}(r)$ which minimizes c_i.

In our experiments, null track seems to do slightly better than this bound.

Time complexity of full history null track For this analysis, we assume that all matrices are stored in a sparse format, so that operations on zero entries may be skipped.

Subroutine nmat will only require time linear in the size of its input. This size, at time step t, is the number of columns of N_{t-1}; id est, $\mathsf{corank}(A_{t-1})$.

The time to compute the matrix product for N_t can be bounded above by $||N_{t-1}||_{\tilde{0}} + c_i||a^t||_0$ since this is an upper bound for the number of elements in N_t which need to be computed.

Since $||N_t||_{\tilde{0}} \leq n\mathsf{corank}(A_t) \leq n^2$ and $||a^t||_0 \leq n$ where n is the number of time series, we can safely say that the time needed for a single iteration is $O(n^2)$. Of course better speed is achieved when the algorithm manages to maintain some sparsity in N_t.

Before moving on to the partial history version, let us note that this algorithm could be slightly modified to execute a statement of the form Let $N_t := \mathsf{sparsify}(N_t)$ at the end of each iteration, where $\mathsf{sparsify}(A)$ is any function which returns a matrix $B \overset{c}{\sim} A$ with $||B||_{\tilde{0}} \leq ||A||_{\tilde{0}}$. The point is that this version of null track requires nothing special about N_t to work except that N_t is a full null matrix for A_t, so modifying N_t in the meantime will not destroy the incremental action of the algorithm. Thus, if one could derive an $O(n^2)$ time algorithm to significantly increase the sparsity of any matrix, it would be useful to augment null track with this technique.

Tracking partial history sliding windows

The partial history version of null track operates on time series by maintaining a full null matrix N_t for the last (bottom-most) m time units of our time series. This number m is constant over time. If a^t is the row of time series values at time t, then we will now (re-)define the $m \times n$ matrix A_t to consist of rows (from top to bottom) a^t through a^{t+m-1}.

This algorithm actually maintains a decomposition $A_tB_t = C_t$, where B_t is invertible and C_t is in column echelon form. Thus, the last $c = \mathsf{corank}(A_t)$

columns of C_t are exactly the zero columns of C_t; from this it follows that the last c columns of B_t, which we denote as N_t, form a full null matrix for A_t.

The algorithm operates by updating $B_t \to B_{t+1}$ and $C_t \to C_{t+1}$ based on the new incoming row a_{t+m}. We first give pseudocode, and then justify the algorithm in more detail:

Null track: partial history sliding window

// based on (a^{t+s}, B_t, C_t), calculate (B_{t+1}, C_{t+1})

$c^{t+s} := a^{t+s} B_t$
Define $\tilde{C}_{t+1}$ so that $\binom{C_t}{c^{t+s}} = \binom{c^t}{\tilde{C}_{t+1}}$
$X :=$ col_reduce$(\tilde{C}_{t+1})$
$B_{t+1} := B_t \cdot X$
$C_{t+1} := \tilde{C}_{t+1} \cdot X$

The subroutine col_reduce(Y) returns a matrix X so that YX is in column echelon form.

To describe the algorithm in more detail, it will be convenient to define matrix A'_t as the last $m-1$ rows of A_t, so that $A_t = \binom{a^t}{A'_t}$ and $A_{t+1} = \binom{A'_t}{a^{t+s}}$. Notice that $A_{t+1}B_t = \binom{A'_t}{a^{t+s}} B_t = \binom{A'_t B_t}{a^{t+s} B_t} = \binom{C'_t}{c^{t+s}}$, where C'_t are the last $m-1$ rows of C_t and $c^{t+s} = a^{t+s} B_t$ (which is set in the first line of the algorithm).

So far, if we let $\tilde{C}_{t+1} = \binom{C'_t}{c^{t+s}}$ (as is consistent with the algorithm), then we have

$$A_{t+1} B_t = \tilde{C}_{t+1}.$$

All that remains is to column reduce $\tilde{C}_{t+1} \to C_{t+1}$ to maintain that C_{t+1} is in column echelon form. By performing the same column operations on both B_t and $\tilde{C}_{t+1}$ simultaneously, we can maintain the desired matrix decomposition; this is represented by the last three lines of pseudocode. More practically, we can carry out these actions by column reducing

$$\begin{pmatrix} \tilde{C}_{t+1} \\ B_t \end{pmatrix} \quad \to \quad \begin{pmatrix} C_{t+1} \\ B_{t+1} \end{pmatrix}.$$

Figure 8.4 illustrates an example iteration of this algorithm.

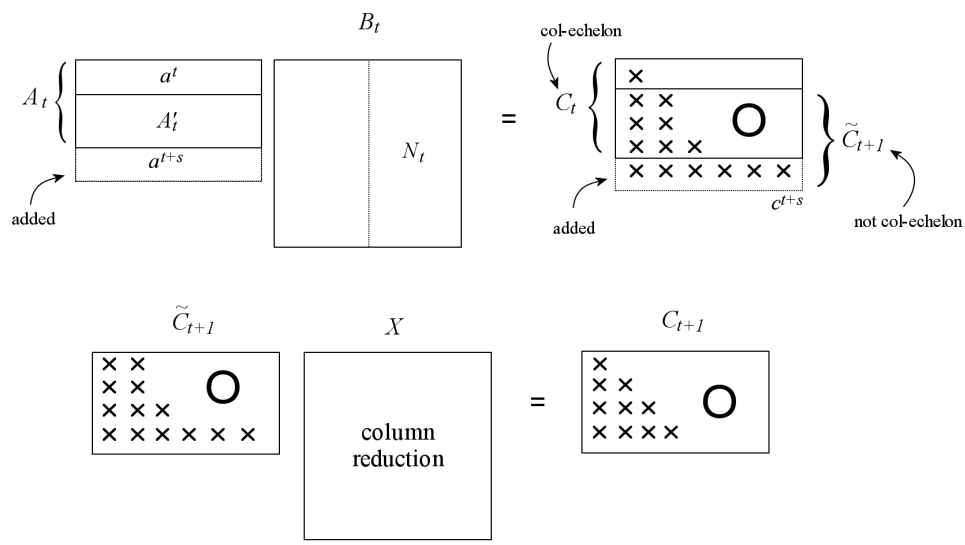

Figure 7.1: One iteration of the partial history null track algorithm

At first sight, this algorithm may appear to be a relatively naïve way to track the null space of a sliding window of time series. To be sure, we have not yet introduced any non-elementary sparsity heuristics; but we will see that this algorithm, when properly implemented, has the advantages of encouraging the conservation of sparsity as well as enabling a faster time complexity when the involved matrices are sparse.

Conservation of sparsity In any single increment $A_t \to A_{t+1}$, the corank may either decrease by one, remain the same, or increase by one. We claim that the sparsity is well maintained when the corank does not increase, and that one relatively dense column may be added to N_t in the remaining case.

The key is to notice that $\tilde{C}_{t+1}$ is *almost* already in column echelon form, since we know that C_t was.

In particular, the bottom row of $\tilde{C}_{t+1}$ may have many nonzeros, but among the rightmost columns corresponding with the null space of A_t, each column will have at most one nonzero (in the bottom row), since in C_t these columns were all zero. Thus we may choose the one of these columns with the most sparse corresponding column in B_t, and use that as a pivot in reducing $\tilde{C}_{t+1}$.

Thus our loss of sparsity (that is, the increase from $||N_{t-1}||_{\tilde{0}}$ to N_t) is bounded above by $n(\ell+1)$, where ℓ is the number of nonzeros in the pivot column — the "+1" is to compensate for the fact that a new (relatively dense) column may be added to N_t if the corank of A_t has increased.

Time Complexity Computing $c^{t+s} = a^{t+s}B_t$ requires time of order $||B_t||_{\tilde{0}} \leq n(n-c) + ||N_t||_{\tilde{0}} \leq n^2$.

We can also give a quadratic bound for the time required to column reduce $\tilde{C}_{t+1}$. In the worst case, $\tilde{C}_{t+1}$ has all nonzeros on the superdiagonal c_{ij}, $j = i+1$. Then we must perform $O(n)$ column operations, one for each consecutive pair of nonzero columns in C'_t, in order to column reduce the leftmost (non-nullspace) rows of $\tilde{C}_{t+1}$. As discussed above, the remaining rightmost columns will all have at most one nonzero in the bottom row; hence these may also be handled using $O(n)$ column operations.

Since each column in $\binom{\tilde{C}_{t+1}}{B_t}$ has at most $m+n$ rows, to perform $O(n)$ column operations will take time at most $O(n(m+n))$; this is an upper bound for the overall time complexity of this algorithm.

Note that, as in the full history version of null track, it is possible to augment this algorithm with a sparsify subroutine at each step. The invariant $A_tB_t = C_t$ would still be maintained since we could rewrite this equation as

$$A_t(\hat{B}_tN_t) = (\hat{C}_t0),$$

where $\hat{C}_t$ is composed of all nonzero columns in C_t, and $\hat{B}_t$ are the corresponding leftmost columns of B_t. Then N_t could be replaced by any matrix $M \overset{c}{\sim} N_t$, and we still would have $AM = 0$, so that

$$A_t(\hat{B}_tM) = (\hat{C}_t0),$$

and our new matrix $B_t = (\hat{B}_tM)$ works as well. Thus, adding a final statement of the form Let $N_t =$ sparsify(N_t) could help increase sparsity while maintaining the integrity of the algorithm.

We also point out that this algorithm really is an extension of our full history version because it operates on the null matrix N_t in essentially the same way whenever corank(A_t) is not increasing; that is, we modify N_t by choosing a pivot column, as nmat did in the first version, and using this pivot column to re-align the old null columns with the updated data. We may even choose the pivot in the same manner: by minimizing the density among all our choices of a potential pivot. Of course, the partial history version can also deal with increasing corank(A_t) at the cost of some time complexity.

Experimental results

We used both real (stock prices) and simulated data to test each of these two algorithms (full or partial history sliding window). Our real data is from the NYSETAQ stock database, and contains trade prices of a randomly selected subset of 500 stocks. The data was normalized so that 100 time ticks correspond to 1 full day (9am–4pm) of trading. The simulated data consists of time series which were mostly random linear combinations of the several previous values (as rows), and occasionally a completely new random set of values (a new random row).

Our algorithm is compared with (nonincremental) back substitution. The back substitution is performed at each time tick on the full sliding window. Recall that the turnback algorithm does not guarantee any increase in sparsity over back substitution (although it does encourage a banded structure); hence these sparsity comparisons should be similar to those from a turnback experiment as well (unless the data in question becomes naturally further sparsified in band-form).

Although time comparisons between these two algorithms are somewhat unfair, it is useful to see how well the sparsity of our much faster incremental algorithms fares against that of the slower nonincremental algorithm. (The author is unaware of any other incremental sparse null space tracking algorithms to compare against.)

Initially, both algorithm versions (full or partial history sliding windows) achieve sparsity nearly as efficiently as back substitution. However, the stability and sparsity of the partial history sliding window algorithm deteriorates slightly over time. To compensate, we periodically restarted the tracked decomposition $AB = C$. Luckily, when the need to periodically recalibrate is anticipated, we can do so in an incremental fashion (using essentially the full history technique in parallel), without any awkward pauses in computation.

- Figure 7.2 shows how the density (percentage of the null matrix which is nonzero) evolves over 60 iterates of our algorithm on the simulated data. Notice the significant improvement we gain by periodically recalibrating our decomposition (which occurs every 20 iterations in the figures).

- Figure 7.3 compares the average number of nonzero entries per column between our algorithm and back substitution run on the stock price data. Although back substitution consistently does at least as well, the factor of improvement is small – in this experiment, the average ratio is 1.11 and

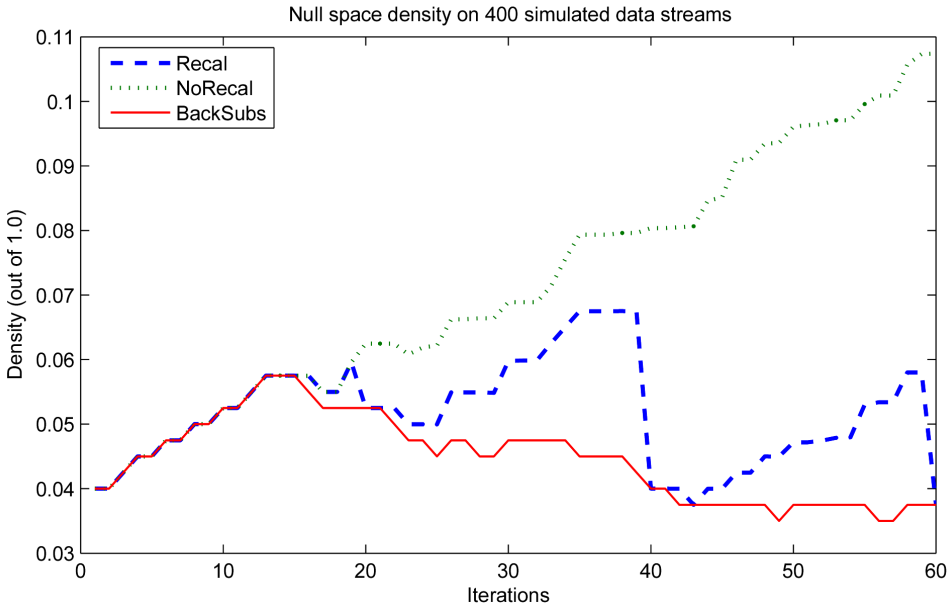

Figure 7.2: Density of null space comparison on simulated data. Back substitution remains between 3–6% while our periodically recalibrated algorithm achieves between 3–7%.

the maximum is 1.25. In practice, a factor of 1.11 would mean that the linear identities we are finding contain 11% more terms than they would with the (much slower) back substitution method. For example, in the stock market application of tracking an index, the larger dependent set would mean that 11% more stocks would be needed to track the index.

- Figure 7.4 illustrates how the time complexity of our algorithm grows with respect to the width of the partial history sliding window. Each data point is the ratio between one iteration of our algorithm in proportion to back substitution on 1000 columns of simulated data. (In this experiment our algorithm always ran at least 49 times faster than the alternative.)

In summary, these incremental algorithms are time efficient and maintain sparsity well in comparison with back substitution. In addition, they are easily

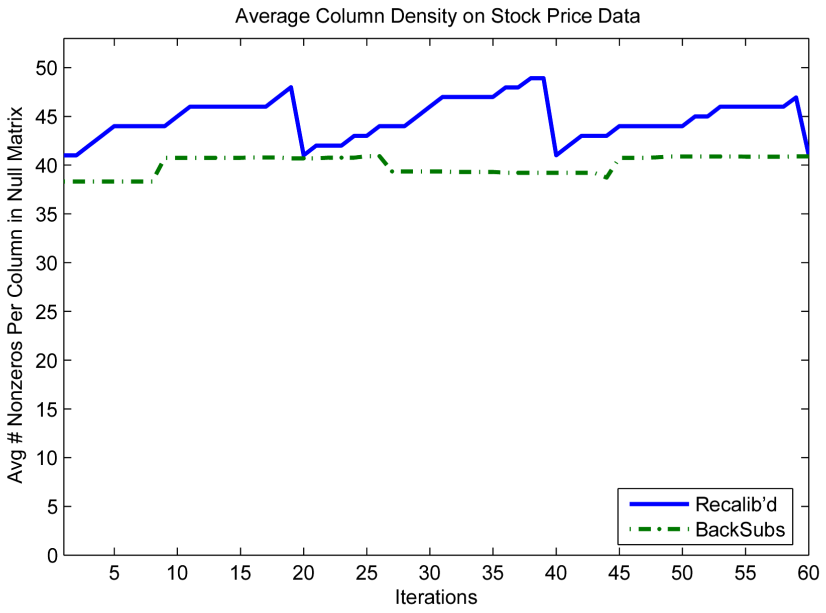

Figure 7.3: Average number of nonzeros per column in stock price data. A full column could have contained up to 500 nonzeros. Back substitution alternated between 38–41 while our algorithm accomplished between 41–49.

extensible with augmented null matrix manipulation via an optional sparsify function.

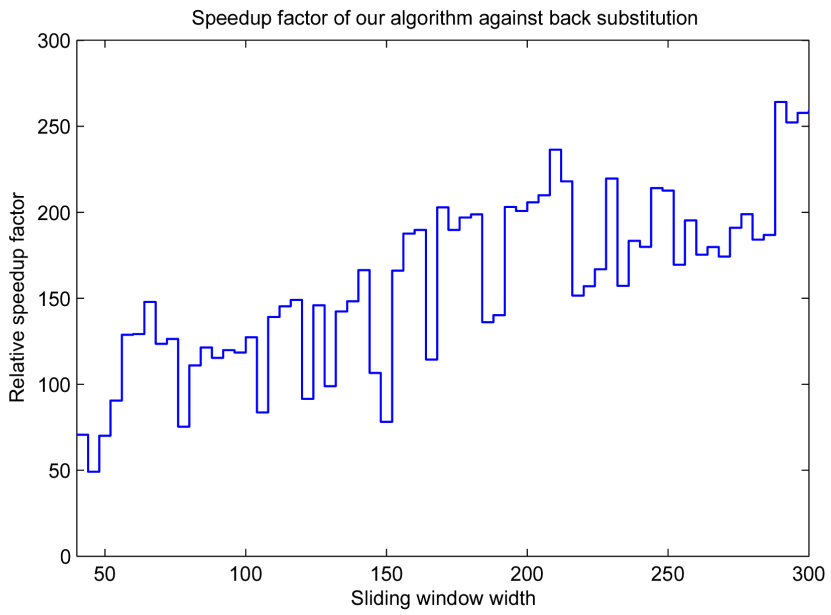

Figure 7.4: Relative speedup factor between our algorithm and back substitution over partial history sliding windows of increasing width. Tests were run on 1000 streams of simulated data. The factor represents the time efficiency gained over one iteration.

Chapter 8

Probabilistic and approximate sparsification

8.1 Hardness of approximation

In this section, we'll see that the problems we care about — Matrix Sparsification, as well as P_0, an affine version of Sparsest Vector — are both NP-hard to approximate within a certain factor of the optimal answer. We'll also examine two techniques for probabilistically approximating these problems. The first, originally published by Piotr Berman and Marek Karpinski [5], justifies that a certain method of "random sparsification" is likely to achieve an answer within a factor of n/c of optimal, for some fixed c. The second technique is to use L^1 minimization as a convex optimization alternative to L^0 minimization. David Donoho et al. [16], [17], [11], [9] have provided some theoretical justification for this idea. The L^1 minimization has the disadvantage of failing completely on a certain class of inputs; but it has the advantage of often achieving the *exact* optimal answer when it does work, and it does work "often" in a certain sense. Because of these advantages, our incremental algorithm presented in §8.2 will use L^1 minimization rather than the "random sparsification" technique.

8.1.1 Equivalence and APX-hardness of Min-Unsatisfy

We'll begin by formally stating problems P_0 and Min-Unsatisfy, and showing that they are equivalent. Previous work has shown that Min-Unsatisfy is hard to approximate, so this equivalence reveals that P_0 is hard as well.

Problem P_0 is a slight variant to Sparsest Vector:

P_0	Given matrix A and column vector $b \neq 0$, find nonzero x which solves $$\mathsf{P}_0 \quad \begin{cases} \min & ||x||_0 \\ \text{s.t.} & Ax = b, \end{cases}$$ or indicate that no such x exists.

If we required that $b = 0$, then this problem would be equivalent with Sparsest Vector.

Our other problem of interest is:

Min_Unsatisfy	Given matrix A and vector $b \neq 0$, find y which solves $$\min ||Ay - b||_0.$$

Suppose x^* are y^* are optimal answers for $\mathsf{P}_0$and Min_Unsatisfy, respectively. In the approximation version of these problems, we are then attempting to minimize the approximation factors

$$\frac{||x||_0}{||x^*||_0} \quad \text{and} \quad \frac{||Ay - b||_0}{||Ay^* - b||_0}.$$

In the following, we write $\mathsf{P}_0(A, b)$ and Min_Unsatisfy(C, d) to denote the solutions x and y to these problems on inputs (A, b) and (C, d).

Property 37 *Problems* P_0 *and* Min_Unsatisfy *are equivalent. That is, for every instance* (A, b) *of input to* P_0*, there is a polynomial-time computable instance* (C, d) *of input for* Min_Unsatisfy *such that*

$$P_0(A, b) = \textit{Min_Unsatisfy}(C, d).$$

Symmetrically, there is also a polynomial-time reduction in the other direction.

We'll give the proof in a moment.

In their 1998 paper [14], Irit Dinur, Guy Kindler, and Shmuel Safra showed that Min-Unsatisfy is NP-hard to approximate within a factor of $n^{\Omega(1)/\log\log n}$ over any finite field. For arbitrary fields, Sanjeev Arora et al [3] have shown it to be quasi-NP-hard to approximate Min-Unsatisfy within a factor of $\rho = 2^{\log^{1-\gamma} n}$ for any $\gamma \in (0, 1)$; a problem prob is **quasi-NP-hard** when prob $\in$

$P \implies NP \subset DTIME(n^{\text{poly}(\log(n))})$. To get a better feel for this factor ρ, notice that $e^{\log^{1-\gamma} n} = n^{1/\log^{\gamma} n}$. Asymptotically, writing $f(n) \prec g(n)$ to indicate that $f(n)/g(n) \to 0$ as $n \to \infty$, we can see that

$$\log^{1-\gamma}(n) \prec \rho = n^{1/\log^{\gamma} n} \prec n^{1/\log\log n} \prec n.$$

Since these problems (P_0 and Min-Unsatisfy) are equivalent, any inapproximability result for one also holds for the other. This is interesting to us because, in §8.2, our main algorithm relies on solving P_0; since we know it is a hard problem, we are justified in using a probabilistic-approximate approach. The hardness of Min_Unsatisfy is also interesting to us because we may reduce it to Matrix Sparsification in a gap-preserving way (as we will see in a moment); thus knowing that Min_Unsatisfy is quasi-NP-hard to approximate suffices to inform us that Matrix Sparsification, and thus the equivalent problem Sparse Null Space, are both also hard to approximate.

Proof of property 37.

The main idea behind the equivalence is essentially the same as that behind the reduction between Matrix Sparsification and Sparse Null Space; indeed, in both cases we are trying to find a vector in an affine space with maximal sparsity.

Given input (A, b) to P_0, we would like to find an input (C, d) to Min_Unsatisfy so that

$$\{x | x = Cy - d \text{ for some } y\} = \{x | Ax = b\}. \tag{8.1}$$

If $Ax = b$ has no solutions, then no reduction is necessary. Otherwise, choose d so that $Ad = -b$ and matrix C so that $\mathsf{col}(C) = \mathsf{null}(A)$. (To confirm that we can find d and C in polynomially time, please refer to the Linear Computations appendix at the end of this thesis.) Then

$$x = Cy - d \implies Ax = ACy + b = b$$

and

$$\begin{aligned} Ax = b \quad &\implies \quad A(x + d) = 0 \quad \implies \quad x + d \in \mathsf{col}(C) \\ &\implies \quad x = Cy - d \text{ for some } y, \end{aligned}$$

confirmed that equation 8.1 holds, as we wanted. This shows that the solution y to Min_Unsatisfy(C, d) immediately gives us the solution $x = Cy - d$ to $\mathsf{P}_0(A, b)$.

Now suppose we are given input (C, d) to Min_Unsatisfy; our reduction is essentially unchanged. We choose A so that $\mathsf{col}(C) = \mathsf{null}(A)$, and let $b = -Ad$ (again, the Linear Computations appendix explains how this may be found in polynomial time). As above, equation 8.1 still holds, and the solution x to $\mathsf{P}_0(A, b)$ allows us to solve $Cy = x + d$ for y, the solution to Min_Unsatisfy(C, d) — again, we know that $x + d \in \mathsf{col}(C) = \mathsf{null}(A)$ since $A(x + d) = b - b = 0$. This completes the equivalence. □

8.1.2 APX-hardness of Matrix Sparsification

We will see that Matrix Sparsification is at least as hard to approximate (up to an additive ϵ term) as Min_Unsatisfy by reducing the latter to the former in a gap-preserving manner. This theorem is a result of joint work between the author and Lee-Ad Gottlieb.

First we'll define precisely our approximation factors, and then we'll proceed with the statement and proof of the reduction. In the following, we use the phrase "minimization version" to emphasize that we are focused on minimizing the number of nonzero elements, as opposed to maximizing the number of zeros. Although the exact solutions to these optimization perspectives are identical, their approximation factors are significantly different. In our case, the minimization (of nonzeros) perspective is more interesting than maximization (of zeros) since we care about those solutions which are very sparse — indeed, when a matrix may be sparsified so that it contains 98% zeros, an approximating matrix with twice as many nonzeros (and thus 96% sparsity) is much more interesting than an approximating matrix with half as many zeros (and thus 49% sparsity).

We will say that an algorithm f is a **ρ-approximation for the minimization version of** Matrix Sparsification when, for any matrix C, algorithm f computes matrix $E = f(C)$ in polynomial time with $E \overset{c}{\sim} C$ and $||E||_{\tilde{0}} \leq \rho||D||_{\tilde{0}}$, where D is an optimal answer to Matrix Sparsification(C). Thus $\rho \geq 1$ and a smaller ρ indicates a better approximation algorithm. We will allow ρ to depend on the size of the input.

Similarly, we will say that an algorithm g is a **κ-approximation for the minimization version of** Min_Unsatisfy when, for any matrix-vector instance (A, b) with an optimal answer x (so that $||Ax - b||_0$ is minimized), algorithm g computes $y = g(A, b)$ in polynomial time with $||Ay - b||_0 \leq \kappa||Ax - b||_0$.

We are now ready to state our result.

Theorem 38 *Suppose there exists a ρ-approximation algorithm for the minimization version of* Matrix Sparsification*, for some $\rho = \rho(m, n)$ where the instance matrix A is of size $m \times n$. Then, for any fixed $\epsilon > 0$, there also exists a $(\rho+\epsilon)$-approximation algorithm for the minimization version of* Min_Unsatisfy*.*

Proof. Suppose Min_Unsatisfy is given instance (A, b), where A is $m \times n$ in size. We assume without loss of generality that the columns of A are linearly independent (otherwise we may throw out dependent columns) and that there is no x for which $Ax = b$ exactly (otherwise this problem instance is trivial); these conditions may easily be handled in polynomial time.

We will create an instance C of Matrix Sparsification, such that any $\rho-$approximation to its answer will give us a $(\rho + \epsilon)$-approximation to the optimal answer $||Ax - b||_0$ to Min_Unsatisfy(A, b).

We'll begin by creating the large matrix

$$C = \left(\begin{array}{cccc|ccc} s & & & 0 & & & \\ & s & & & & \mathbf{0} & \\ & & \ddots & & & & \\ 0 & & & s & & & \\ \hline x_{11}b & x_{12}b & \cdots & x_{1q}b & A & & 0 \\ & \ddots & & & & A & \\ & & \ddots & & & & \ddots \\ x_{q1}b & & & x_{qq}b & 0 & & A \end{array}\right).$$

Here, s is the $m \times 1$ column vector of all one$s$ and X denotes some $q \times q$ completely unsparsifiable matrix. We remind the reader that, as we saw in theorem 28, X being completely unsparsifiable implies that

$$\left|\left|\begin{pmatrix} I \\ X \end{pmatrix}\right|\right|_{\tilde{0}} \leq \left|\left|\begin{pmatrix} I \\ X \end{pmatrix} Y\right|\right|_{\tilde{0}}$$

for any invertible matrix Y. We leave q as an unspecified quantity until it becomes clear which value we should choose.

Let us introduce the notation '$A * B$,' indicating the matrix obtained by replacing each entry a of A by matrix aB. Then we could succinctly write our large matrix as

$$C = \begin{pmatrix} I * s & 0 \\ X * b & I * A \end{pmatrix}. \tag{8.2}$$

Here, I denotes the $q \times q$ identity matrix. We will analyze this matrix in terms of these four blocks. We will say "top part," "left part," and so on to indicate the top two blocks, $(I * s \;\; 0)$, or the left two blocks, $\binom{I*s}{X*b}$, of C respectively (and likewise for "right" and "bottom"). We will use the word "subcolumn" to indicate one of the $m \times 1$ submatrices in the lower-left block of the form $x_{ij}b$.

Let $D \overset{c}{\sim} C$ denote an optimal answer to Matrix Sparsification, and let $u = ||Ax - b||_0$ be the size of an optimal answer x to Min_Unsatisfy(A, b). Also let $E \overset{c}{\sim} C$ denote our ρ-approximation to D. We now give an outline for the remainder of our argument:

1. No sparsification can take place within the left part of C.

2. The top part of C remains essentially unchanged, and most of the sparsification in D can occur only in the lower-left block by replacing each $x_{ij}b$ subcolumn with another of the form $Aw + \lambda b$; thus each subcolumn will optimally have exactly u nonzeros.

3. So we can determine bounds on $||E||_{\tilde{0}}$ in terms of u (and ρ).

4. These bounds allow us to build $g(A, b)$ from E, so that it is a $(\rho + \epsilon)$-approximation for u.

We may write the left part of C as $\binom{I*s}{X*b}$. Any column sparsification of this matrix may also be performed on $\binom{I}{X}$ to reach the same fraction of sparsification. By our choice of X as completely unsparsifiable, this means that no linear combination of columns just within the left part can help sparsify any column within this part of C; this is part 1 of our proof outline.

We will now argue that the upper-left part of C remains unchanged in D. Using theorem 28, we can see that, for every all-zero subcolumn in the lower-left part of D, there must be an added nonzero subcolumn in the upper-left part. Yet by elementary linear algebra the optimal answer $y = Ax - b$ to Min_Unsatisfy(A, b) must have $u = ||y||_0 < m$. Thus, for each subcolumn of the lower-left part of D, we may either essentially move these nonzeros to the upper-left part, or replace them by some vector (such as y) which has strictly less than m nonzeros. Clearly, by the optimal sparsity of D, it must be the case that no new nonzero subcolumns have been added in the upper-left, and each subcolumn in the lower-left has been replaced by some vector of the form $Av + \lambda b$ with u nonzeros. This justifies step 2 of our outline.

So the lower-left part of D will have exactly q^2u nonzeros. The upper-left part will have exactly qm nonzeros, and the lower-right part will have between 0 and qmn nonzeros. That is,

$$q^2u + qm \leq ||D||_{\tilde{0}} \leq q^2u + qm(n+1).$$

If matrix $E \overset{c}{\sim} C$ is an approximate answer to Matrix Sparsification with $||E||_{\tilde{0}} \leq \rho||D||_{\tilde{0}}$, this means that

$$q^2u + qm \leq ||E||_{\tilde{0}} \leq \rho(q^2u + qm(n+1)).$$

These are the bounds mentioned in step 3 of our outline.

Notice that each of the subcolumns in the lower-left block of E are still of the form $Av+\lambda b$. Then there must exist some nonzero subcolumn $z = Av+\lambda b$ which has

$$||z||_0 \leq \rho\left(u + \frac{m(n+1)}{q}\right),$$

otherwise $||E||_{\tilde{0}} > q^2\rho\left(u + \frac{m(n+1)}{q}\right) = \rho(q^2u + qm(n+1))$, which contradicts the above inequality. Using this z, we now define $g(A, b) = w$ where $Aw-b = z$.

From the above bounds it follows that

$$u \leq ||Aw - b||_0 = ||z||_0 \leq \rho u + \rho\frac{m(n+1)}{q}.$$

Now we know to choose $q = \rho m(n+1)/\epsilon$, so that

$$u \leq ||Aw - b||_0 \leq \rho u + \epsilon \leq (\rho + \epsilon)u,$$

since $1 \leq u$, which is our final step (4) in the proof. □

8.1.3 An approximation algorithm in RP

In 2001, Piotr Berman and Marek Karpinski published a proof [5] that a type of random guessing procedure which estimates Min_Unsatisfy(A, b) is likely to approximate the optimal answer within a factor of $(n/c) + 2$ for any fixed constant c, where n is the number of columns in input A. In particular, for any fixed probability p and constant c, we can run enough iterations of the algorithm (explained below) so that with probability at least p, the result returned is within a factor of $(n/c) + 2$ of the optimal.

Because we will extend this approximability result to Matrix Sparsification, we'll present here a slightly more rigorous version of their proof in a style consistent with this thesis.

First we present the algorithm itself:

Random Guess toward Min_Unsatisfy(A, b)

```
// Grow a set B_i of independent row indices of A,
// with |B_i| = i at the end of the i^th iteration.
// Matrix A has size m × n.

Let B_0 = ∅
Let r = rank(A)
For i = 1 to r
    Let I_{i-1} :=
        {t ∈ [m] − B_{i-1} | row t is independent of rows B_{i-1}}
    Randomly choose one t ∈ I_{i-1}
    Let B_i := B_{i-1} ∪ {t}
Next i
Solve for x in A(B_r, :)x = b
Return x
```

It is important to notice that the reduced system $A(B_r, :)x = b$ is guaranteed to have a solution x since the rows of $A(B_r, :)$ are linearly independent.

We now claim that

Theorem 39 *With probability at least e^{-d}, Random Guess returns an answer within a factor of $(n/d) + 2$ of the optimal (measured in terms of number of nonzeros), where n is the number of columns in input (A, b).*

Proof. We will argue in terms of the (shrinking) size of the set $\mathcal{I}_{i-1}$. Intuitively, if $\mathcal{I}$ is large and the optimal answer is very sparse, then most of the guesses in $\mathcal{I}$ are "good;" if $\mathcal{I}$ is very small at any point, then we can give an upper bound on the ratio between the size of this answer x and the optimal x^*.

In more detail: let x^* denote a solution which minimizes $||Ax^* - b||_0$, and U^* the support of $Ax^* - b$ (so that $|U^*| = ||Ax^* - b||_0$). Analogously, x will be

the vector returned by Random Guess, and we'll let $u = \text{supp}(Ax - b)$. Our goal is then to show that

$$P\left(\frac{|U|}{|U^*|} < \frac{n}{d} + 2\right) > e^{-d}.$$

Our first claim is that, for any $\alpha > 1$,

$$\left(B_{i-1} \subset [m] - U^* \text{ and } |\mathcal{I}_{i-1}| < \alpha|U^*|\right) \implies |U| < (1+\alpha)|U^*|. \tag{8.3}$$

The first part, $B_{i-1} \subset [m] - U^*$, means that the rows selected thus far by Random Guess represent satisfied equations in the optimal answer x^*; that is, if $t \in B_{i-1} \subset [m] - U^*$, then row a^t satisfies equation $a^t x^* = b_t$.

So we now suppose that $B_{i-1} \subset [m] - U^*$. Recall that our eventual output x will solve $A(B_{i-1}, :)x = b$ and u will be the set of row indices t for which $a^t x \neq b_t$ (this is the support of vector $Ax - b$). Although at step i of the algorithm we can't know x, we can still know certain of its properties.

If row a^t is dependent on the rows B_{i-1}, then equation $a^t x = b_t$ is either consistent or inconsistent with system $A(B_{i-1}, :)x = b$. If they are consistent, then any solution x to the system will also solve $a^t x = b_t$. If they are inconsistent, then no solution to the system will solve $a^t x = b_t$. Thus if $t \in u$ (that is, if $a^t x \neq b_t$), it must be the case that they are inconsistent, and it follows that t must also be in u^* since we have supposed that x^* also solves $A(B_{i-1}, :)x^* = b$. In other words,

$$\left(t \in U \text{ and } t \notin \mathcal{I}_{i-1}\right) \implies t \in U^*.$$

From this it is easy to see that

$$U - \mathcal{I}_{i-1} \subset U^*$$

$$\implies \quad |U| \leq |\mathcal{I}_{i-1}| + |U^*|.$$

This verifies equation (8.3).

We now note that

$$|\mathcal{I}_{i-1}| \geq \alpha|U^*| \implies \frac{|U^*|}{|\mathcal{I}_{i-1}|} \leq \frac{1}{\alpha},$$

which is trivially true.

Now suppose that $B_{i-1} \subset [m] - U^*$ and we are about to add a new row to create B_i. If $|\mathcal{I}_{i-1}| \geq \alpha|U^*|$, then, by the above equation, we will choose another row in $[m]-U^*$ with probability at least $1-1/\alpha$. On the other hand, if $|\mathcal{I}_{i-1}| < \alpha|U^*|$, then equation (8.3) shows that, no matter how Random Guess finishes, we have already achieved an approximation factor at least as good as $\alpha+1$. Thus, with probability at least $(1-1/\alpha)^r$, Random Guess will achieve an approximation factor at least as good as $\alpha+1$. Since $r = \mathsf{rank}(A) \leq n$, this probability is at least $(1-1/\alpha)^n$.

A little extra analysis allows us to turn this inequality into the form stated in the theorem.

We claim that, for any positive n and any $\alpha > 1$,

$$\left(1 - \frac{1}{\alpha}\right)^n \geq e^{-n/(\alpha-1)}.$$

We'll prove this below as a lemma.

Using this inequality, we can set $\alpha = n/d + 1$ to see that

$$P\left(\frac{|U|}{|U^*|} < 1 + \alpha = \frac{n}{d} + 2\right) > e^{-n/(\alpha-1)} = e^{-d},$$

as claimed. This completes the proof. □

Lemma 40 *For $x > 0$,*

$$\log(x) \geq 1 - \frac{1}{x}.$$

From this we can see that

$$\left(1 - \frac{1}{\alpha}\right)^n \geq e^{-n/(\alpha-1)}$$

for any $n > 0$ and $\alpha > 1$.

Proof. The first part is easy. We know that

$$\forall y, \quad 1 + y \leq e^y,$$

so that

$$\forall y > -1, \quad \log(1+y) \leq y.$$

From here, set $x = 1/(1+y)$ to see that

$$\forall x > 0, \quad \log(x) \geq 1 - 1/x.$$

Then, setting $x = 1 - 1/\alpha$,

$$\begin{aligned} x \geq e^{1-1/x} &\implies (1 - 1/\alpha) \geq e^{-1/(\alpha-1)} \\ &\implies (1 - 1/\alpha)^n \geq e^{-n/(\alpha-1)}, \end{aligned}$$

for $\alpha > 1$ and $n > 0$ as claimed. □

We will see below (§8.2.1) that this technique can also give an equally good approximation for Matrix Sparsification.

8.1.4 L^1 minimization approximates L^0 minimization

In this section, we state one of the main results supporting the idea that L^1 minimization gives a useful approximation for L^0 minimization. First, we define problem P_1:

P_1 Given matrix A and column vector $b \neq 0$, find $x = \mathsf{P}_1(A, b)$ which solves

$$\mathsf{P}_1 \quad \begin{cases} \min & ||x||_1 \\ \text{s.t.} & Ax = b. \end{cases}$$

This is very similar to P_0 — the only difference being the norm we minimize over. However, P_1 is a convex optimization problem, so it can be solved in polynomial time. In fact, by setting $x = u - v$ where $u, v \geq 0$, we can restate P_1 as the linear programming problem

$$\begin{cases} \min & \mathbf{1}^T u + \mathbf{1}^T v \\ \text{s.t.} & (A \quad -A)\binom{u}{v} = b \\ & \text{and } \binom{u}{v} \geq 0. \end{cases}$$

The surprising outcome is that, in many cases, the solution to P_1 is often *exactly the same* as the solution to P_0 for the same (A, b) pair.

This following result is proven by Donoho as Theorem 1 in [16]. Here, $\mathbb{S}^{m-1} = \{x \in \mathbb{R}^m : ||x|| = 1\}$ denotes the set of unit vectors in $\mathbb{R}^m$.

Theorem 41 *Suppose we have a sequence of random matrices $A_1, A_2, \ldots$, such that A_m is of size $m \times \lfloor \alpha m \rfloor$ for some fixed aspect ratio $\alpha > 1$, and each column of A_m is drawn uniformly from $\mathbb{S}^{m-1}$. Then there exists a fraction $\rho = \rho(\alpha) \in (0,1)$ so that, for any sequence $b_1, b_2, \ldots$ with $b_m \in \mathbb{R}^m$,*

$$P\big(||x_m^*||_0 < \rho m \implies \mathsf{P}_1(A_m, b_m) = x_m^*\big) \to 1 \quad as \quad m \to \infty,$$

where x^* denotes an optimal answer to $x^* = \mathsf{P}_0(A_m, b_m)$.

Intuitively, this gives strong evidence that for input (A, b), when an optimal answer of size $||x^*||_0 = o(m)$ exists, it is likely that

$$\mathsf{P}_1(A, b) = \mathsf{P}_0(A, b),$$

so that we may discover x^* by solving the tractable problem P_1. The precondition that $||x^*||_0$ be small is not a significant drawback (to some extent), since we are not usually interested in non-sparse answers. The primary drawback of this result is that the probability distribution is over the *input* (A, b) to the algorithm, rather than over different runs of the algorithm (P_1). Thus, P_1 will fail altogether on some inputs, and give us no immediate hints as to whether or not it has even failed. However, considering the NP-hardness of even approximating this problem, it seems necessary (modulo the $P \neq NP$ conjecture) that any polynomial-time heuristic for P_0will require some sacrifice in whatever guarantees it can provide on how often it finds the exact answer.

8.2 Algorithms

In this section, we will first describe (in §8.2.1) a way in which any heuristic for P_0, such as P_1, may be used to also approximate Matrix Sparsification. We then continue to describe (in §8.2.2) a relatively fast incremental technique which tracks a sparse linear estimator of a particular time series in terms of n others over a sliding window of size m. This new algorithm utilizes a natural extension of the null space, which we call an $\epsilon-$*space* (in §8.2.3), to allow a degree of error tolerance in finding a sparse linear approximator. It also relies on the *block power iteration* technique of computing the singular value decomposition, which we briefly review for the reader (in §8.2.4). Finally (in §8.2.5), we show that this new algorithm runs in time $O(k^2 n + \ell)$ per step, where k is an *a priori* reduced dimension to use (so $k \ll n$), n is the number of time series, and ℓ

is the time complexity needed for a linear programming solver to complete a warm-started version of P_1. We leave the variable ℓ indirectly defined because in practice it is much better than the best polynomial time bounds known. This time complexity is particularly nice because it is *completely independent* of m, the size of the sliding window.

8.2.1 Heuristics for solving Matrix Sparsification

As we saw in property 37, P_0 and Min_Unsatisfy are dual versions of essentially the same problem; analogous to the matrix versions Sparse Null Space and Matrix Sparsification (whose equivalence was shown in §7.1). We will see here how any approximator to the vector versions (Min_Unsatisfy or P_0) can be extended to its respective matrix version (Matrix Sparsification or Sparse Null Space).

Approximating Matrix Sparsification Berman and Karpinski, in [5], conclude by giving a derandomized version of their Random Guess approximator for Min_Unsatisfy. They show that for any fixed c, there is a deterministic polynomial-time algorithm $x =$Approx_Min_Unsatisfy(A, b) so that $||Ax - b||_0 < (n/c + 2)||Ax^* - b||_0$, where x^* is an optimal solution. It is not too difficult to extend this approximation to Matrix Sparsification:

$(\frac{n}{c} + 2)$ **Approximation for** Matrix Sparsification

```
// Matrix A has size m × n.
// A_i denotes the m × (n − 1) matrix we achieve by
// removing a_i, the i^th column of A

For i := 1 to n
    Let x :=Approx_Min_Unsatisfy(A_i, a_i)
    Replace a_i by a_i − A_i x
Next i
Return A
```

We will argue that this algorithm does indeed achieve the claimed approximation factor: suppose $D = AY$ is an optimally sparse matrix with $D \overset{c}{\sim} A$.

Since Y is invertible, there must be some column permutation P so that every element on the diagonal of YP is nonzero. From here it is possible to replace D with DP and Y with YP; so we will assume, without loss of generality, that every element on the diagonal of Y is nonzero. Intuitively, this means that every column d_i of D is a linear combination of columns, including a_i, from A.

Thus by solving $x :=$Approx_Min_Unsatisfy(A_i, a_i), we will find a column $\tilde{a}_i = a_i - A_i x$ with $||\tilde{a}_i||_0 \leq (\frac{n}{c} + 2)||d_i||_0$ and thus, for any output E from this approximation algorithm,

$$||E||_{\tilde{0}} \leq \left(\frac{n}{c} + 2\right) \sum_i ||d_i||_0 = \left(\frac{n}{c} + 2\right) ||D||_{\tilde{0}},$$

as desired.

While this approximation scheme is the most "theoretically satisfying" presented in this thesis, it is not as pragmatically useful as L^1 minimization since it tends to achieve less sparse results in practice.

We may easily extend this approximation result to our equivalent problem Sparse Null Space. Given an instance A of Sparse Null Space, simply find any full null matrix N for A, and then proceed by solving Matrix Sparsification on N itself.

8.2.2 Fast incremental algorithms with high sparsity

Here we present two related algorithms which track over time a sparse vector x so that $\tilde{A}x = b$, where $\tilde{A}$ is our sliding window data, and b is the target time series. For example, we may set b as a time-delayed stock price for a particular symbol, and choose $\tilde{A}$ as a set of other stock prices. Then any solution to $\tilde{A}x = b$ in effect is predicting the value of b using a small number of symbols in $\tilde{A}$.

It will be easier to replace the matrix/column pair $(\tilde{A}, b)$ by a single matrix A, in which our target vector (formerly b) is simply the ith column a_i, for a particular i. If we write A_i for matrix A without column a_i, then we could state the objective of this algorithm as approximating, at each step, the solution to $\mathsf{P}_0(A_i, a_i)$.

The algorithms assume that there is an $m \times n$ sliding window A, which is being updated to the new window $\hat{A}$ (in general, we will use the hat accent $\hat{X}$ to denote the incrementally-updated version of variable X). We assume that α is the new incoming data (as a row), and β is the old outgoing row. We

summarize this by writing

$$\begin{pmatrix} A \\ \alpha \end{pmatrix} = \begin{pmatrix} \beta \\ \hat{A} \end{pmatrix}.$$

Our first version incrementally maintains matrices B and Q so that $B = A^T A$ is $n \times n$ and the rows of $k \times n$ matrix Q are the k eigenvectors of B with the largest eigenvalues. In the following pseudocode, the subroutine most_sig_eig$(\hat{B}, Q^T, k)$ returns the k most significant eigenvectors of $\hat{B}$ — since a converging iterative method could be used to implement this subroutine, most_sig_eig also accepts the input Q^T as a starting point for this convergence. We will see in the next section why tracking this matrix Q allows us to essentially apply a constraint of the form $A_i x \approx a_i$.

We refer to this first version as the *stable* one in contrast with our second, which we will see is less stable although faster.

Incrementally Approximate $\mathsf{P}_0(A_i, a_i)$: Stable version

// $m \times n$ matrix A is the old sliding window
// We have the input:
// α is the new row of data, β is the old row
// τ is the taper factor, $\tau \in (0, 1]$
// $n \times n$ matrix $B = A^T A$
// rows of $k \times n$ matrix Q are the k most significant eigenvectors of B
// Also, $\hat{Q}_i$ denotes matrix $\hat{Q}$ without its ith column $\hat{q}_i$

Let $\hat{B} = \tau B + \alpha^T \alpha - \tau^m \beta^T \beta$
Compute $\hat{Q}^T =$ most_sig_eig$(\hat{B}, Q^T, k)$
Solve $\begin{cases} \min & ||x||_1 \\ \text{s.t.} & \hat{Q}_i x = \hat{q}_i \end{cases}$
Keep $(\hat{B}, \hat{Q}, x)$ for the next time step
Return x

We will see that the above algorithm guarantees to provide an upper bound on both the sparsity of x as well as the accuracy of the approximation $Ax \approx 0$ (although this latter bound is data-dependent). If we are willing to sacrifice some certainty in the accuracy bound (concerning the quality of $Ax \approx 0$), then

we may obtain a significant increase in speed which can be achieved by our *very fast* version below (it would be misleading to call it simply the "fast version" since in fact both algorithms are fast relative to any naive implementation).

In the fast version, we no longer track B, although we still maintain a $k \times n$ matrix Q whose rows should *approximate* the most significant eigenvectors of $A^T A$. We also keep track of the $k \times n$ matrix S, which approximates $S \approx QA^T A$. The motivation for maintaining the pair (S, Q) is to perform, at each time step, a single iteration of the block power iteration method toward computing the singular value decomposition of A; we will explain this idea in more detail in §8.2.4 below.

Incrementally Approximate $\mathsf{P}_0(A_i, a_i)$: Very fast version

// $m \times n$ matrix A is the old sliding window
// We have the input:
// α is the new data, β is the old
// $k \times n$ matrix Q are the first k right singular vectors for A
// $S = QA^T A$
// Also, $\hat{Q}_i$ denotes matrix $\hat{Q}$ without its ith column $\hat{q}_i$

Let $\hat{S} := S + Q(\alpha^T \alpha - \beta^T \beta)$
Let $\hat{Q} :=$ orthonormalize_rows$(\hat{S})$
Solve $\begin{cases} \min & ||x||_1 \\ \text{s.t.} & \hat{Q}_i x = \hat{q}_i \end{cases}$
Keep $(\hat{S}, \hat{Q}, x)$ for the next time step
Return x

In the following sections, we'll see why these algorithms work and examine how quickly and accurately they each operate.

8.2.3 Idea of the $\epsilon-$space

We are all familiar with the idea of the *null space* of a matrix (or linear operator in general). The idea of this section is to extend this idea to a natural set of

vectors v for which

$$\frac{||Av||}{||v||} < \epsilon \tag{8.4}$$

for some $\epsilon > 0$ which we may specify.

One might be tempted to consider the set

$$S = \{v : ||Av|| \leq \epsilon ||v||\}.$$

However, it is often the case that S is *not* a vector space. For example, if we let

$$A = \begin{pmatrix} -1 & 3 \\ -2 & 3 \end{pmatrix} \quad x = \begin{pmatrix} -1 \\ 0 \end{pmatrix} \quad y = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

then

$$\frac{||Ax||}{||x||} = \sqrt{5} < 3, \quad \frac{||Ay||}{||y||} = \sqrt{5/2} < 3, \quad \text{but} \quad \frac{||A(x+y)||}{||x+y||} = 3\sqrt{2} > 3.$$

In other words, when $\epsilon = 3$ we have $x, y \in S$ but $x + y \notin S$.

In order to arrive at a vector space (which is of course closed under addition) that satisfies equation (8.4) for nonzero v, we will need to select a particular subset of S. The following choice seems natural:

Definition 42 *For any $\epsilon > 0$ and $m \times n$ matrix A, find the singular value decomposition $A = U\Sigma V^T$ for A; let σ_i denote the ith singular value along the diagonal of Σ.*

Then we define the **ϵ-space** *N_ϵ of A as the vector space spanned by those columns of V corresponding to singular values $\sigma_i \leq \epsilon$.*

Notice that, by definition, the null space is always a subset of the ϵ-space.

Property 43 *If $v \in N_\epsilon$, the ϵ-space of A, then $||Av|| \leq \epsilon ||v||$.*

Proof.

It suffices to see that

$$||Ax|| \leq \epsilon \tag{8.5}$$

for all unit vectors $x \in N_\epsilon$, so this is what we will show.

First, we recall in more detail what the singular value decomposition is composed of for real matrices: $A = U\Sigma V^T$. Here, U and V are real unitary matrices (so $U^T = U^{-1}$, or, equivalently, the columns of U form an orthonormal

basis). Matrix Σ is diagonal, with entries $\sigma_1, \ldots, \sigma_n$ from upper-left to lower-right, with $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$.

If V is unitary, then $||Vx|| = ||\sum_i v_i x_i|| = \sqrt{\sum_i x_i^2} = ||x||$. Let $w = V^T x$. Notice that $V^{-1} = V^T$ is unitary iff V is unitary. Then clearly $||w|| = 1$ because $||x|| = 1$.

Also notice that, for $x \in N_\epsilon$ and w such that $x = Vw$, it must be case that $w_i = 0$ for any i with $\sigma_i > \epsilon$; otherwise, x would not be a linear combination of the right singular vectors corresponding with $\sigma_i \leq \epsilon$, as stipulated by the definition of the ϵ-space. In other words, $w_i \neq 0 \implies \sigma_i \leq \epsilon$.

If $w = V^T x$, then $Ax = U\Sigma w = U(\sum_i \sigma_i w_i e_i) = \sum_i (\sigma_i w_i u_i)$. Therefore,

$$||Ax|| = \sqrt{\sum_i \sigma_i^2 w_i^2}. \tag{8.6}$$

We may now combine these observations to confirm equation (8.5) for an arbitrary unit vector $x \in N_\epsilon$. We still have $w = V^T x$. Then

$$||x|| = 1 \implies ||w|| = 1$$

and

$$(x \in N_\epsilon \ \& \ w_i \neq 0) \implies \sigma_i \leq \epsilon$$

together, along with (8.6), give us

$$||Ax|| \ = \ \sqrt{\sum_i \sigma_i^2 w_i^2} \ \leq \ \epsilon \sqrt{\sum_i w_i^2} \ = \ \epsilon,$$

which completes the proof. □

How does the idea of an $\epsilon-$space apply to our algorithm? Let us begin by defining the **first k right singular vectors** of a matrix A as the first k columns of matrix V in the singular value decomposition $A = U\Sigma V^T$. Now notice that, if $A = U\Sigma V^T$, then $B = A^T A = V\Sigma^2 V^T$. It follows that if the singular values — the diagonal entries $\sigma_1, \sigma_2, \ldots$ of Σ — are distinct, then the eigenvectors of B are exactly the *right singular vectors* of A. From this point on, we assume that the singular values of A are distinct, so that the rows of matrix Q, originally defined as the k most significant eigenvectors of A, are also the first k right singular vectors of A.

Now let $\epsilon = \sigma_{k+1}$ and define matrix N_ϵ so that

$$V = \begin{pmatrix} Q^T & N_\epsilon \end{pmatrix}, \tag{8.7}$$

where V are all the right singular vectors of A (recall that $A = U\Sigma V^T$), and Q are the first k right singular vectors. Notice that x is in the ϵ-space of A, by definition, iff $x \in \mathsf{col}(N_\epsilon)$. Equation (8.7) also reveals to us that $\mathsf{col}(Q^T)$ and $\mathsf{col}(N_\epsilon)$ are complementary vector spaces in $\mathbb{R}^n$. This gives us a nice characterization of those vectors x in the ϵ-space of A in terms of Q. We may summarize this by writing

$$x \in \mathsf{col}(N_\epsilon) \quad \Leftrightarrow \quad Qx = 0.$$

Since the algorithm returns a vector x with $Q_i x = q_i$, we may write $\hat{x}$ for the augmented vector with $\hat{x}([n] - i) = x$ and $\hat{x}(i) = -1$ to see that $Q\hat{x} = 0$, and $\hat{x} \in \mathsf{col}(N_\epsilon)$.

Therefore the stable version of our algorithm bounds the error of the approximation $A_i x \approx a_i$ by

$$||A_i x - a_i|| = ||A\hat{x}|| \leq \epsilon ||\hat{x}|| \leq \epsilon(1 + ||x||_1),$$

where $\epsilon = \sigma_{k+1}$. Since we have minimized the value $||x||_1$, we have also imposed a degree of minimization on this bound as well.

The very fast version also attempts to maintain the same matrix Q, so that the above bound is also *approximated* in this case, although the bound is no longer guaranteed to hold.

8.2.4 The block power iteration

Here we will see the main idea behind how our very fast algorithm incrementally tracks the matrix Q as part of the singular value decomposition of sliding window A.

The *block power iteration* is a simple method of computing either the left- or right-singular vectors of a matrix A. We present the traditional version of the algorithm, which computes matrix U, the left singular vectors, in the reduced singular value decomposition.

Block Power Iteration

```
Let U = orthonormalize_columns(A)
Repeat
    Let U := AA^T U
    Let U := orthonormalize_columns(U)
Until U converges
Return U
```

If $A = U\Sigma V^T$, then $A^T = V\Sigma^T U^T$ is a singular value decomposition for A^T. In other words, running the above algorithm on A^T will allow us to compute matrix V. In that case, the main step in each iteration would be

$$\text{Let } V^T := \text{ orthonormalize_rows}(V^T A^T A).$$

In our case, assuming $m \gg n$, this iteration has the advantage of multiplying at each step by $A^T A$, of size $n \times n$, rather than by AA^T, of size $m \times m$.

Another advantage is that we may run the above iteration on only the first k rows of V^T if we wish. As is consistent with our pseudocode in §8.2.2, we will let Q denote the top $k \times n$ submatrix of V^T. Now our iteration takes the form

$$\text{Let } Q := \text{ orthonormalize_rows}(QA^T A).$$

Now suppose, as in the pseudocode, that $S = QA^T A$ and that Q are the right singular values of A (so that $Q = \text{orthonormalize_columns}(S)$). Also recall that α and β are the incoming and outgoing rows of data as $A \to \hat{A}$, which we summarize with

$$\begin{pmatrix} A \\ \alpha \end{pmatrix} = \begin{pmatrix} \beta \\ \hat{A} \end{pmatrix}.$$

Then

$$\begin{aligned} Q\hat{A}^T\hat{A} &= Q(A^T A - \beta^T\beta + \alpha^T\alpha) \\ &= S + Q(\alpha^T\alpha - \beta^T\beta), \end{aligned} \tag{8.8}$$

so that the first line of our pseudocode effectively computes $\hat{S} = Q\hat{A}^T\hat{A}$. Then $\hat{Q} = \text{orthonormalize_rows}(\hat{S})$ completes this iteration of the block power method in our incremental algorithm.

8.2.5 Time complexity

In this section we will summarize those time complexity bounds which are available for our algorithms. These bounds do not appear to be optimal, primarily due to the difficulty in anticipating how many iterations will be required by our convergence techniques — finding eigenvectors or using the simplex method to solve our linear programming problem.

We claim that a single iteration of our stable algorithm runs in time $O(ckn^2 + LP_{time})$. As above, k is the number of eigenvectors of B that we track, and n is the number of time series in our data window A. We have also introduced the variable c as the number of iterations needed to compute subroutine most_sig_eig, and LP_{time} as the amount of time needed to solve our linear programming problem.

It is clear that updating $B \to \hat{B}$ takes at most time $O(n^2)$. There are several implementations of most_sig_eig available to use (see, e.g., [44] or [25]). If we use the block power method, then each block power iteration involves a matrix multiplication $S = QB$ followed by an orthonormalization $Q =$orthonormalize_rows(S); together a single step will take time $O(kn^2)$. If there are c block power iterations, then certainly this portion of the algorithm requires time $O(ckn^2)$.

Narendra Karmarkar [32] provides a linear programing algorithm which runs in time $O(n^{3.5}\log(n))$, although in practice we expect even better performance than this. In addition, we can expect better incremental speed if we use a *warm-start* technique, in which the previous time step's coefficient vector x is used as a starting point to converge to the current time step's new coefficient vector. Thus we simply summarize this portion of the time complexity as $O(LP_{time})$, and leave further evidence of incremental speedup to the experiments.

Now we claim that our very fast version runs in time $O(kn + LP_{time})$ per time step.

The first line of this version's pseudocode, captured again as (8.8) above, may be computed in time $O(kn)$ by first computing $X = Q\alpha^T$ and $Y = Q\beta^T$, and then adding $X\alpha - Y\beta$ to S to arrive at S'.

Our next line orthonormalizes the rows of S'. Any common method, such as Householder triangularization, can be used to perform this step in time $O(k^2n)$. If we also track the $k \times k$ upper-triangular matrix R so that $S^T = Q^T R$ is a QR-decomposition of S^T, then we may incrementally update the pair (Q, R) in time $O(kn)$. This technique, detailed in §12.6 of [25], relies on the fact that

our update to S can be considered as a pair of rank-one changes.

As above, we have purposefully left the linear programming time state ambiguously as $O(LP_{time})$, and recall that, in practice, this portion of the algorithm experiences a significant increase in speed by utilizing a warm-start initialization.

Thus far we have outlined pseudocode for our algorithms, demonstrated bounds on the accuracy in the stable version, and briefly analyzed the time complexity of both versions. We are now ready to empirically test these ideas on real data.

8.2.6 Experiments

In this section we will describe several experiments performed on stock price data in order to test the speed, accuracy, and stability of our algorithms. Our data consists of stock prices obtained from the NYSETAQ database via the Wharton research data services (wrds.upenn.edu). We began with the prices of all stock trades in the NYSETAQ database during the first 10 business days of 2003. We then isolated the 500 most frequently traded stocks. From these 500 stocks, we built synchronized time series with a resolution of about 25 seconds per time step; this resolution was chosen to approximate the actual average time between trades for these stocks.

In each of these experiments, a particular stock, say the ith, is time-shifted by one time step. We then use our algorithms to find, at each step, a sparse coefficient vector x so that $A_i x \approx a_i$. Since column a_i has been time-shifted forward, our vector x can be thought of as predicting the next value of that time series based on the current value of the other time series.

In our first experiment, we tracked one particular stock, chosen at random, amongst all 500 time series. Our sliding window consisted of 5000 time steps. We used the taper factor $\tau = 0.95$.

Figure 8.1 supports the role of sparsity in structural risk minimization. In the plot, we have used the absolute value of the *relative test error* r — if a is the actual stock price being predicted, and p is the predicted value, then

$$r = \frac{(p - a)}{a}.$$

Each point on the plot is the average of the absolute value $|r|$ of several relative test errors from a set of 25 consecutive iterations of the stable version of our algorithm.

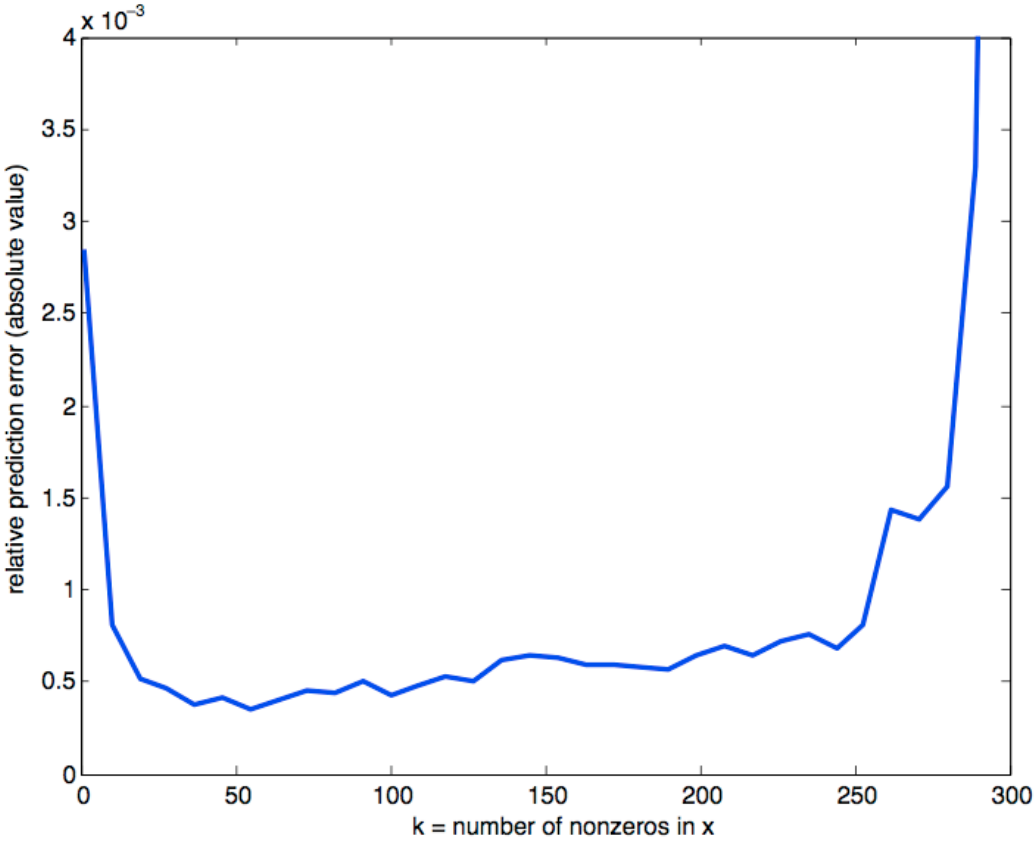

Figure 8.1: Average relative test error for different k

We remind the reader that low k corresponds with high sparsity, while higher values of k give better training accuracy. In this figure, plotting the relative test error of stock predictions against various values of k, we see that the worst (highest) relative test errors are those for which k is either very high or very low. This suggests that a good choice of k will exhibit a balance between extreme sparsity and extreme training accuracy.

Figure 8.2 illustrates the average percentage change of nonzero coefficients in the vector x between consecutive time steps. In particular, if x_t is the vector found by our algorithm (stable version) at time t, then the percentage change

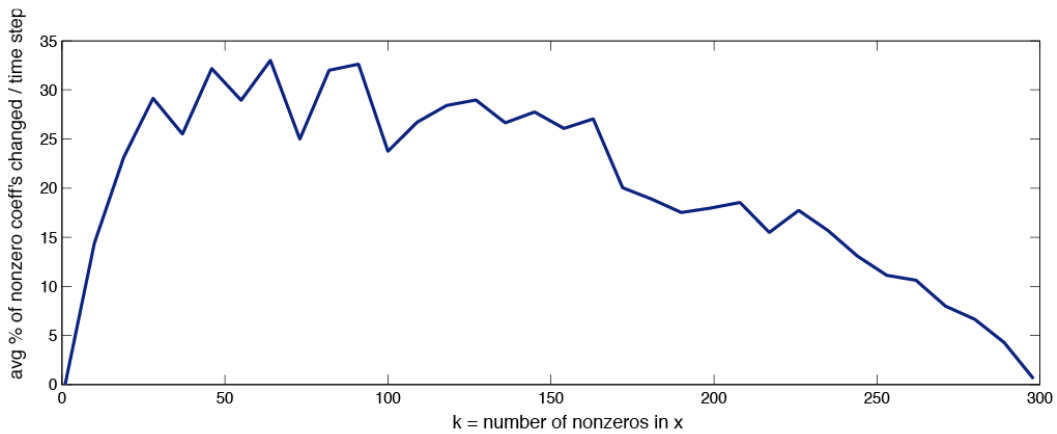

Figure 8.2: Average percentage change of nonzero coefficients (of x) for different k

p between time t and $t+1$ is given by

$$p - \frac{\#\big(\mathrm{supp}(x_t)\,\Delta\,\mathrm{sup}(x_{t+1})\big)}{k}.$$

We remind the reader that, for sets A and B, the symmetric difference

$$A\,\Delta\,B = (A - B) \cup (B - A)$$

is exactly the set of points on which A and B differ. Since the percentages in this figure never exceed 35%, this figure supports the idea that our algorithms give locally stable subset selection results.

Our next experiment observed the quality of stock price predictions as a potential basis for a trading strategy. In this experiment, we set $k = 100$ and used the first difference of the natural logarithm of stock prices as our time series. This effectively encouraged our algorithm to try to predict the *change* in stock value (as a ratio) rather than the price itself.

In figure 8.3, we see a histogram of points from a particular set of 300 predictions. For each time step, we computed the value

$$v = \mathrm{sign}(\text{predicted change}) \cdot \log(\text{actual change, as a ratio}).$$

Notice that positive v correspond to correct predictions while negative v to incorrect; moreover, the magnitude of this quantity reflects the "degree of

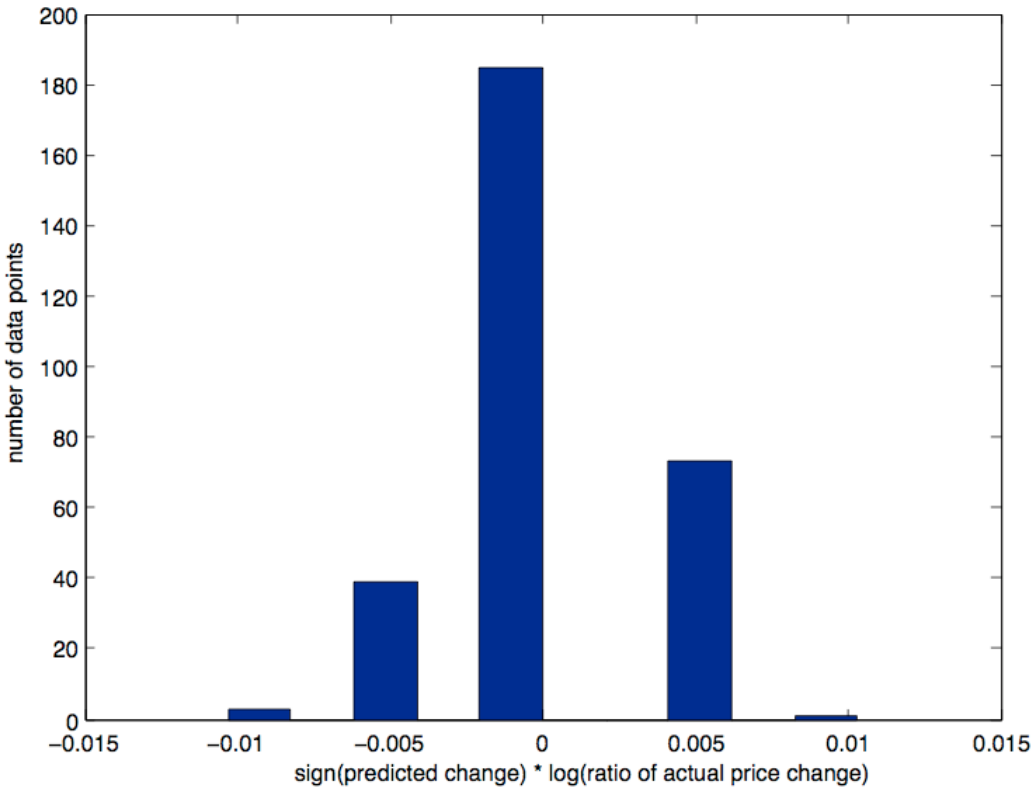

Figure 8.3: Histogram of classification quality

correctness" of this particular prediction. Thus a good predictor will achieve a histogram which is highly skewed toward positive values. And, in figure 8.2.6, we do indeed see a positive skew.

Finally, in figure 8.4, we see the running times for three versions of our algorithm: a relatively naive nonincremental implementation, the stable version, and the fast version. Clearly, the slowest of the three is the nonincremental case. Furthermore, it appears (as much as we can tell from this limited data set) that the asymptotic time complexity of both our very fast and our stable versions is much better than that of the nonincremental version.

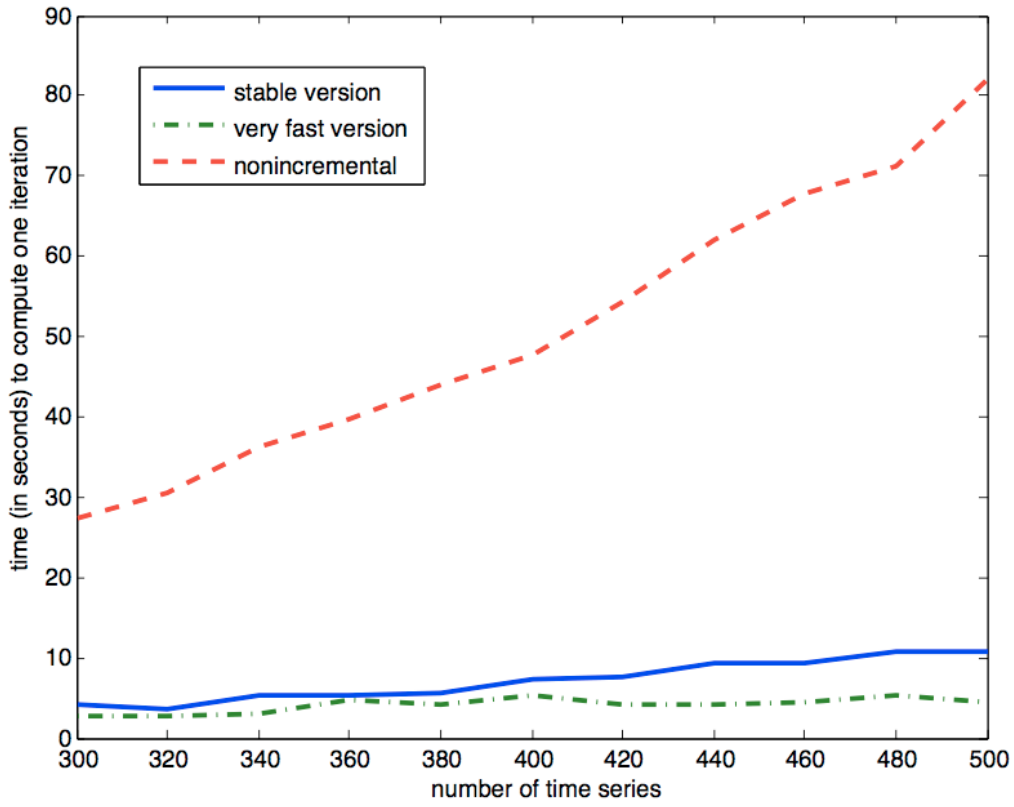

Figure 8.4: Time (in seconds) per iteration for different versions of our algorithm

Chapter 9

Conclusion

9.1 Summary

The goal of this thesis was to explore the power and applicability of sparse linear classifiers to learning problems. In particular, we have seen a general learning theoretical perspective about *why* sparsity-based hypotheses might provide better predictions, as well as *how* one may reasonably compute such patterns in an incremental fashion on time series data.

We have achieved a number of new results including the following, which are roughly ordered by importance:

1. A new bound on the VC-dimension of sparse linear classifiers, which is based upon:

2. A strongly improved version of Warren's theorem in the linear case.

3. A series of incremental algorithms for either exactly or approximately tracking a portion or all of a sparse null space for a set of time series, which could then be used to predict future behavior of the time series.

4. The fact that most matrices in any nonatomic probability distribution are completely unsparsifiable, which allowed us to show that:

5. Matrix Sparsification (and thus also Sparse Null Space) is quasi-NP-hard to approximate within a certain factor $\rho(n)$.

6. A simplified proof of the existence of size d labeled sample compression schemes, based upon:

7. A new characterization of finite maximum concept classes, which also allows for an elegant decomposition theorem of these classes (although this theorem also follows easily from several results in Sally Floyd's thesis [20]).

8. And that, for some concept classes, no faithful unlabeled sample compression scheme can possibly exist.

These contributions have been mentioned in this thesis in the following reader-friendly order:

In chapters 1 and 1.3, we motivated and introduced the primary problems, surrounding Sparse Null Space, addressed in this thesis. Then in chapter 2 we briefly reviewed the elements of learning theory to gently carry the reader into chapter 3, in which we assayed the idea of sample compression schemes. As our most abstract outlook on sparsity in learning, the existence of small sample compression schemes demonstrates that sparsity in learning is often possible; moreover, we saw that any learning algorithm based on a size d compression scheme would achieve asymptotic learning performance close to optimal.

In chapter 4, our scope narrowed to focus on sparse linear patterns, and we saw here our VC dimension bound based upon our strengthened version of Warren's theorem.

We then turned our attention, in chapter 5, to understanding what information a solution to Sparse Null Space would give us, and how it might be used for prediction. Before giving hardness results for these problems, we characterized in chapter 6 what it means for a matrix to be sparsifiable — the crown result of this chapter being that any matrix with no singular subsquares is completely unsparsifiable. Next, in chapter 7, we saw a series of reductions and equivalences among exact (non-approximating) versions of our algorithmic challenges; analyzed several previous exact algorithms; and proposed two versions of the null track algorithm for efficiently and incrementally computing a sparse null space for a set of time series. At this point, it was quite natural to consider analogous results, in chapter 8, in the realm of probabilistic and approximate approaches. There we used our completely unsparsifiable matrices result to derive a gap-preserving reduction from Min_Unsatisfy to Matrix Sparsification, revealing the hardness of approximating the latter. In addition, we saw a deterministic approximating algorithm with theoretically nontrivial, although not practically useful, bounds on the approximating factor. On a more utilitarian note, we then witnessed the possibility of L^1 minimization as a probabilistic approximator for L^0 minimization, which we then combined

with the idea of an $\epsilon-$space, a natural application of the singular value decomposition to our purposes, in order to unveil a fast very sparse incremental technique for tracking a sparse linear approximator of a particular stream from within a given set of time series.

9.2 Future Work

This work provides many opportunities for future research:

Sample compression schemes With respect to sample compression schemes, it remains undecided as to whether or not there exist size d schemes, labeled or not, for arbitrary maximal concept classes (this question originally posed by Manfred Warmuth et al [21]). We also raise the question as to whether or not a nice decomposition theorem (which can be based on the Alon-Frankl theorem 14) may be extended to the infinite case.

VC dimension bounds Our proof of theorem 20, giving an upper bound for the VC dimension of sparse linear classifiers, might offer an easy generalization to a broader set of concept classes which are in some sense "sparse." We invite the reader to explore the depths of these proof ideas.

Incremental Sparsification The null track algorithms, although *preserving* sparsity well without an augmenting sparsify subroutine, could be improved significantly with the presence of such an algorithm. In particular: suppose a *fast* method $X = \mathsf{sparsify}(Y)$ could be found which returns $X \overset{c}{\sim} Y$ such that $||X||_{\tilde{0}}$ is close to minimal, and utilizes the fact that Y was already highly sparse. Then the null track algorithms might become superior to our L^1-based incremental algorithms since null track follows the entire null space, while the L^1 technique focuses on a single time series to approximate.

9.3 Conclusion

Sparsity can clearly help achieve better compression rates and lower computational costs in many learning scenarios. We have seen here that sparsity can also help achieve better generalization error by lowering the VC dimension of the concept class within which we choose our hypothesis.

In the context of linear prediction problems, it is thus desirable to discover part or all of a Sparse Null Space for the matrix representing our data. Solving this problem in polynomial time, even approximately, appears very hard (in fact impossible depending on certain conjectures in complexity class theory); we are thus justified in applying probabilistic approximation heuristics via a new fast and efficient incremental algorithm which usually obtains excellent sparsity in a linear approximator of evolving time series data.

Notation

Sets For $A, B \subset X$,

$$A \,\Delta\, B \quad := \quad (A - B) \cup (B - A).$$

When $A \cap B = \emptyset$,

$$A \,\dot\cup\, B \quad := \quad A \cup B,$$

where the dot simply emphasizes that this is a union of disjoint sets.

For any set X and integer $d \geq 0$,

$$\begin{Bmatrix} X \\ k \end{Bmatrix} \quad := \quad \{i \subset X : |i| = k\},$$

and

$$\begin{Bmatrix} X \\ \leq k \end{Bmatrix} \quad := \quad \{i \subset X : |i| \leq k\}.$$

For any positive integer n,

$$[n] \quad := \quad \{1, 2, \ldots, n\}.$$

Concept Classes On base set X, if we have a concept class $C \subset 2^X$ and a set of points $A \subset X$, then the **restriction of** C **to** A is defined as

$$C|A \quad := \quad \{c \cap A | c \in C\}.$$

We may also define

$$\begin{aligned} C - A &:= \{c - A | c \in C\}, \quad \textit{and} \\ C^A &:= \{c - A | \forall a \subset A, c \cup a \in C\}. \end{aligned}$$

The **VC dimension** of C is

$$VCdim(C) \quad := \quad \max_{A \subset X} \{|A| : C|A = 2^A\}.$$

For integers $k \leq n$,

$$\binom{n}{\leq k} \quad := \quad \sum_{i=0}^{k} \binom{n}{i}.$$

Vectors and Matrices

$$e_i \quad := \quad i^{\text{th}} \text{ column of the identity matrix } I$$

Suppose x is a vector in $\mathbb{R}^n$.

$$\text{supp}(x) \quad := \quad \{i : x_i \neq 0\}$$

$$||x||_0 \quad := \quad \#\text{supp}(x)$$

(This "norm" obeys the triangle inequality, is nonnegative, and zero exactly when x is zero, *but* $||\lambda x||_0 \neq |\lambda| \cdot ||x||_0$ except when $\lambda = 1$, so $|| \cdot ||_0$ is not technically a norm.)

Suppose A is an $m \times n$ matrix.
If $i \in [m]$, then a_i is the ith column of A.
If $i \in [n]$, then a^i is the ith row of A.
Given $r \subset [m]$ and $c \subset [n]$, we write

$$A(r, c) \quad := \quad \text{the submatrix of } A \text{ on rows } r \text{ and columns } c.$$

The abbreviations $A(r, :)$ or $A(:, c)$ indicate that $c = [n]$ or $r = [m]$, respectively.

A **subsquare** of A is any submatrix $A(R, C)$ with $|R| = |C|$.

A **candidate submatrix** of A, written

$$A(R, C) \lhd A,$$

has columns which form a circuit and is row-inclusive in A.

$$||A||_{\tilde{0}} \quad := \quad \#\{a_{ij} : a_{ij} \neq 0\}.$$

Matrices A and B are **column equivalent**, written

$$A \overset{c}{\sim} B$$

iff there is an invertible matrix C so that $A = BC$.

Matrix A is **optimally sparse** when, $\forall\, B \overset{c}{\sim} A$, $||B||_{\tilde{0}} \geq ||A||_{\tilde{0}}$.

An $m \times n$ matrix A of rank r is **completely unsparsifiable** iff

$$\forall\, B \overset{c}{\sim} A,\ ||B||_{\tilde{0}} \geq (m - r + 1)r.$$

Linear Algebra Glossary

An $m \times n$ matrix A has m rows and n columns. An $n \times 1$ matrix x consisting of one column is a **column vector**, and written $x \in \mathbb{R}^n$. If a_i is the ith column of A, then we may write the product Ax as

$$Ax = \sum_{i=1}^{n} a_i x_i,$$

where x_i is the ith coordinate of x.

If B is an $n \times p$ matrix, and $C = AB$, then we may write c_i, the ith column of C as

$$c_i = Ab_i,$$

where b_i is the ith column of B.

The **column-row expansion** form for $C = AB$ is

$$C = \sum_{i=1}^{n} a_i b^i,$$

where b^i is the ith *row* of B. Note that each term $a_i b^i$ here is an $m \times p$ matrix, not to be confused with $a^i b_i$, which is just a scalar (here a^i is the ith row of A).

By $\mathsf{col}(A)$, called the **column space** of A, we indicate the vector space spanned by the columns of matrix A. Specifically,

$$\mathsf{col}(A) = \{Ax : x \in \mathbb{R}^n\}.$$

Similarly, $\mathsf{row}(A)$ indicates the **row space** of A:

$$\mathsf{row}(A) = \{yA : y \text{ is a row vector in } \mathbb{R}^m\}.$$

A **row vector** is a $1 \times m$ matrix (any matrix with only one row).

The **null space** of a matrix A, written $\mathsf{null}(A)$, is the vector space given by

$$\mathsf{null}(A) = \{x \in \mathbb{R}^n : Ax = 0\}.$$

The columns of A are **linearly independent** iff $\mathsf{null}(A)$ contains only the zero vector.

The **rank** of a matrix A, written $\mathsf{rank}(A)$, is the dimension of $\mathsf{col}(A)$, defined rigorously as the size of the largest set of linearly independent vectors in $\mathsf{col}(A)$. It is a fact that

$$\dim(\mathsf{col}(A)) = \dim(\mathsf{row}(A))$$

for all matrices A. A matrix has **full rank** when $\mathsf{rank}(A) = \min(m, n)$ (this makes sense since necessarily $\mathsf{rank}(A) \leq \min(m, n)$).

The **corank** of a matrix A is the dimension of $\mathsf{null}(A)$. It is a fact that, for all $m \times n$ matrices A,

$$\mathsf{rank}(A) + \mathsf{corank}(A) = n.$$

For some matrix A, it is traditional to write a_{ij} to indicate the value in the ith row and j^{th} column. A matrix is **diagonal** iff ($i \neq j \implies a_{ij} = 0$); that is, all entries not on the diagonal are zero. A matrix is **upper-triangular** iff ($i > j \implies a_{ij} = 0$).

Given a matrix A, we say that $B = A^T$ is the **transpose** of A iff $b_{ij} = a_{ji}$. We say that A is **lower-triangular** iff A^T is upper-triangular.

The $n \times n$ **identity matrix**, usually denoted with the particular letter I, has entry I_{ij} in row i and column j given by

$$I_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

It is also traditional to write e_i for the ith column of this matrix.

An $m \times n$ matrix A is **invertible** iff it is square ($m = n$) and there exists another matrix, written A^{-1} such that $AA^{-1} = A^{-1}A = I$. It is a fact that A is invertible iff its columns are linearly independent iff $\mathsf{rank}(A) = n$ iff $\mathsf{null}(A) = \{\vec{0}\}$.

The columns of a matrix Q are called **orthonormal** iff

$$q_i^T q_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise,} \end{cases}$$

where q_i is the ith column of Q. A real square matrix Q is **unitary** iff its columns are orthonormal. It is a fact that a real matrix is unitary iff $Q^T = Q^{-1}$ iff its rows are orthonormal.

Matrix U is in **reduced row echelon** form iff

- there is an increasing sequence $P = \{p_1, \ldots, p_r\} \subset [n]$ of column indices so that column $u_{p_i} = e_i$, the ith column of the identity matrix; and
- for $i \leq r$, $j < p_i \implies u_{ij} = 0$; and
- for all $i > r$, $u_{ij} = 0$,

where $r = \mathsf{rank}(U)$. The set of columns indexed by P are the **pivot columns** in U. This form is what we reduce a matrix to by Guassian elimination. Notice that any U in reduced row echelon form is also upper-triangular.

Reduced row echelon example An *elementary row operation* on a matrix A is a simple operation such as

- switching two rows,
- multiplying a row by a scalar, or
- replacing a row r by $r + s$, where s is another row in the matrix.

It is a fact that matrix B may be derived from matrix A by elementary row operations iff there is an invertible matrix X such that $B = XA$. Thus we may always algebraically represent a series of row operations by some matix X which is to be left multiplied by A (that is, our result is given by XA).

As a quick example, consider

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 4 \\ 3 & 6 & 5 \end{pmatrix}.$$

We can use the upper-left entry as a pivot in Guassian elimination to arrive at

$$B = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 2 \end{pmatrix}.$$

This matrix is upper-triangular but not yet in row echelon form — although the first column qualifies as a pivot column, the third does not, and there is no assignment of pivot columns which would allow this matrix to meet the conditions of being in reduced row echelon form. Intuitively, the problem is that the third column "goes down by two" at a time, which is not allowed in row echelon form.

So our next step will be to exercise another row operation in order to clean up the third column:

$$C = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{pmatrix}.$$

Technically, this is still not in reduced row echelon form since the third column is still not of the form e_i, although we could easily solve a system of the form $Cx = b$ at this point.

Let's eliminate the upper-right nonzero, and then divide the third column by 2 in order to arrive at:

$$D = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}.$$

At last, we have reached reduced row echelon form. Indeed, let $p_1 = 1$ and $p_2 = 3$ be our pivot columns. No column "goes down" by more than one row at a time, and those columns which do (the pivot columns) are identity columns (from an identity matrix).

Decompositions The **QR decomposition** of $m \times n$ matrix A is

$$A = QR,$$

where Q is $m \times m$ and unitary, and R is upper triangular. Every matrix has a QR decomposition.

The **LU decomposition** of $m \times n$ matrix A is

$$A = LU,$$

where L is invertible and U is in reduced row echelon form. As we have defined it, every matrix has an LU decomposition. (Many authors prefer to require that L is also lower-triangular, in which case this decomposition may not exist,

although it will when one allows a row permutation to take place. For the sake of simplicity within this thesis, we will not require L to be lower-triangular when using the LU decomposition.)

The **singular value decomposition** (SVD) of $m \times n$ matrix A is

$$A = U\Sigma V^T,$$

where U is $m \times m$ and unitary, V is $n \times n$ and unitary, and Σ is $m \times n$ and diagonal. In addition, if σ_i indicates the ith value along the diagonal of Σ, we require that $\sigma_1 \geq \sigma_2 \geq \ldots$ and that each σ_i be nonnegative. All real matrices have a real singular value decomposition. We refer to σ_i as the $i^{\mathbf{th}}$ **singular value** of A.

Norms Given two column vectors x and y, their **inner product**, sometimes written as $\langle x, y\rangle$, or just $x \cdot y$, is simply the matrix product

$$\langle x, y\rangle = x^T y.$$

A **norm** is a function mapping $\mathbb{R}^n \to \mathbb{R}$ which is nonnegative, zero only when input x is zero, scales linearly with the input, and obeys the triangle inequality. In notation:

- $||x|| \geq 0$ and $||x|| = 0$ iff $x = 0$;
- $||\lambda x|| = |\lambda| \cdot ||x||$ for any scalar λ; and
- $||x + y|| \leq ||x|| + ||y||$.

The standard norm, which we denote simply by $||x||$ is defined as

$$||x|| = \left(\sum_i x_i^2\right)^{1/2}.$$

This may also be written as $||x||_2$ in cases where the context may suggest otherwise.

The **Cauchy-Schwarz** inequality tells us that

$$\langle x, y\rangle \leq ||x|| \cdot ||y||$$

for any pair of vectors x, y.

Another norm we will use occasionally is given by

$$||x||_1 = \sum_i |x_i|.$$

Linear computations

We will briefly argue that certain key linear algebraic operations are computable in polynomial time.

Row reduction First, given matrix A, we may use Gaussian elimination to compute its decomposition $A = LU$ in polynomial time, where L is invertible and U is in reduced row echelon form.

Solving a system Next, suppose we are attempting to solve the system $Ax = b$, where A is a matrix and x, b are column vectors. Then we may decompose $A = LU$ as above, so that our system has the same set of solutions x as $Ux = L^{-1}b$. Due to the specific form of U, we may immediately decide if there is a solution x, and if so, compute the value of one such solution.

That is, in polynomial time we can both determine if system $Ax = b$ has any solutions, and find the value of a particular solution x if one exists.

Computing the null space We will say that U is in **reduced column echelon form** iff U^T is in reduced row echelon form. Since a row echelon matrix has exact $r = \mathsf{rank}(U)$ nonzero rows, it follows that a column echelon matrix has exactly r nonzero *columns*.

Given matrix A, compute the decomposition $A^T = L^T U^T$, where L^T is invertible and U^T is in reduced row echelon form. Then $A = UL$, where U is in reduced column echelon form. We may write this as $AL^{-1} = U$. Since the left $r = \mathsf{rank}(U) = \mathsf{rank}(A)$ columns of U are nonzero, we know that the right $c = \mathsf{corank}(A)$ columns of U must be zero. This means that the c rightmost columns of L^{-1} are linearly independent columns in the null space of A.

In summary, decomposing $A^T = L^T U^T$, the rightmost $c = \mathsf{corank}(A)$ columns N of L^{-1} form a full null matrix for A. This procedure clearly runs in polynomial time.

Given N, finding A with $\mathsf{null}(A) = \mathsf{col}(N)$ Given matrix N, let A^T be a full null matrix for N^T. We may find such a matrix using the procedure described immediately above. Then $N^T A^T = 0 \implies AN = 0$, so that the columns of N are in the null space of N — that is, we know that $\mathsf{col}(N) \subset \mathsf{null}(A)$. To show that $\mathsf{null}(A) \subset \mathsf{col}(N)$, it will suffice to demonstrate that $\dim(\mathsf{null}(A)) = \mathsf{corank}(A) = \mathsf{rank}(N) = \dim(\mathsf{col}(N))$.

Suppose N is $n \times c$ in size. Then A must be $m \times n$ for some m. Since A^T is a full null matrix for N^T, we know that

$$\mathsf{rank}(A^T) = \mathsf{corank}(N^T).$$

But $\mathsf{rank}(A^T) = \mathsf{rank}(A) = n - \mathsf{corank}(A)$ and $\mathsf{corank}(N^T) = n - \mathsf{rank}(N^T) = n - \mathsf{rank}(N)$ so that indeed $\mathsf{corank}(A) = \mathsf{rank}(N)$.

In summary, we have shown that choosing A^T as a full null matrix for N^T provides a matrix A with $\mathsf{null}(A) = \mathsf{col}(N)$; moreover, this procedure runs in polynomial time.

Bibliography

[1] Noga Alon. On the density of sets of vectors. *Discrete Math.*, 46(2):199–202, 1983.

[2] R.P. Anstee, Lajos Rónyai, and Attila Sali. Shattering news. *Graphs and Combinatorics*, 18(1):59–73, March 2002.

[3] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes and linear equations. In *Proceedings of the 34th IEEE FOCS*, pages 724–733, 1993.

[4] Eric B. Baum and David Haussler. What size net gives valid generalization? *Neural Computation*, 1(1):151–160, 1990.

[5] Piotr Berman and Marek Karpinski. Approximating minimum unsatisfiability of linear equations. *Electronic Colloquium on Computational Complexity (ECCC)*, 8(25), 2001.

[6] M. Berry, M. Heath, I. Kaneko, M. Lawo, R. Plemmons, and R. Ward. An algorithm to compute a sparse basis of the null space. *Numer. Math.*, 47:483–504, 1985.

[7] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. Technical Report UCSC-CRL-87-20, UC Santa Cruz, 1987.

[8] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's Razor. *Inf. Proc. Let.*, 24:377–380, 1987.

[9] Emmanuel Candes and Terence Tao. Decoding by linear programming. *IEEE Transactions on Information Theory*, 51(12):4203–4215, December 2005.

[10] S. Frank Chang and S. Thomas McCormick. A hierarchical algorithm for making sprase matrices sparser. *Mathematical Programming*, 56(1–3):1–30, August 1992.

[11] Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. Atomic decomposition by basis pursuit. *SIAM Review*, 43(1):129–159, 2001.

[12] T.F. Coleman and A. Pothen. The null space problem I. complexity. *SIAM Journal on Algebraic and Discrete Methods*, 7(4):527–537, October 1986.

[13] T.F. Coleman and A. Pothen. The null space problem II. algorithms. *SIAM Journal on Algebraic and Discrete Methods*, 8(4):544–563, October 1987.

[14] I. Dinur, G. Kindler, and S. Safra. Approximating CVP to within almost polynomial factors is NP-hard. In *Proceedings of the 39th IEEE FOCS*, pages 99–109, 1998.

[15] P. Domingos. The role of Occam's Razor in knowledge discovery. *Data Mining and Knowledge Discovery*, 3:409–425, 1999.

[16] David L. Donoho. For most large underdetermined systems of linear equations, the minimal l1-norm solution is also the sparsest. http://www-stat.stanford.edu/˜ donoho/Reports/2004/l1l0EquivCorrected.pdf, 2004.

[17] David L. Donoho and Jared Tanner. Sparse nonnegative solutions of underdetermined linear equations by linear programming. http://www-stat.stanford.edu/˜ donoho/Reports/2005/NonNegative-R5.pdf, 2005.

[18] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 1986.

[19] Uriel Feige and Daniel Reichman. On the hardness of approximating Max-Satisfy. Technical Report 119, Electronic Colloquium on Computational Complexity, 2004.

[20] S. Floyd. *On space-bounded learning and the Vapnik-Chervonenkis dimension*. Technical report TR-89-061, International Computer Science Institute, Berkeley, California, 1989.

[21] S. Floyd and M. Warmuth. Sample compression, learnability, and the Vapnik-Chervonenkis dimension. *Machine Learning*, 21(3):269–304, December 1995.

[22] Peter Frankl. On the trace of finite sets. *J. Combin. Theory (A)*, 34:41–45, 1983.

[23] J.R. Gilbert and M.T. Heath. Computing a sparse basis for the null space. *SIAM Journal on Algebraic and Discrete Methods*, 8(3):446–459, July 1987.

[24] Federico Girosi. An equivalence between sparse approximation and support vector machines. *Neural Computation*, 10(6):1455–1480, August 1998.

[25] Gene H. Golub and Charles F. Van Loan. *Matrix Computations, 2nd Ed.* The Johns Hopkins University Press, 1989.

[26] Thore Graepel, Ralf Herbrich, and John Shawe-Taylor. Generalisation error bounds for sparse linear classifiers. In *Proceedings of the thirteenth annual conference on computational learning theory*, pages 298–303, 2000.

[27] T. Grapel, R. Herbrich, and J. Shawe-Taylor. Generalization error bounds for sparse linear classifiers. In *Thirteenth Annual Conference on Computational Learning Theory, 2000*. Morgan Kaufmann, 2000.

[28] R. Gribonval, R. Figueras i Ventura, and P. Vandergheynst. A simple test to check the optimality of a sparse signal approximation. Technical Report 1661, Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), 2004.

[29] Rémi Gribonval and Morten Nielsen. Sparse representations in unions of bases. Technical Report 4642, Institut National de Recherche en Informatique en Automatique (INRIA), November 2002.

[30] A.J. Hoffman and S.T. McCormick. A fast algorithm that makes matrices optimally sparse. In *Progress in Combinatorial Optimization*. Academic Press, 1984.

[31] W. Johnson and J. Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemp. Math.*, 26:189–206, 1984.

[32] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the 16th annual ACM symposium on theory of computing*, pages 302–311. ACM press, 1984.

[33] T. Kavitha, K. Mehlhorn, D. Michail, and K. Paluch. A faster algorithm for minimum cycle basis of graphs. In *Automata, Languages, and Programming: 31st International Colloquium, ICALP 2004 Proceedings*, Lecture Notes in Computer Science, pages 846–857. Springer, 2004.

[34] Telikepalli Kavitha and Kurt Mehlhorn. A polynomial time algorithm for minimum cycle basis in directed graphs. In *STACS 2005: 22nd Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, pages 654–. Springer, 2005.

[35] S. Khanna, M. Sudan, and L. Trevisan. Constraint satisfaction: the approximability of minimization problems. In *Proceedings of the twelfth annual IEEE conference on computational complexity*, pages 282–296, 1997.

[36] D. Kuzmin and M. Warmuth. Unlabeled compression schemes for maximum classes. In *Proceedings of the 18th Annual Conference on Computational Learning Theory (COLT '05)*, pages 591–605, June 2005. Bertinoro, Italy.

[37] J.G. Oxley. *Matroid Theory*. Oxford University Press, 1992.

[38] L. Rónyai, L. Babai, and M. Ganapathy. On the number of zero-patterns of a sequence of polynomials. *J. of the Amer. Math. Soc.*, 14(3):717–735, 2001.

[39] N. Sauer. On the density of families of sets. *J. Combinatorial Theory, Series A*, 13:145–147, 1972.

[40] S. Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific J. Mathematics*, 41:247–261, 1972.

[41] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, August 2004.

[42] Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society Series B*, 67:91, February 2005.

[43] A. Topcu. *A contribution to the systematic analysis of finite element structures through the force method*. PhD thesis, University of Essen, Essen, Germany, 1979. In German.

[44] Lloyd N. Trefethen and III David Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematic (SIAM), 1997.

[45] J.A. Tropp. Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, October 2004.

[46] J.A. Tropp. Just relax: convex programming methods for subset selection and sparse approximation. Technical Report 0404, University of Texas at Austin, 2004.

[47] L. Valiant. A theory of the learnable. *Comm. ACM*, 27(11):1134–1142, 1984.

[48] V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[49] Vladimir Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[50] Eric W. Weisstein. *CRC Concise Encyclopedia of Mathematics, 2nd Edition*. CRC Press, 2002.

[51] X. Zhao, X. Zhang, T. Neylon, and D. Shasha. Incremental methods for simple problems in time series: Algorithms and experiments. In *Ninth International Database Engineering and Applications Symposium (IDEAS 2005)*, pages 3–14, 2005.