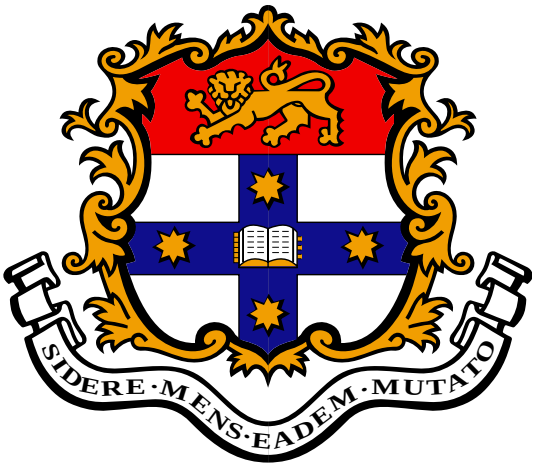
Random projections and applications to dimensionality reduction

ADITYA KRISHNA MENON
SID: 200314319



Supervisors: Dr. Sanjay Chawla, Dr. Anastasios Viglas

This thesis is submitted in partial fulfillment of
the requirements for the degree of
Bachelor of Science Advanced (Honours)

School of Information Technologies
The University of Sydney
Australia

1 March 2007

Abstract

Random projections are a powerful method of dimensionality reduction that are noted for their simplicity and strong error guarantees. We provide a theoretical result relating to projections and how they might be used to solve a general problem, as well as theoretical results relating to various guarantees they provide. In particular, we show how they can be applied to a generic data-streaming problem using a *sketch*, and prove that they generalize an existing result in this area. We also prove several new theoretical results relating to projections that are of interest when trying to apply them in practise, such as analysis on the reduced dimension guarantees, and the error incurred on the dot-product.

Acknowledgements

I thank my supervisors for all their help during the year, and their endless patience in answering all my questions, no matter how trivial, thus helping me navigate the sometimes treacherous theoretical analysis of projection. I also thank them for putting up with my endless claims of having derived a new result on dimension bounds, only to have it disproved the next day!

I also thank my family and friends for helping me get through this sometimes stressful honours year in mostly one piece. Special thanks go to S for helping me rediscover interests I thought were long gone.

CONTENTS

List of Figures

List of Tables

The University of Sydney **School of Information Technologies**

Unit of Study: _____

Assignment/Project Title Page

Individual Assessment

Assignment name: _____

Tutorial time: _____

Tutor name: _____

Declaration

I declare that I have read and understood the *University of Sydney Student Plagiarism: Coursework Policy and Procedure,* and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that understand that failure to comply with the *Student Plagiarism: Coursework Policy and Procedure* can lead to severe penalties as outlined under Chapter 8 of the *University of Sydney By-Law 1999* (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

Student Id: _____

Student name: _____

Signed: _____

Date: _____

CHAPTER 1

Introduction

In many problems, the data available to us can be thought of as representing a number of distinct objects or items, each with a certain number of attributes. As a simple example, suppose we are analyzing text documents, and in particular, are trying to get information about specific words that commonly occur in some subsets of the documents. We can think of this data as consisting of many objects in the form of documents, where each document has the words that are in it as attributes.

When we have such data, we can interpret it in high-dimensions by simply thinking of each object as a point, and each attribute of that object as a dimension. This is convenient because it lets us use a matrix representation of the data,

$$A = \begin{bmatrix} \mathbf{u_1} \\ \vdots \\ \mathbf{u_n} \end{bmatrix}$$

where each $\mathbf{u_i}$ is an object in our data-set, and has, say, d dimensions (and is thus a d-dimensional vector).

As a result, analysis of the data can be done using this powerful matrix based representation - "powerful" because there are innumerable techniques known to extract information from matrices, ranging from the components that give the most change, to the minimal space that is spanned by the objects (rows) it contains.

Conceptually, such a representation might be thought of as fairly strong. But this conceptual simplicity can hide a very common problem that arises in practise.

	Cat	Dog	...	England
Guide to cats	1	0	...	0
Household pets	1	1	...	0
Tourist guide	0	0	...	1

TABLE 1.1: Example of documents and terms that have the object/attribute interpretation. Here, a 1 denotes that a document has a particular attribute

1.1 Problem

The problem posed by high-dimensional data is trivial to state, but not so simple to solve. Simply put, the problem is that the number of dimensions of our data can be extremely large - several thousands, and in some cases, even millions of dimensions. Such a high number of dimensions may be enough to either make storing the data in memory infeasible, or analysis of the data intractable. The latter is more common, because there are many algorithms of practical interest that do not perform efficiently when we start getting a very large number of dimensions (for example, clustering and searching problems).

This is related to a deeper problem called the "curse of dimensionality" (Bellman, 1961) that has plagued researchers in a wide number of fields for many years now. Loosely, it says that for any data, as we increase the number of dimensions it possesses, the complexity involved in working with it increases at an exponential rate. This makes all but the most trivial manipulations and analysis impractical[1]. This poses a dilemma, because naturally we would like to be able to use as much information as is available to us - and yet, the "curse" is that as we collect more information, we spend dramatically more time trying to make sense of it!

A simple example of the "curse" is when we try to get a representative sample of the set $[0, 1]^d$ (see Figure 1.1)[2]. In one dimension, this is just an interval - and we can see that 10 samples are enough to get a good representation of this interval (for example, it is very unlikely that all samples will be on one half of the interval). When we move to two dimensions, 10 samples is reasonable, but we can see that it is now more likely that we will leave some areas unexplored. Once we get to three dimensions, we notice that these 10 samples start to get "lost in space", and do not provide a good sampling! One can imagine the problem involved with several thousands of dimensions - it is basically that by linearly increasing the number of dimensions, we are forcing an exponential increase in the number of points required to approximate the new space.

The problem has become more pressing in recent times, as today nearly every field has large volumes of data that need to be made sense of - be it sets of image-data that need to be analyzed, or sets of users with attributes where patterns need to be found. Given that the amount of data itself is growing at an exponential rate, it is not difficult to see why the problem of dimensionality is very pertinent today, and that some solutions need to be sought.

[1] It should be noted that while it is obvious that there be some kind of growth, the fact that it is *exponential* means that problems can become intractable

[2] This is motivated by a similar example, attributed to Leo Breiman

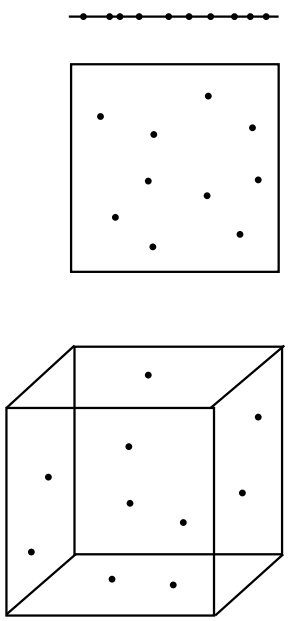


FIGURE 1.1: The curse of dimensionality shows itself when trying to cover the set $[0,1]^d$

1.2 Solution

Fortunately, there has been work done on trying to get around the problems of high-dimensionality. Much like how the problem of high-dimensionality can be stated simply, so too can the solution - it is simply to produce an approximation to the original data that has fewer dimensions, but still has the same structure as the original data[3]. Such a technique is called a method of *dimensionality reduction*.

More formally, as the name suggests, dimensionality reduction involves moving from a high-dimensional space to a lower-dimensional one that is some sort of approximation to it. So, instead of performing our analysis with the original data, we work on this low-dimensional approximation instead. The intent here is that by reducing the number of dimensions, we reduce significantly the time that is needed for our analysis, but, at the same time, we keep as much information as we can about the original data, so that it is a good approximation.

Obviously, it is not possible to be completely faithful to the original data and still have fewer dimensions (in general). Therefore, there is always a trade-off between the number of

[3]We will explain this in the next chapter

dimensions and the accuracy of the reduced data. We are *accepting* that we will not get 100% accurate results, but in many applications[4], a slight inaccuracy is not an issue.

There are quite a few ways of performing dimensionality reduction, and one of the more popular approaches of late has been random projections (Vempala, 2004). Random projections have emerged as a technique with great promise, even though they are rather counter-intuitive. As the name suggests, the method involves choosing a *random* subspace for projection - this is completely independent of what the input data is! It offers a substantially better runtime than existing methods of dimensionality reduction, and thus is immediately attractive, but perhaps surprisingly, it has shown to be remarkably accurate as well (Johnson and Lindenstrauss, 1984), with error-rates being well within acceptable parameters for a number of problems.

1.3 Motivation

The power of random projections comes from the strong theoretical results that guarantee a very high chance of success. However, there are still several theoretical properties that have not been fully explored, simply because the technique is so new (in relative terms). Certainly a complete understanding of projections is rewarding in its own terms, but it also offers the possibility of being able to better apply projections in practise.

Further, since the field has only started to grow quite recently, there are potentially many problems that could benefit from the use of projections. What is required here is a connection between the theoretical properties of projections and the needs of a problem, and there might be quite a few problems for which this link not been made thus far.

These observations served as the motivation for this thesis, which looks at random projections from both a theoretical and a practical angle to see whether we can

- Improve any of the theoretical results about random projections, or formulate new theoretical statements from existing theory
- Apply projections to some specific, real-world problem(s)

1.4 Aims and objectives

Leading on from the motivating questions, this project had the following broad aims and objectives:

[4]The best example is the class of approximation algorithms that are used to solve (with high probability) intractable or NP-hard problems

- Applying random projections to a data-streaming problem.
- Derive theoretical guarantees related to random projections and dot-products.
- Study the theoretical results related to random projections and the so-called "lowest reduced dimension".
- Study the behaviour of projections for special types of input (namely, normal input data, and sparse data).

1.5 Contributions

Briefly, the major contributions made in the thesis are as follows:

- Several simple, but new results relating to preliminary properties of random projections (§3.1, §3.2).
- A theoretical proof that one can apply random projections to data-streaming problems, and that it is superior to an existing solution (Indyk, 2006) to the problem (§4.4).
- A new upper-bound on the distortion of the dot-product under a random projection (§5.2, §5.3).
- An experimental confirmation of the theoretical guarantees made about projections and data-streams (§6.1).
- The derivation of a necessary and sufficient condition for a particular projection guarantee to be "meaningless" (§7.1).
- A signficant improvement on existing dimension bounds for a restricted range of error values (§7.3).
- A simplified proof of Achliopats' theorem based on analysis of input distribution (§8.2).
- A model of sparse data, and a preliminary analysis of the behaviour of projections for such data (§9.1).

1.6 Structure of the thesis

Since there are two distinct aspects to the thesis, namely, an application to data-streams and a series of theoretical results (that are not restricted to the streaming problem), the thesis has been split into three parts.

The first part contains some preliminary facts and results needed for the next two parts, and includes:

- Chapter 2 - provides an introduction to the field of random projections, and attempts to give some intuitive understanding of what projections are, and they work.
- Chapter 3 - goes over some preliminary properties of random projections that are used in later theoretical analysis of the projections. Some of these facts are well-known (and are almost "folk-lore" in random projection literature), but others are to our knowledge new results.

The second part details the application of random projections to data-streams, and entails:

- Chapter 4 - describes the theory behind a novel application of random projections to a data-streaming problem, where we need to make quick and accurate approximations to high-volume queries. The projection approach is compared to an existing sketch.
- Chapter 5 - looks at the dot-product, and how it behaves under random projections. It includes a novel bound on the distortion of the dot-product under a random projection, which motivates the use of the sketch in the previous chapter to also estimate dot-products.
- Chapter 6 - provides experimental results showing that projections provide superior performance to an existing sketch for data-streams.

The third, and final part, consists of general theoretical results relating to projections, and includes:

- Chapter 7 - addresses in more detail the dimension-bound guarantees that projections provide, and includes some novel results on improving the tightness of these bounds for some restricted cases.
- Chapter 8 - gives a theoretical analysis of dimension bounds when the input comes from a special distribution, and includes new results for this case.
- Chapter 9 - discusses the notion of input and projection sparsity, and why it is an interesting property for random projections. We present some results that show that input sparsity has no impact on projections, whereas projection sparsity does.

Chapter 10 offers a summary of the thesis, and points to areas of further work.

Part 1

Preliminaries

CHAPTER 2

Background

2.1 Dimensionality reduction

Dimensionality reduction, as introduced in §1.2, refers to the process of taking a data-set with a (usually large) number of dimensions, and then creating a new data-set with a fewer number of dimensions, which are meant to capture only those dimensions that are in some sense "important". The idea here is that we want to preserve as much "structure" of the data as possible, while reducing the number of dimensions it possesses; therefore, we are creating a lower-dimensional approximation to the data. This has two benefits:

(1) It reduces the work we need to do, since the number of dimensions is now more manageable
(2) Since the structure is preserved, we can take the solution for this reduced data set as a reliable approximation to the solution for the original data set

Notice that the precise meaning of the "structure" of the data really depends on the nature of our problem. For instance, suppose we are trying to solve a high-dimensional nearest neighbour problem. Our data-set is a collection of points, and we can think of the structure of the data as being represented by *distances* between points (certainly the specific coordinates do not matter, as we can just translate and scale axes as required). This is not the only definition of structure, though. The Euclidean distance might not be meaningful for a binary matrix, representing the state of various objects. For such a problem, the Hamming-distance might be a more appropriate measure.

In summary, a method of dimensionality reduction tries to preserve the "structure" of data, for some nice definition of "structure" that depends on the context we are working in. This obviously means that the specific method of dimensionality reduction we choose depends a lot on what we are trying to get out of the data.

To illustrate some ideas of dimensionality reduction, it is illuminating to focus our attention on data that represents points in the plane, and think about the structure that manifests here.

2.1.1 A simple example

For instance, consider the plane $z = 1$. An arbitrary point p on this plane will be of the form $(x, y, 1)$. Suppose we have picked some arbitrary points from this plane, $\{p_1, p_2, \ldots p_n\}$, and this is our $n \times 3$ data-set A.

Now, is it harmful, in terms of the structure of the point-set, if we throw away the z-coordinate? Conceptually, we would agree that the answer is no, but what do we mean by "structure"? A natural definition is that of the distances between points, and this is certainly preserved:

$$||(x_1, y_1, 1) - (x_2, y_2, 1)||^2 = ||(x_1, y_1) - (x_1, y_1)||^2$$

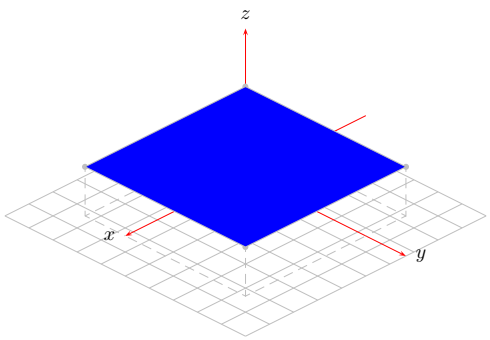


FIGURE 2.1: Projecting the plane $z = 1$ onto the xy-axis

It is easy to see that if we consider a perturbed situation, where the z-coordinate is bounded by $\pm\epsilon$, we can again disregard this coordinate, this time incurring an $O(\epsilon)$ error.

In this situation, then, it is obvious enough that the really "important" dimensions are the first and second dimension. The third dimension does not really contribute anything to the *structure* of the data, and so if we only considered the first two dimensions, the structure would remain the same.

It is important to note that by discarding the third coordinate, we have essentially projected the data onto the plane $z = 0$ i.e. the xy plane. In fact, this is a very common method of trying to do dimensionality reduction. We will get an intuitive sense of why this is so by considering the following example, where the data exhibits a less artifical structure.

2.1.2 Another example of structure preservation

Suppose we have a data-set that is vaguely elliptical, as shown in Figure 2.2.

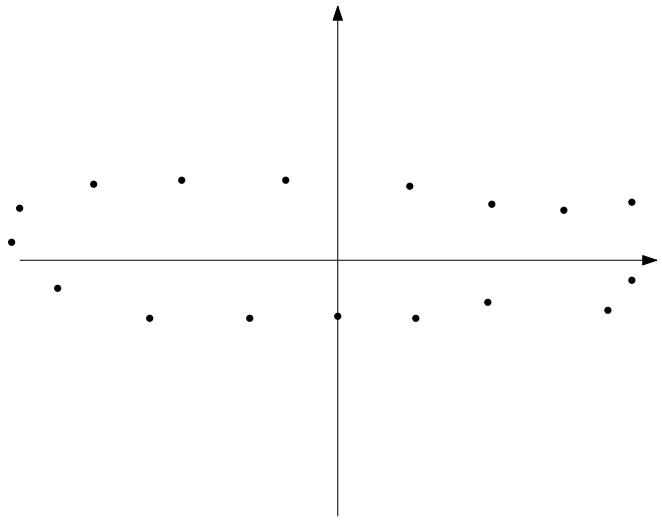


FIGURE 2.2: Simple data set

We can see that there is not much variance among the y-coordinates, but clearly there is a lot of variance among the x-coordinates. We saw with the previous example that by discarding a coordinate, we are simply projecting onto a plane. In this case, since there are two dimensions, by discarding one we are essentially projecting onto a line (either the x or y axis).

Due to the nature of the data-set, a projection onto the x-axis will preserve distances fairly well. This can be obviously deduced by looking at the projected points in Figure 2.3.

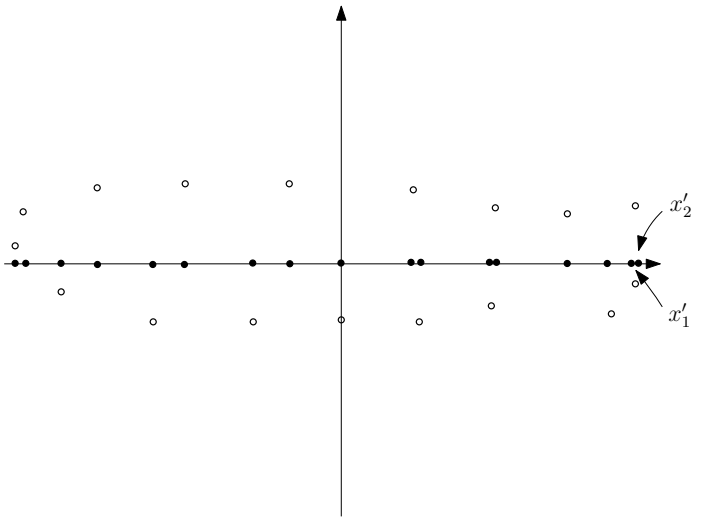


FIGURE 2.3: Projection onto the x-axis is fairly accurate

Of course, note that points that have nearly equal x-coordinates will get mapped very close together. So, points x_1' and x_2' appear to be very close in the projected space, when in the original space they could be arbitrarily far apart. But the point here is that there is far more variance along the x-axis than along the y-axis; hence, this projection is actually fairly good, given the restrictions we have (one dimension can only express so much!).

However, if we projected along the y-axis, we would expect to incur a heavy error, because now our distances have become very inaccurate, as indicated in Figure 2.4.

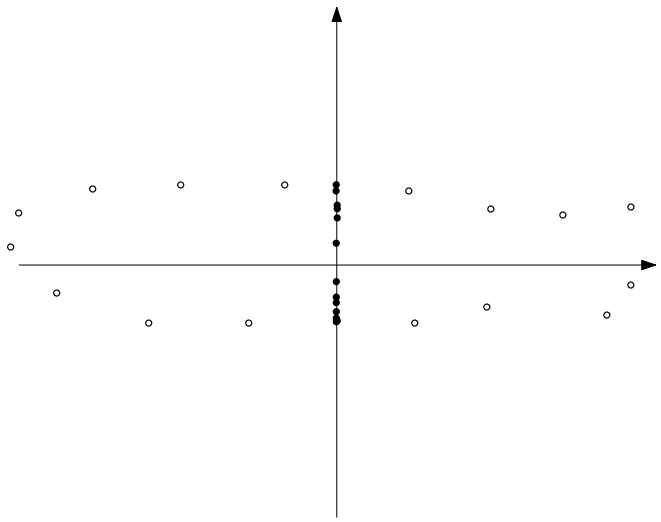


FIGURE 2.4: Projection onto the y-axis is not a good idea

We have bunched too many points together! Clearly, this is worse than the projection along the x-axis.

So, for this elliptical data-set, we again see that we can project onto a line (in this case, the x or y axis) to reduce the number of dimensions, but also keep the structure of the data. It also turns out that some projections are better than others - projecting onto the x axis is much better than projecting onto the y axis!

We will take the simple example a little further, and see that we needn't think of projections as "throwing away" a dimension, but that in fact they might combine important dimensions.

2.1.3 A final example

As a final example, let us suppose we now rotate the data by some angle, so that we still have the elliptic shape, but not along the x or y axis. For simplicity, let us take the case where the data is rotated by 45 degrees.

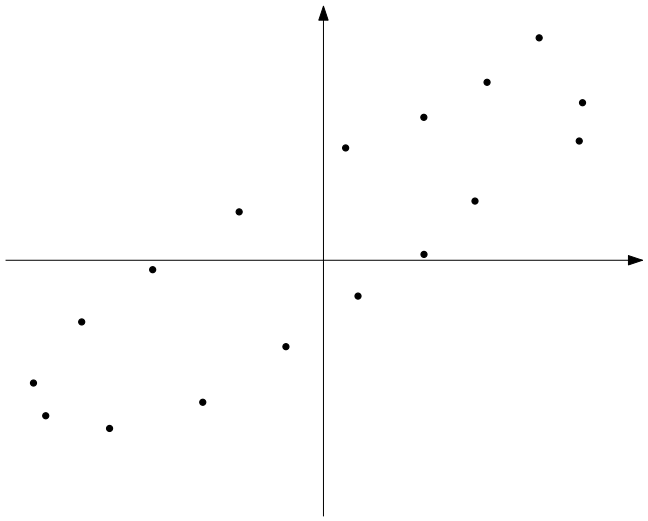


FIGURE 2.5: Data-set after rotation

Now, projecting onto either x or y axis will yield unsatisfactory results. This means that we can't just "throw away" a dimension. But, we can try to *combine* the dimensions together somehow. Suppose that instead of throwing away a dimension, we try to project onto the "major axis", which is the line $y = x$ as shown in Figure 2.6.

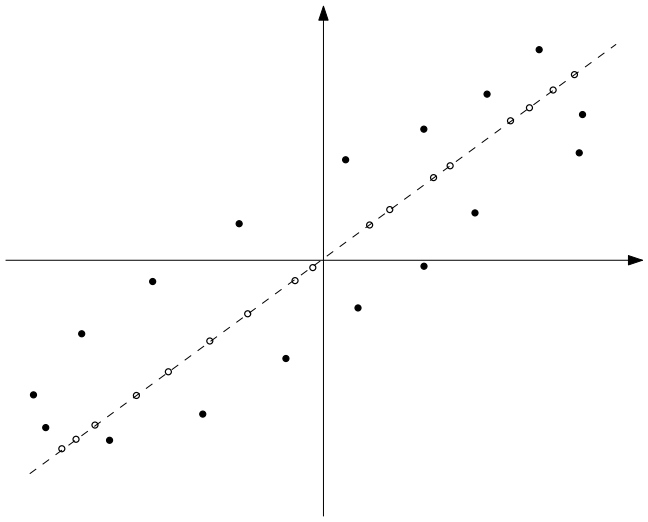


FIGURE 2.6: Projecting the rotated data-set

We see that this yields good results as far as distances go. In fact, it can be shown that all we have done here is create a new dimension by mapping

$$(x_0, y_0) \mapsto \frac{x_0 + y_0}{2}$$

We can make the observation here that in both this and the original data-set, the best projection *captured the most variance in the data*. This is in fact the motivation for methods of dimensionality reduction such as PCA (which is discussed in §2.2). Importantly, this example shows that dimensionality reduction is not just about throwing away dimensions; it is about finding the dimensions that are important, and then either combining them together to reduce the total number of dimensions, or throwing away the ones that are not important.

2.1.4 Taking this further

Techniques of dimensionality reduction take this simple idea and make it more general, allowing us to preserve other types of structure. Of course, real problems are not as simple and obvious as the example given above, and so there is the question of *how* the important dimensions are found. A lot of methods of dimensionality reduction operate using the approach outlined in the examples, namely, projecting onto a subspace. Therefore, the aim is to find the "best", or at least a good hyper-plane to project onto. In the course of this chapter, we will look at two examples of dimensionality reduction that are based on this idea.

2.1.5 Technical note

It should be noted that we mentioned that dimensionality reduction is concerned with the approximate preservation of structure. Note that this does not say that the reduced data can serve as a flawless, compressed version of the original data. Indeed, there is no perfect way for us to go from our projected data to our original - dimensionality reduction is essentially one-way! The point is that we are trying to choose our projected data so that we don't *have to* go back to our original space, and that the projected data serves as a good-enough approximation of the original. We are essentially forfeiting the specifics concerning our original data once we move into a projected space.

In other words, the preservation of structure is at the expense of the explicit data. For instance, suppose our input matrix represents the coordinates of n high-dimensional points. A method of dimensionality reduction that, say, preserves Euclidean distances can be used to answer the question "What is the distance between the first and second point?" with some degree of accuracy. However, they *cannot* be used to answer the question "What is the 32767th coordinate of the first point?"! From the point of view of the reduction, such information constitutes uninteresting details that are irrelevant as far as the over-arching structure of the data goes.

2.2 PCA

Principal Component Analysis (PCA) is a very popular method of dimensionality reduction (Jolliffe, 1986) that uses a conceptually simple idea. Suppose we have a high-dimensional data-set in the form of a matrix, A. The dimensions can be thought of as directions that the data varies along. So, for example, the first dimension over all rows by itself is just a line that each of the points (rows) run along.

If we want to reduce the dimensionality of the data, then what we really want to do is find an approximate "basis" to the set of directions. That is, we want to find only the cruical dimensions that serve as the building blocks for other dimensions - the idea here is that the dimensions we throw away are in some sense duplicates of the dimensions we keep, because they can be reconstructed from the small set of basis dimensions.

Equivalently, we want to find the dimensions of maximal variance - the points are approximately constant along the other dimensions, and so these are in some sense less important. Our reasoning is that if, say, there is a dimension along which each of the points is constant, then we would not lose a whole lot by throwing that dimension away. But if there is absolutely no pattern along a single dimension, and the points take on a gamut of values in that component, then it might not be a good idea to throw away that information. Note that this is similar to the simple examples of dimensionality reduction we presented earlier.

2.2.1 How PCA works

2.2.1.1 Theory

The theoretical idea behind PCA is that we find the *principal components* of the data, which correspond to the components along which there is the most variation. This can be done using the *covariance matrix*, AA^T for our input matrix A, as follows.

Suppose we are able to find the eigenvalues λ_i of the covariance matrix. Then, we can store them in a diagonal matrix

$$L = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ & & \vdots & \\ 0 & 0 & \dots & \lambda_n \end{bmatrix}$$

The eigenvectors $\mathbf{v_i}$ of the matrix XX^T must of course satisfy

$$XX^T\mathbf{v_i} = \lambda_i\mathbf{v_i}$$

So, if we write the eigenvectors of the data as the rows of a matrix P, then we have the system

$$XX^TP = LP \tag{2.1}$$

It can be shown that the columns of the matrix P correspond to the principal components of the original matrix (Golub and Loan, 1983), and hence captures the directions of most variance.

2.2.1.2 Finding the principal components

The most common way of finding the principal components is via a technique known as SVD, or the *singular value decomposition* of a matrix, which relies on the above theory to provide a typically simpler approach. This decomposition asserts the following - for any matrix A, we can find orthogonal[1] matrices U, V, and a diagonal matrix D, so that

$$A = UDV^T$$

In fact, it turns out that we have the following.

FACT. *The diagonal matrix D has entries σ_i, where*

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$$

These values σ_i are known as the singular values of the matrix A

It turns out that the singular values are the squares of the eigenvalues of the covariance matrix (Golub and Loan, 1983). As a result,

$$\begin{aligned} XX^T &= (UDV^T)(UDV^T)^T \\ &= (UD)V^TV(D^TU^T) \\ &= UDD^TU^T \text{ since } VV^T = I \text{ by definition} \end{aligned}$$

So, by Equation 2.1

$$P = U, L = DD^T$$

Therefore, the SVD provides a direct way to compute the principal components of the data.

[1] That is, $UU^T = VV^T = I$

2.2.1.3 Principal components and dimensionality reduction

So, how is it that this information is used for dimensionality reduction? One point to be made is that the spaces spanned by U, V are quite integral to the space occupied by A. But the main theorem is a guarantee on the goodness of the approximation that we can derive from this decomposition.

THEOREM 1. *Suppose we have the singular value decomposition of a matrix A*

$$A = UDV^T$$

Let A_k denote the matrix

$$A_k = U_k D_k V_k^T$$

where we are only consider the first k columns of A. Then, we have that A_k is the best rank-k approximation to the original matrix A:

$$||A - A_k||_F = \min_{rank(B)=2} ||A - B||_F$$

where $||.||_F$ denotes the Frobenius norm of a matrix,

$$||A||_F = \sum_i \sum_j |a_{ij}|^2$$

PROOF. See (Papadimitriou *et al.*, 1998). □

The matrix A_k is known as the *truncated SVD* of the matrix A.

So, it follows that, by construction, PCA is the best linear method of dimensionality reduction - no other linear technique can give us a better answer in terms of mean-square error.

2.3 Random projections

2.3.1 What is a random projection?

Concisely, random projections involve taking a high-dimensional data-set and then mapping it into a lower-dimensional space, while providing some guarantees on the approximate preservation of distance. As for how this is done, suppose our input data is an $n \times d$ matrix A. Then, to do a projection, we choose a "suitable" $d \times k$ matrix R, and then define the projection of A to be

$$E = AR$$

which now stores k-dimensional approximations for our n points (it is clear that the matrix E is $n \times k$). We look at what "suitable" means in §2.5.

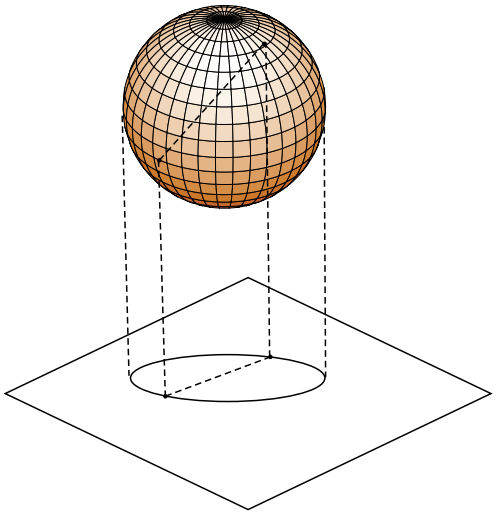


FIGURE 2.7: An example of what we mean by "projecting" an object to a lower space - in this case, a sphere in 3D going to a plane in 2D. In this case, the indicated distance will be roughly preserved in the projection

The mere fact that something like this can be done is in itself remarkable, and is thanks to the Johnson-Lindenstrauss lemma (Johnson and Lindenstrauss, 1984), which says the following.

THEOREM 2. *(Johnson and Lindenstrauss, 1984) Suppose we have an arbitrary matrix $A \in \mathbb{R}^{n \times d}$. Given any $\epsilon > 0$, there is a mapping $f : \mathbb{R}^d \to \mathbb{R}^k$, for any $k \geq 12\frac{\log n}{\epsilon^2}$, such that, for any two rows $\mathbf{u}, \mathbf{v} \in A$, we have*

$$(1-\epsilon)||f(\mathbf{u}) - f(\mathbf{v})||^2 \leq ||\mathbf{u} - \mathbf{v}||^2 \leq (1+\epsilon)||f(\mathbf{u}) - f(\mathbf{v})||^2$$

There are a few interesting things about this lemma that are worth noting.

- The lemma says that if we reduce to $k = O(\log n/\epsilon^2)$ dimensions, we can preserve pairwise distances upto a factor of $(1 \pm \epsilon)$. Notice that this means that the original dimension, d, is irrelevant as far as the reduced dimension goes - it is the *number of points* that is the important thing. So, if we just have a pair of points, then whether they have a hundred or a million dimensions, the guarantee is that there is a mapping into $O(1/\epsilon^2)$ dimensions that preserves distances

- The lemma says that we can find such an f for *any* data set, no matter how convoluted! Of course, this f is not the same for every set, but nonetheless, this is quite an amazing result!

It is important to note that the lemma only talks about the *existence* of such a mapping, and does not say how we actually find one. There has been work on finding such a mapping explicitly, and one successful approach is the work of (Engebretsen *et al.*, 2002), which resulted in an $O\left(dn^2(\log n + \frac{1}{\epsilon})^{O(1)}\right)$ time algorithm.

In practise, fortunately, there are ways to find a mapping that is "almost" as good, and which can be done in one-pass of the projection matrix (that is, we only pay for construction time - there is no other overhead). Achlioptas (Achlioptas, 2001), for instance, provided a way to find a mapping in constant time that provides the above guarantees, but with some explicit probability. However, the probability of success can be made arbitrarily large, making the result very useful. His result was the following.

THEOREM 3. *(Achlioptas, 2001) Suppose that A is an $n \times d$ matrix of n points in $\mathbb{R}^d$. Fix constants $\epsilon, \beta > 0$, and choose an integer k such that*

$$k \geq k_0 := \frac{4 + 2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n$$

Suppose now that R is a $k \times d$ matrix with entries r_{ij} belonging to the distribution

$$r_{ij} = \sqrt{3}. \begin{cases} +1 & p = 1/6 \\ 0 & p = 2/3 \\ -1 & p = 1/6 \end{cases}$$

Define the $n \times k$ matrix $E = \frac{1}{\sqrt{k}} AR$, which we use as the projection of A onto a k-dimensional subspace. For any row $\mathbf{u}$ in A, let us write $f(\mathbf{u})$ as the corresponding row in E. Then, for any distinct rows $\mathbf{u}, \mathbf{v}$ of A, we have

$$(1 - \epsilon)||\mathbf{u} - \mathbf{v}||^2 \leq ||f(\mathbf{u}) - f(\mathbf{v})||^2 \leq (1 + \epsilon)||\mathbf{u} - \mathbf{v}||^2$$

with probability at least $1 - n^{-\beta}$.

So, note that Achlioptas only says "I will *most probably* preserve all pairwise distances" - but his "most probably" can be thought of as "arbitrarily likely", since we can control the parameters ϵ, β to get, say 99.99% chance of no distortion. The advantage here is that we have

an immediate way of generating the random matrix, and a guarantee that there is no distortion 9,999 times out of 10,000 is good enough for most practical setings.

2.3.2 Why projections?

There are quite a few reasons why random projections have proven to be a very popular technique of dimensionality reduction, and they come back to the technique's simplicity and strong guarantees.

Firstly, unlike other methods of dimensionality reduction, such as PCA, random projections are quite easy to do, as they just require a matrix multiplication with a matrix that can be generated using a fairly simple procedure. In contrast, with PCA, one way to find the projected data is to compute the covariance matrix, decompose it into its singular value form, and then choose the top k eigenvectors from this decomposition. The second step is in fact not trivial, and so PCA, while accurate, requires some effort to be used.

Another property of projections that is appealing is that they manage to approximately preserve individual lengths, and pairwise distances - specifically, they approximately preserve the lengths of the original points, as well as the distances between the original points with arbitrary accuracy. In fact, Papadimitriou notes in his introduction to (Vempala, 2004) -

> "So, if distance is all you care about, there is no need to stay in high dimensions!"

Projections also tell us what the reduced dimension needs to be to achieve a certain error in the accuracy of our estimates, which is very useful in practise. This lets us answer questions of the form - if I have a data-set, and want to project it so as to reduce its dimension, what is an appropriate choice for the reduced dimension k that will let me preserve distances upto a factor of ϵ? By contrast, with most others methods of dimensionality reduction, there is no way to know what this lowest dimension is, other than with trial and error.

2.3.3 Structure-preservation in random projections

Random projections are a method of dimensionality reduction, so it is important to get some idea of what route projections take and how this falls under the general framework of dimensionality reduction. As mentioned in §2.1, essentially, with dimensionality reduction, we are asking whether a data-set *needs* all its dimensions, or whether some dimensions are superfluous. Projections are based on the Johnson-Lindenstrauss lemma, which deals with *Euclidean-distance preserving* embeddings. Therefore, we can apply projections when the structure of our

data can be represented using the distance between points. Note that this means that they are certainly not applicable to every problem where we need to reduce dimensions, simply because for other definitions of structure, they might give terrible results.

2.3.4 How can projections work?

In §2.1.2, we saw that some projections onto lines are better than others. Random projections involve projecting onto hyper-planes, and so it makes sense that we try and find similarly "good" projections. In PCA, the aim is to find such axes that minimize the variance of points along them. It is no wonder then that PCA is able to produce the best approximation for a projection onto k-dimensions.

By contrast, random projections do not look at the original data at all. Instead, a *random* subspace is chosen for projection. It is fair to ask how random projections manage to work at all! That is, how is it that we pick, from the multitude of "bad" projections that must be out there, a "good" projection by just working at random?

It is not easy to give an intuitive explanation of why this is so, but there are at least two aspects to the matter. First, one might use the argument that while random projections do not take advantage of "good" data-sets, they are not affected by "bad" data-sets - if we take these rough statements and consider all data-sets, we should be able break even. The fact that we are able to do more than this is surprising, however. A simple answer might be that the randomness means that we have to be very unlucky for our subspace to be bad for the data - it is far more likely that the subspace is good, and in fact, it turns out that the randomness means it is *very* likely that the subspace is very good!

To understand this, there is a theorem that says that as we go into higher-dimensions, the number of *nearly*-orthogonal vectors increases (Hecht-Nielsen, 1994). What this means is that the number of sets of vectors that are *nearly* a set of coordinates increases. Therefore, once we move into high-dimensions, the chance of us picking a "good" projection increases. In some sense, there become fewer and fewer "bad" projections in comparison to "good" projections, and so we have a good chance of preserving structure with an arbitrary projection. If however we chose to project our data onto one dimension, we are bound to get in trouble.

Also, as we get to higher dimensions, distances become meaningless (Hinneburg *et al.*, 2000). The reason for this is that there are so many dimensions that, unless we have truly large volumes of points, all distances are roughly the same. Therefore, as we increase our

original dimension, we have to do something truly abominable if we destroy these distances - the chance of doing this is quite small.

In some sense, this means that the "curse of dimensionality" is actually beneficial for us[2] - the high-dimensional points are essentially "lost in space", and so it becomes very easy to try and separate the points using a hyper-plane. The Johnson-Lindenstrauss lemma seems to use the "curse" to its advantage!

2.4 Projections and PCA

At one very obvious level, PCA is different to random projections in that it is explicitly concerned with using the input data. As noted in §2.2, the goal of PCA is to capture the directions of maximal variance, and project the data onto these directions. Clearly, finding these directions means that we have to make some use of the original data, and indeed, it seems almost impossible that we *don't* try to study the input data.

The optimality of PCA's reduction may seem like a compelling reason to choose it as a dimensionality reduction procedure, and indeed it is the reason why the method has been so popular. However, it is important to realize that PCA's accuracy comes at a significant cost - it has a runtime that is $O(nd^2)$, where n is the number of points and d the number of dimensions. This quadratic dependence on the number of original dimensions means that it quickly becomes infeasible for problems with very large number of dimensions. This is of course ironic - the whole point of dimensionality reduction is to get around the curse of dimensionality, but here the method of dimensionality reduction is itself hampered by the number of dimensions! In fact, for large data sets, it has been seen to be quite infeasible to apply in practise (Bingham and Mannila, 2001). In contrast, random projections have a much better runtime that is only linear in the number of dimensions[3], and so are more appealing in this regard, so the question becomes whether the errors they induce are comparable to that of PCA.

Also, perhaps surprisingly, it has been shown that random projections give errors that are very favourable for many applications, and are quite commensurate with the error from PCA (Bingham and Mannila, 2001; Lin and Gunopulos, 2003).

It should be noted that in the process of performing PCA, we do not respect local distances. This means that the distance between any pair of points may be arbitrarily distorted - PCA's

[2] Credit for this insight goes to Dr. Viglas

[3] The runtime is $O(ndk)$, where k is the reduced dimension

focus is always on the "big picture", of whether the overall distortion is minimized. Therefore, there are no guarantees on whether the distance between a pair of points in the original space will be anywhere near the distance in the new, projected space. There are problems where this is not particularly desirable, as it means that we are not able to say anything at all about the distance between a given pair of points in the reduced space[4].

In contrast, random projections work by preserving *all* pairwise distances with a high probability. This property of projections means that it can be used as a reliable estimator of distances in the original space - since computing these quantities in the original space takes $\Theta(d)$ time, where d is the original dimension, if we can reduce them to simply $\Theta(k)$ time, where $k \ll d$, that is quite a significant reduction. With PCA, we cannot get such guarantees.

Indeed, projections also offer guarantees on the lowest dimension we need to reduce to in order to get a certain amount of error, while in contrast, PCA does not offer us any such theory.

So, we can list the following differences between PCA and random projections.

(1) **Accuracy**: By design, PCA offers the best possible solution for any linear technique of dimensionality reduction[5]. However, random projections give errors that are typically comparable with that of PCA.

(2) **Runtime**: As mentioned earlier, PCA has a runtime that is quadratic in the number of original dimensions, which means that when this gets somewhat large, it simply takes too long to run. In contrast, random projections involve only a matrix multiplication, and so take time that is linear in the number of original dimensions.

(3) **Generality**: PCA is a technique that has seen successful applications to a wide range of problems. However, it has not been feasible to apply it with problems with truly high dimensionality, due to the cost of performing the reduction. Random projections have surprisingly been applied to an equally large range of problems, and have the advantage that they are able to operate quickly even when the number of dimensions that we are working with are very large.

(4) **Distance guarantees**: Random projections give strong guarantees on the preservation of pairwise distance, but do not say anything about the mean-square error of

[4]For example, take the problem of finding the approximate diameter of a high-dimensional point-set. If the distance between a pair of points can be arbitrarily distorted, we cannot get a nice relationship between the true diametric-pair in the original space, and the diametric-pair in the reduced space

[5]So there are certainly non-linear methods that may produce a better answer. However, these typically have an even worse runtime than PCA, and so in terms of practical applications, they are usually only useful when the original number of dimensions is relatively small

the projection. In contrast, PCA minimizes this mean-square error, but may distort pairwise distances arbitrarily, if it helps minimize the overall distortion. This means that depending on the application, either technique might not find itself particularly suited to the problem. For instance, consider the fact that projections preserve pairwise distance. It is certainly true that this is desirable in some problems (for instance, approximate-nearest neighbour problems). However, it may be that for certain problems, the Euclidean distance is fairly meaningless (for instance, problems involving a binary matrix). Therefore, we have to consider all options first.

(5) **Lowest dimension guarantees**: There is an interesting theory for random projections that lets us get fairly accurate bounds on the lowest dimension we can reduce to in order to achieve a maximal error of ϵ. In contrast, there are no such results for PCA, which means that some experimentation is required if we are to find the "lowest" dimension we can project down to, while not incurring too much error.

2.5 On the projection matrix

We mentioned that the matrix R had to be "suitable". What exactly does this mean? Clearly, the choice of projection matrix is crucial in determining if the random projections have their distance-preserving properties. Indeed, the term "random" in random projections should not be taken to mean that we are projecting onto some completely arbitrary subspace - it is better to think of the situation as having some underlying structure, but within that structure very loose restrictions.

Consider that our original points come from $\mathbb{R}^n$. This has a *basis* made up of orthogonal vectors - what this means is that the various components of the vectors can be expressed as the sum of n scaled vectors, each of which are perpendicular to the other $(n-1)$ vectors. Therefore, it is best if the subspace we choose is also orthogonal.

Early projection matrices tried to generate such subspaces. Unfortunately, it is not particularly easy to generate an orthogonal subspace. However, as mentioned earlier, it turns out that in high-dimensional spaces, the number of *nearly* orthogonal directions increases (Hecht-Nielsen, 1994), and as a result, it becomes easier to find subspaces that are "good enough".

We look at some important projection matrices that use this idea.

2.5.1 Classical normal matrix

Initially, random projections were done with a normal matrix, where each entry r_{ij} was an independent, identically distributed $\mathcal{N}(0,1)$ variable. The advantage of this was spherical symmetry among the random vectors, which greatly simplified analysis over existing Johnson-Lindenstrauss style embeddings (Dasgupta and Gupta, 1999). Further, the generation of the projection matrix was simpler, conceptually and practically, than existing approaches.

The importance of this matrix was that it showed that even though the random subspace was not orthogonal, this spherical symmetry was enough to ensure that it came close to having this property anyway.

2.5.2 Achlioptas' matrix

Achlioptas provided the seminal matrix (Achlioptas, 2001)

$$r_{ij} = +\sqrt{3}\begin{cases} +1 & p = 1/6, \\ 0 & p = 2/3, \\ -1 & p = 1/6 \end{cases}$$

in addition to the matrix

$$r_{ij} = \begin{cases} +1 & p = 1/2 \\ -1 & p = 1/2 \end{cases}$$

Note that the classical normal matrix required the generation of dk normal variables, which is not a completely inexpensive operation, especially for large d (where projections are supposed to be used!). Another point is that it requires that there be code for Gaussian generation readily available on hand, which might not be the case in, say, database applications (which was the motivation for Achlioptas' work).

In contrast, Achlioptas' matrix is very easy to compute, requiring only a uniform random generator, but even more importantly, it is $\frac{2}{3}$rds sparse, meaning that essentially $\frac{2}{3}$rds of our work can be cut off. Achlioptas noted "...spherical symmetry, while making life extremely comfortable, is not essential. What is essential is concentration", referring to the powerful concentration bounds that were available even with this very simple matrix.

2.5.3 Li's matrix

Recently, Li et al. (Li *et al.*, 2006) saw how to generalize Achlioptas' result by providing the *very-sparse projection matrix*

$$r_{ij} = +\sqrt{s} \begin{cases} +1 & p = 1/2s, \\ 0 & p = 1 - 1/s, \\ -1 & p = 1/2s \end{cases}$$

Note that Achlioptas' matrices are the cases $s = 2, 3$. They proved that for $s = o(d)$, the guarantees were still satisfied, only asymptotically. In particular, they suggested a $\sqrt{d}$-fold speedup of random projections by setting $s = \sqrt{d}$ - this means that the higher the original dimension, the better the speedup! The cost is of course that the higher the dimension, the longer it takes for the necessary convergence to happen. We study this matrix in §9.3.

2.6 On the lowest reduced dimension

2.6.1 Reduced dimension bounds

The Johnson-Lindenstrauss lemma (Johnson and Lindenstrauss, 1984), as mentioned in §2.3, tells us that we only need to reduce to $k = O(\log n/\epsilon^2)$ dimensions in order to approximately preserve pairwise distances upto a factor of $(1 \pm \epsilon)$. This is a very useful theoretical upper-bound, but in practise, we might be interested in knowing what constant the O hides. That is, we would like to get some explicit formula for k.

There were a series of simplifications to the original proof of Johnson and Lindenstrauss, culminating in the work of (Dasgupta and Gupta, 1999), who showed that the Johnson-Lindenstrauss guarantees could be obtained when

$$k \geq \frac{4}{\epsilon^2/2 - \epsilon^3/3} \log n$$

Achlioptas made one of the most important contributions to this area, by providing the following bound for k, guaranteeing pair-wise distance preservation with probability at least $n^{-\beta}$:

$$k \geq k_0 := \frac{4 + 2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n$$

This is not a strict lower-bound, but simply says that

$k \geq k_0 \implies$ Pairwise distance is probably preserved i.e. with the Achlioptas guarantees

Of course, we want to know if these bounds are tight. We look at trying to improve this bound for k, under more restricted conditions, in Chapter 8.

2.6.2 Theoretical lower bound

The only known lower bound, to our knowledge, on the reduced dimension that a Johnson-Lindenstrauss style embedding must have was provided by Alon (Alon, 2003). Via an ingenious connection to a simplex and the rank of a perturbed identity matrix, he managed to deduce that

$$k = \Omega\left(\frac{\log n}{\epsilon^2 \log \frac{1}{\epsilon}}\right)$$

This says that the *generic* dimension bound for k, if we focus on all data-sets, cannot beat $\frac{\log n}{\epsilon^2 \log \frac{1}{\epsilon}}$.

2.7 Applications of projections

Random projections have been successfully applied in many different fields. Obviously there are many areas where it has served as a dimensionality reduction tool, but here, we look at applications that are perhaps not as obvious, so that one appreciates the true power of the method.

2.7.1 Algorithmic applications

Random projections have seen numerous algorithmic applications, and have been used extensively to try and get approximate solutions to various NP-hard problems. We offer the following example that serves as an introduction in (Vempala, 2004). The original idea for this algorithm is due to (Goemans and Williamson, 1995).

The max-cut problem in a graph involves finding a cut, or partition of the graph, so that we maximize the number of edges that go across the cut. An edge goes across the cut if its start and end node are not in the same partition of the graph.

Suppose we label each node i with an integer $x_i \in \{-1, 1\}$ to label which partition the node falls into. Then, the max-cut problem asks us to maximize

$$\frac{1}{4} \sum_{(i,j)\in E} (x_i - x_j)^2 : x_i, x_j \in \{-1, 1\}$$

The relaxation of this is

$$\max \frac{1}{4} \sum_{(i,j)\in E} ||\mathbf{x_i} - \mathbf{x_j}||^2 : \mathbf{x_i} \in \mathbb{R}^n, ||\mathbf{x_i}||^2 = 1$$

It might not seem like a particularly useful relaxation of the problem - after all, we now have to solve a problem for a high-dimensional vector, which might seem to be just as hard. However, it turns out that we have preserved enough of the original structure to make this problem have an easy solution. For, we may write

$$||\mathbf{x_i} - \mathbf{x_j}||^2 = ||\mathbf{x_i}||^2 + ||\mathbf{x_j}||^2 - 2\mathbf{x_i}.\mathbf{x_j}$$

Suppose we denote $x_{ij} = \mathbf{x_i}.\mathbf{x_j}$. Then, certainly $x_{ii} = ||\mathbf{x_i}||^2 = 1$, and so, the relaxation really says

$$\max \frac{1}{2} \sum_{(i,j)\in E} 1 - x_{ij} : x_{ii} = 1, x_{ij} \geq 0$$

We can represent the dot-products with a matrix $X = \{x_{ij}\}$. Then, since each of the elements is non-negative, we call X *positive semi-definite*. It turns out that this program can be solved in polynomial time (we do not give the details here).

To relate our solution back to the original problem, what we do is *randomized rounding*, by projecting the vector $\mathbf{x}$ onto one dimension, and the rounding to ± 1, whichever is closer. That is, we do a random projection onto one dimension, and round our answer to make it answer the original question. The problem therefore uses a special case of a random projection.

As for the accuracy of our solution, we have the following lemma.

LEMMA 1. *Let* $\mathbf{x_i}, \mathbf{x_j}$ *be any two vectors in* $\mathbb{R}^n$ *with unit length, with the angle between them* θ_{ij}. *Suppose these are projected onto a random line* $\mathbf{r}$, *which also has unit length. Then, the probability that the signs of* $\mathbf{r}.\mathbf{x_i}$ *and* $\mathbf{r}.\mathbf{x_j}$ *are different is*

$$\frac{\theta_{ij}}{\pi}$$

Further, the accuracy factor of the solution is at least 0.87856.

We offer an intuitive proof of this lemma for the case $n = 2$ in Appendix A.1; this can be easily generalized to higher dimensions (Goemans and Williamson, 1995).

2.7.2 Coding theory

Random projections have been linked to the study of *compressed sensing*, which deals with the encryption and decryption of signals. The basic idea is this - we are given a signal $\mathbf{x}$, and we

want to encode it into a lower dimensional space. We do this with the help of an *encoder*, $\mathbf{\Phi}$, that produces our encoded signal $\mathbf{y}$:

$$\mathbf{\Phi x} = \mathbf{y}$$

However, once we are done manipulating the encoded signal, we want to revert back to the original space. To do this, we use a *decoder* Δ, that produces an approximation to the original signal, $\overline{\mathbf{x}}$:

$$\overline{\mathbf{x}} = \Delta y$$

One problem in compressed sensing is figuring out what good choices of encoder/decoder pairs are. Since we typically don't deal with just one signal, we are concerned about the performance over *all* signals in a given space. So, what we want is something that minimizes the maximal possible error when we retrieve a signal by encoding and decoding (this is of course only one possible way to measure the 'goodness' of the solution, but it is one of the more popular ones).

It turns out that given a encoder, finding an appropriate decoder is not very difficult - in fact, it amounts to solving a linear program, which can be done in polynomial time. However, finding an encoder is not very easy. There is however an important property that good encoders are known to satisfy, called the *Restricted Isometry Property* (RIP), which says that the length of the encoded signal is fairly close to that of the original signal:

$$(1-\epsilon)||\mathbf{x}||^2 \leq ||\mathbf{\Phi x}||^2 \leq (1+\epsilon)||\mathbf{x}||^2$$

It can be seen that this bears a striking similarity to the Johnson-Lindenstrauss embedding theorem. In fact, it was shown recently (Baraniuk *et al.*, 2006) that there is a direct connection between the two, with a proof that if we use a random projection matrix for our encoder, then the probability that we fail the RIP property is exponentially small. So, random projections can be applied to help find effective encoders for signals!

2.8 Special functions

We make use of some special functions during our theoretical analysis, since these functions often come up when working with probabilities (which form an integral part of random projections). We have compiled the definitions of these functions here for reference, but they are mostly for completeness' sake. There is no requirement to understand the detailed properties

of these functions, in that they are never used to make any deep statements, and are more the "glue" to get to certain theoretical results.

2.8.1 The Gamma function

The Gamma function is denoted by $\Gamma(z)$ and is defined by

$$\Gamma(z) := \int_0^\infty e^{-t} t^{z-1} dt$$

This function can be seen as the extension of the factorial to complex numbers:

$$\Gamma(z+1) = z\Gamma(z), z \in \mathbb{Z}$$

2.8.2 The Kronecker-Delta function

The Kronecker-Delta function δ_{xy} is defined by

$$\delta_{xy} := \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

This is really nothing more than a convenient indicator function - it is bound to a variable x, and says if y equals x or not.

2.8.3 The Dirac-Delta function

The Dirac-delta function $\text{Diract}(t)$ is not a true function, but rather a function that "lives under integrals", so to speak. It is defined by

$$\int_{-\infty}^{\infty} \text{Dirac}(x) dx = 1$$

$$\text{Dirac}(x) = 0 \text{ for } x \neq 0$$

At $x = 0$, the function is considered by some to be $+\infty$, but this can be quite confusing (and in fact this definition is very loose[6]). For our purposes, we will just need to rely on the fact that

$$\int_{-\infty}^{\infty} f(x)\text{Dirac}(x) dx = f(0)$$

[6] In fact, the delta function can be thought of as a measure, but this is not really relevant for our purposes

2.9 Elementary statistics

Some of the work in this thesis is of a theoretical nature, and so there is the need use some elementary concepts in statistics. These are presented here briefly. For more details, see (Mitzenmacher and Upfal, 2005), which is especially suited to the level of statistics that we require.

2.9.1 Random variables and distribution functions

We call a real-variable X a *random variable* if it relates the outcome of some 'experiment' to specific values.

The *cumulative distribution function* (CDF) of a variable X is the function

$$F(x) = Pr(X \leq x)$$

The CDF allows us to make the following distinction. We call X a *discrete random variable* if its CDF over an interval is either constant or has a jump-discontinuity. We call X a *continuous random variable* if its CDF is a continuous function.

For example, a variable

$$X = \begin{cases} +1 & p = \frac{1}{2} \\ -1 & p = \frac{1}{2} \end{cases}$$

is discrete, because its CDF will have a discontinuity at $x = \pm 1$, and is constant otherwise.

This is a very theoretical definition, and we can just think of a discrete variable as "a variable that takes on a constant number of constant numeric values". In this case, we have a constant number (two) of constant numeric values (± 1), so the variable is discrete.

The *probability density function* (pdf) of a variable X is a non-negative function such that

$$Pr(a \leq X \leq b) = \int_a^b f(x)dx$$

This gives us the probability that the variable lies between a and b. Note that

$$\int_{-\infty}^{\infty} f(x)dx = 1$$

2.9.2 Expected value

The *expected value* of a random variable X is denoted by $E(X)$, and is defined by

$$E(X) := \sum_i iPr(X = i) \tag{2.2}$$

Here, we are summing over all i values that the random variable X can attain. This is the "mean" of the random variable. So, for example, the expected value of a variable X that is either $+1$ or -1 with equal probability is zero.

For a continuous variable, this generalizes to

$$E(X) = \int_{-\infty}^{\infty} xf(x)dx$$

where $f(x)$ is the probability density function of X.

In fact, we have that for any continuous function g,

$$E(g(X)) = \int_{-\infty}^{\infty} g(x)f(x)dx \tag{2.3}$$

The expected value has a linearity property:

$$E\left(\sum_i X_i\right) = \sum_i E(X_i) \tag{2.4}$$

provided $\sum_i E(|X_i|) < \infty$. Note that the X_i's do **not** have to be independent!

We also have

$$E(aX) = aE(X)$$

for any $a \in \mathbb{R}$.

2.9.3 Variance

The *variance* of a random variable X is denoted by $Var(X)$, and is defined by

$$Var(X) := E(X^2) - (E(X))^2 \tag{2.5}$$

The variance satisfies linearity in the case of **independent variables**:

$$Var\left(\sum_i X_i\right) = \sum_i Var(X_i) \tag{2.6}$$

provided the X_i's are independent.

We also have

$$Var(aX + b) = a^2 Var(X) \tag{2.7}$$

for any $a, b \in \mathbb{R}$

2.9.4 Moment-generating function

The *moment-generating function* of a random variable X is denoted by $M_X(t)$, and is defined by

$$M_X(t) := E(e^{tX})$$

By the Taylor-series expansion of e^{tX}, we have

$$M_X^n(0) = E(X^n) \tag{2.8}$$

The quantities $E(X^n)$ are called the *moments* of the variable X, and so we can think of $M_X(t)$ as capturing all the moments of the variable in a single function.

An important property of these functions is that, for independent variables X, Y,

$$M_{X+Y}(t) = M_X(t)M_Y t \tag{2.9}$$

2.9.5 Chernoff bounds

The Chernoff-style bound is a powerful bound that this based on the moment-generating function. Markov's inequality states that

$$Pr(X \geq a) \leq \frac{E(X)}{a}$$

We can apply this to the moment-generating function of X, as follows:

$$Pr(X \geq a) = Pr(tX \geq ta) \text{ for any } t > 0 \tag{2.10}$$

$$= Pr(e^{tX} \geq e^{ta}) \tag{2.11}$$

$$\leq \frac{E(e^{tX})}{e^{ta}} \tag{2.12}$$

$$= \frac{M_X(t)}{e^{ta}} \tag{2.13}$$

In particular, we can choose t so that the RHS is minimized (since we can choose any t we want), and this gives us the optimal bound.

2.9.6 Distributions

2.9.6.1 Normal distribution

The normal distribution (sometimes called the Gaussian distribution) is one of the most important statistical distributions, and the normal distribution with mean μ and variance σ^2 is denoted by $\mathcal{N}(\mu, \sigma^2)$.

The distribution satisfies linearity

$$X \sim \mathcal{N}(\mu_1, \sigma_1^2), Y \sim \mathcal{N}(\mu_2, \sigma_2^2) \implies X + Y \sim \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2) \tag{2.14}$$

and also has the property that

$$X \sim \mathcal{N}(\mu, \sigma^2) \implies aX \sim \mathcal{N}(\mu, a^2\sigma^2) \tag{2.15}$$

An important property of the normal distribution is that of *2-stability*:

$$\sum \alpha_i X_i \sim \sqrt{\sum \alpha_i^2} X \tag{2.16}$$

where $X_i, X \sim \mathcal{N}(0, 1)$. Note that this means

$$\sum \alpha_i \mathcal{N}(0, 1) \sim \mathcal{N}(0, \sum \alpha_i^2)$$

and so every normal variable can be decomposed as the sum of other (scaled) normal variables.

The pdf for a normal distribution with mean μ and variance σ^2 is

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2} \tag{2.17}$$

It has moment generating function

$$M_X(t) = \exp\left(\mu t + \frac{\sigma^2 t^2}{2}\right) \tag{2.18}$$

Now, note that for $T \sim \mathcal{N}(0, \sigma^2)$, we have

$$\begin{aligned} E(T^{2n}) &= \int_{-\infty}^{\infty} t^{2n} \frac{e^{-t^2/2\sigma^2}}{\sqrt{2\pi}\sigma} dt \text{ by Equations 2.3, 2.17} \\ &= 2\int_0^{\infty} t^{2n} \frac{e^{-t^2/2\sigma^2}}{\sqrt{2\pi}\sigma} dt \text{ by even-ness of the integrand} \end{aligned}$$

Use the substitution $u = \frac{t^2}{2\sigma^2}$ and we get $du = \frac{t}{\sigma^2} dt$, $t = \sqrt{2u}\sigma$, and so

$$\begin{aligned} E(T^{2n}) &= 2\int_0^{\infty} \frac{1}{\sqrt{2\pi}\sigma} 2^n \sigma^{2n} u^n e^{-u} \frac{\sigma^2}{\sqrt{2u}\sigma} du \\ &= \int_0^{\infty} \frac{2^n \sigma^{2n}}{\sqrt{\pi}} e^{-u} u^{n-\frac{1}{2}} du \\ &= \frac{2^n \sigma^{2n}}{\sqrt{\pi}} \Gamma\left(n + \frac{1}{2}\right) \end{aligned}$$

Certainly, we have

$$E(T^{2n+1}) = \int_{-\infty}^{\infty} t^{2n+1} \frac{e^{-t^2/2\sigma^2}}{\sqrt{2\pi}\sigma} dt = 0$$

since the integrand is odd.

Therefore, for a normal variable

$$E(T^n) = \begin{cases} 0 & n \text{ odd} \\ \frac{2^{n/2}\sigma^n}{\sqrt{\pi}} \Gamma\left(\frac{n+1}{2}\right) & n \text{ even} \end{cases} \tag{2.19}$$

Another important result is the duplication formula,

$$\Gamma(n)\Gamma\left(n + \frac{1}{2}\right) = 2^{1-2n}\sqrt{\pi}\Gamma(2n) \tag{2.20}$$

2.9.6.2 Chi-square distribution

The Chi-square distribution with parameter k, denoted by χ_k^2, is defined by the sum of k variables that are the squares of $\mathcal{N}(0,1)$:

$$\sum_{i=1}^{k} X_i^2 \sim \chi_k^2 \tag{2.21}$$

where $X_i \sim \mathcal{N}(0,1)$.

This distribution has mean and variance given by

$$E(\chi_k^2) = k \tag{2.22}$$

$$Var(\chi_k^2) = 2k \tag{2.23}$$

and the moment-generating function

$$E(e^{t\chi_k^2}) = \frac{1}{(1-2t)^{k/2}} \tag{2.24}$$

2.9.6.3 Bernoulli distribution

A variable X follows a Bernoulli distribution, denoted $\mathcal{B}_p$, if

$$X = \begin{cases} +1 & \text{probability } p \\ 0 & \text{probability } (1-p) \end{cases}$$

This has the moment-generating function

$$M_X(t) = (1 - p) + pe^t$$

CHAPTER 3

Preliminary properties of projections

In this chapter, we look at some preliminary properties dealing with random projections which will be used later on. Some of these facts are not novel (see for instance (Vempala, 2004)), but we compile them here for ease of reference (note that the *proofs* of these known results is our own, however). The theorems that are novel have been explicitly marked as such.

The basic contributions that are made in this chapter include:

- The proof of some new results relating to basic properties of projections
- A simple extension of the work of (Vempala, 2004) to get an explicit projection theorem relating to the case when the projection matrix has Gaussian entries

3.1 Achlioptas' matrix

Recall that Achlioptas' matrix, introduced in §2.5.2, has entries of the form

$$r_{ij} = +\sqrt{3}\begin{cases} +1 & p = 1/6 \\ 0 & p = 2/3 \\ -1 & p = 1/6 \end{cases}$$

Henceforth, we call this distribution the *Achlioptas distribution*. This satisfies the following obvious properties.

LEMMA 2. *For the above distribution,*

$$E(r_{ij}) = 0$$
$$Var(r_{ij}) = 1$$

PROOF. By definition of expectation (Equation 2.2), we have

$$\begin{aligned} E(r_{ij}) &= \sum_k kPr(r_{ij} = k) \\ &= -\sqrt{3}.\frac{1}{6} + 0.\frac{2}{3} + \sqrt{3}.\frac{1}{6} \\ &= 0 \end{aligned}$$

Also,

$$r_{ij}^2 = 3 \begin{cases} +1 & p = 1/3 \\ 0 & p = 2/3 \end{cases}$$

Therefore, by definition of variance (Equation 2.5),

$$\begin{aligned} Var(r_{ij}) &= E(r_{ij}^2) - (E(r_{ij}))^2 \\ &= E(r_{ij}^2) \\ &= 3.\frac{1}{3} \\ &= 1 \end{aligned}$$

□

This generalizes to the following useful result that Achlioptas mentions without proof in his paper.

LEMMA 3. *Given a random variable X following the Achlioptas distribution, for any $n \geq 0$, we have*

$$E(X^n) = \begin{cases} 1 & \textit{when } n = 0 \\ 0 & \textit{when } n \textit{ is odd} \\ (\sqrt{3})^{n-2} & \textit{when } n \textit{ is even and } \neq 0 \end{cases}$$

PROOF. We can enumerate the possible values for X^n:

$$X^n = \begin{cases} (\sqrt{3})^n & p = 1/6 \\ \delta_{n0} & p = 2/3 \\ (-\sqrt{3})^n & p = 1/6 \end{cases}$$

where δ_{n0} is the Kronecker delta function 2.8.2.

$$\delta_{n0} = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$$

So, we have that for $n \neq 0$, $E(X^n) = \frac{((\sqrt{3})^n + (-\sqrt{3})^n)}{6}$, and when $n = 0$, $E(X^0) = 1$. When n is even, this will be $\frac{2.\sqrt{3}^n}{6} = (\sqrt{3})^{n-2}$. When n is odd, this will be zero. □

We also have the following interesting property relating this distribution to the much simpler Bernoulli distribution, defined in §2.9.6.3. This result is, to our knowledge, novel.

LEMMA 4. *The Achlioptas distribution can be expressed as a translation and scaling of two independent Bernoulli trials.*

PROOF. Consider the form of the variable $B_p + B_{1-p}$ by enumerating the possible values. The reason we choose this particular variable will become obvious soon.

$$B_p + B_{1-p} = \begin{cases} 2 & \text{probability } p(1-p) \\ 1 & \text{probability } p^2 + (1-p)^2 \\ 0 & \text{probability } p(1-p) \end{cases}$$

Certainly then, we have the symmetric variable

$$B_p + B_{1-p} - 1 = \begin{cases} 1 & \text{probability } p(1-p) \\ 0 & \text{probability } p^2 + (1-p)^2 \\ -1 & \text{probability } p(1-p) \end{cases}$$

And so, we have

$$\sqrt{3}(B_p + B_{1-p} - 1) = \begin{cases} +\sqrt{3} & \text{probability } p(1-p) \\ 0 & \text{probability } p^2 + (1-p)^2 \\ -\sqrt{3} & \text{probability } p(1-p) \end{cases}$$

This is the Achlioptas distribution, except that we need to solve for

$$p(1-p) = \frac{1}{6}$$

This can easily be solved using the quadratic formula to get $p = \frac{3 \pm \sqrt{3}}{6}$.

Hence,

$$\text{Ach} = \sqrt{3}(B_{\frac{3+\sqrt{3}}{6}} + B_{\frac{3-\sqrt{3}}{6}} - 1)$$

□

This is an interesting result, and it gives us an simple way to generate the Achlioptas distribution in practise. Notice that the scaling and translation are $O(1)$ operations, so we really only need to general two independent Bernoulli variables to get the Achlioptas distribution.

3.2 Basic projection lemmas

Let us suppose we have the input matrix

$$A = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1D} \\ u_{21} & u_{22} & \dots & u_{2D} \\ \vdots & & & \\ u_{n1} & u_{n2} & \dots & u_{nD} \end{bmatrix} = \begin{bmatrix} \mathbf{u_1} \\ \mathbf{u_2} \\ \vdots \\ \mathbf{u_n} \end{bmatrix}$$

and the random projection matrix

$$R = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1k} \\ r_{21} & r_{22} & \dots & r_{2k} \\ \vdots & & & \\ r_{D1} & r_{D2} & \dots & r_{Dk} \end{bmatrix} = \begin{bmatrix} \mathbf{r_1} & \mathbf{r_2} & \dots & \mathbf{r_k} \end{bmatrix}$$

Then, the projection of A is, by definition,

$$E = \frac{1}{\sqrt{k}} A.R = \frac{1}{\sqrt{k}} \begin{bmatrix} \mathbf{u_1}.\mathbf{r_1^T} & \mathbf{u_1}.\mathbf{r_2^T} & \dots & \mathbf{u_1}.\mathbf{r_k^T} \\ \mathbf{u_2}.\mathbf{r_1^T} & \mathbf{u_2}.\mathbf{r_2^T} & \dots & \mathbf{u_2}.\mathbf{r_k^T} \\ \vdots & & & \\ \mathbf{u_n}.\mathbf{r_1^T} & \mathbf{u_n}.\mathbf{r_2^T} & \dots & \mathbf{u_n}.\mathbf{r_k^T} \end{bmatrix} = \begin{bmatrix} \mathbf{v_1} \\ \mathbf{v_2} \\ \vdots \\ \mathbf{v_n} \end{bmatrix}$$

Therefore, the projection of the ith point may be represented as

$$\mathbf{v_i} = \frac{1}{\sqrt{k}} \begin{bmatrix} \mathbf{u_i}.\mathbf{r_1^T} & \mathbf{u_i}.\mathbf{r_2^T} & \dots & \mathbf{u_i}.\mathbf{r_k^T} \end{bmatrix}$$

With this simple representation, we can derive some important results relating to the lengths of the projected vectors, and the distances between the projected vectors.

3.2.1 Length preservation

LEMMA 5. *Let the entries of R have zero mean and unit variance. Then,*

$$E(||\mathbf{v_i}||^2) = ||\mathbf{u_i}||^2$$

PROOF. Without loss of generality, we consider the projection of the first point (i.e. we consider $i = 1$). We have

$$||\mathbf{v_1}||^2 = \frac{1}{k} \sum_i (\mathbf{u_1}.\mathbf{r_i^T})^2$$

$$= \frac{1}{k}\sum_i \alpha_i^2$$

where $\alpha_i := \mathbf{u_1}.\mathbf{r_i}^T$. Note that

$$\alpha_i = \sum_j u_{1j} r_{ji}$$

We can therefore write the expected value as

$$\begin{aligned} E(||\mathbf{v_1}||^2) &= E\left(\frac{1}{k}\sum_i \alpha_i^2\right) \\ &= \frac{1}{k}\sum_i E(\alpha_i^2) \text{ by linearity (Equation 2.4)} \\ &= \frac{1}{k}\sum_i Var(\alpha_i) + (E(\alpha_i))^2 \text{ by the definition of variance (Equation 2.5)} \end{aligned}$$

Now,

$$\begin{aligned} Var(\alpha_i) &= Var\left(\sum_j u_{1j} r_{ji}\right) \\ &= \sum_j Var(u_{1j} r_{ji}) \text{ by independence of } r_{ij}\text{'s, and Equation 2.6} \\ &= \sum_j u_{1j}^2 Var(r_{ji}) \text{ by Equation 2.7} \\ &= \sum_j u_{1j}^2 \text{ since } Var(r_{ij}) = 1 \\ &= ||\mathbf{u_1}||^2 \end{aligned}$$

Also, by linearity, and since $E(r_{ij}) = 0$,

$$E(\alpha_i) = \sum_j u_{1j} E(r_{ji}) = 0$$

Therefore,

$$\begin{aligned} E(||\mathbf{v_1}||^2) &= \frac{1}{k}\sum_i ||\mathbf{u_1}||^2 \\ &= ||\mathbf{u_1}||^2 \end{aligned}$$

□

This is a *extremely* remarkable result! We are saying that for *any* choice of projection matrix where the mean of the entries is zero and the variance is one, we will, on average, preserve

the lengths of the original vectors! The difference, however, is the *variance* in the lengths - this depends a lot on the precise choice of projection matrix. We can prove the following result on this variance when the projection matrix has Gaussian entries.

LEMMA 6. *If the entries of R come from a normal distribution, then*

$$Var(||\mathbf{v_i}||^2) = \frac{2}{k}||\mathbf{u_i}||^4$$

Equivalently,

$$Var\left(\frac{||\mathbf{v_i}||^2}{||\mathbf{u_i}||^2}\right) = \frac{2}{k}$$

PROOF. Again, w.l.o.g. consider the case $i = 1$. Since $r_{ij} \sim \mathcal{N}(0,1)$, by Equation 2.7, we have that

$$\alpha_i \sim \sqrt{\sum_i u_{1i}^2}\mathcal{N}(0,1) = ||\mathbf{u_1}||\mathcal{N}(0,1)$$

This means that, by 2.21,

$$\sum_{i=1}^{k} \alpha_i^2 \sim ||\mathbf{u_1}||^2\chi_k^2$$

So, by Lemma 5,

$$\begin{aligned} Var(||\mathbf{v_1}||^2) &= Var\left(\frac{1}{k}||\mathbf{u_1}||^2\chi_k^2\right) \\ &= \frac{||\mathbf{u_1}||^4}{k^2}Var(\chi_k^2) \text{ by Equation 2.21} \\ &= \frac{||\mathbf{u_1}||^4}{k^2}.2k \text{ by Equation 2.22} \\ &= \frac{2}{k}||\mathbf{u_1}||^4 \end{aligned}$$

□

It is also interesting to note that while the above say that

$$||\mathbf{v_i}||^2 \approx ||\mathbf{u_i}||^2$$

we also have the following statements regarding the individual components of the projected vector. These results are to our knowledge novel.

LEMMA 7. *Let the entries of R have zero mean and unit variance. Then,*

$$(\forall 1 \leq l \leq k) E(v_{il}) = 0$$

That is,

$$E(\mathbf{v_i}) = \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix}$$

PROOF.

$$\begin{aligned} E(v_{il}) &= E\left(\frac{1}{\sqrt{k}}\mathbf{u_i}.\mathbf{r_l}^T\right) \\ &= \frac{1}{\sqrt{k}} E\left(\sum_{n=1}^{d} u_{in} r_{nl}\right) \\ &= \frac{1}{\sqrt{k}} \sum_{n=1}^{d} u_{in} E(r_{nl}) \\ &= 0 \end{aligned}$$

□

Additionally, we have the following lemma that tells us that the square of each component can act, after scaling, as an estimator to the norm of the original vector.

LEMMA 8. *Let the entries of R have zero mean and unit variance. Then,*

$$(\forall 1 \leq l \leq k) E(v_{il}^2) = \frac{d}{k} ||\mathbf{u_i}||^2$$

PROOF. From the previous lemma, we have that

$$E(v_{il}^2) = E(v_{il})^2 + Var(v_{il}) = Var(v_{il})$$

So, we have

$$\begin{aligned} E(v_{il}^2) &= Var(v_{il}) \\ &= Var\left(\frac{1}{\sqrt{k}}\mathbf{u_i}.\mathbf{r_l}^T\right) \\ &= \frac{1}{k} Var\left(\sum_{n=1}^{d} u_{in} r_{nl}\right) \\ &= \frac{1}{k} \sum_{n=1}^{d} u_{in}^2 Var(r_{nl}) \text{ by independence} \end{aligned}$$

$$= \frac{1}{k} \sum_{n=1}^{d} u_{in}^2$$
$$= \frac{d}{k} ||\mathbf{u_i}||^2$$

□

For the case of a general distribution for R with zero mean and unit variance, we have the following result from (Li *et al.*, 2006), whose proof we omit.

LEMMA 9. *Let the entries of R have zero mean and unit variance. Then,*

$$Var(||\mathbf{v_i}||^2) = \frac{1}{k} \left(2||\mathbf{u_i}||^4 + (E(r_{ji}^4) - 3) \sum_j u_{ij}^4 \right)$$

PROOF. See (Li *et al.*, 2006). □

We can now use these results to get the following useful facts about distance preservation.

3.2.2 Distance preservation

The above results on length preservation easily generalize to distance preservation, using an appropriate substitution. Using the above results, we can verify the following theorem.

LEMMA 10. *Let the entries of R have zero mean and unit variance. Then,*

$$E(||\mathbf{v_i} - \mathbf{v_j}||^2) = ||\mathbf{u_i} - \mathbf{u_j}||^2$$

If the entries of R come from $\mathcal{N}(0,1)$, then,

$$Var(||\mathbf{v_i} - \mathbf{v_j}||^2) = \frac{2}{k} ||\mathbf{u_i} - \mathbf{u_j}||^4$$

PROOF. Note that we can simply use the above results to deduce this very similar lemma. Consider that

$$\mathbf{v_1} - \mathbf{v_2} = \frac{1}{\sqrt{k}} [(\mathbf{u_1} - \mathbf{u_2}).\mathbf{r_1^T} \quad (\mathbf{u_1} - \mathbf{u_2}).\mathbf{r_2^T} \quad \dots \quad (\mathbf{u_1} - \mathbf{u_2}).\mathbf{r_k^T}]$$
$$= \frac{1}{\sqrt{k}} [\mathbf{w}.\mathbf{r_1^T} \quad \mathbf{w}.\mathbf{r_2^T} \quad \dots \quad \mathbf{w}.\mathbf{r_k^T}]$$

where we set $\mathbf{w} := \mathbf{u_1} - \mathbf{u_2}$. So we can use the same steps as in Lemma 5 and 6 to deduce that

$$\begin{aligned} E(||\mathbf{v_1} - \mathbf{v_2}||^2) &= ||\mathbf{w}||^2 \\ &= ||\mathbf{u_1} - \mathbf{u_2}||^2 \end{aligned}$$

and when $r_{ij} \sim \mathcal{N}(0,1)$

$$\begin{aligned} Var(||\mathbf{v_1} - \mathbf{v_2}||^2) &= \frac{2}{k}||\mathbf{w}||^2 \\ &= \frac{2}{k}||\mathbf{u_1} - \mathbf{u_2}||^4 \end{aligned}$$

□

Notice the importance of Lemma 10 - it says that we expect the random projection to preserve distances in the original matrix. For the case where the projection matrix comes from a normal distribution, the variance of the distortion factor, $\frac{||\mathbf{v_i}-\mathbf{v_j}||^2}{||\mathbf{u_i}-\mathbf{u_j}||^2}$, is only $\frac{2}{k}$ - for k large enough, we have a good chance that the answer will be accurate!

3.2.3 Dot-product preservation

The dot-product of two vectors is

$$\mathbf{x}.\mathbf{y} = \sum_i x_i y_i$$

This quantity is studied in more detail in Chapter 5. As a preliminary note, we can show that the dot-product is, on average, preserved under a random projection.

LEMMA 11. *Let the entries of R have zero mean and unit variance. Then,*

$$E(\mathbf{v_i}.\mathbf{v_j}) = \mathbf{u_i}.\mathbf{u_j}$$

PROOF. We have that

$$\begin{aligned} E(\mathbf{v_i}.\mathbf{v_j}) &= E\left(\frac{1}{k}\sum_{l=1}^{k}(\mathbf{u_i}.\mathbf{r_l})(\mathbf{u_j}.\mathbf{r_l})\right) \\ &= \frac{1}{k}\sum_{l=1}^{k}E((\mathbf{u_i}.\mathbf{r_l})(\mathbf{u_j}.\mathbf{r_l})) \text{ by linearity} \end{aligned}$$

$$= \frac{1}{k} \sum_{l=1}^{k} E\left(\sum_{m=1}^{k} \sum_{n=1}^{k} u_{im} u_{jn} r_{lm} r_{ln} \right)$$

Now we use the fact that

$$E(r_{ij}) = 0$$

which means the only non-trivial case is when $n = m$, since here we deal with non-zero terms. The sum simplifies to

$$\begin{aligned} E(\mathbf{v_i}.\mathbf{v_j}) &= \frac{1}{k} \sum_{l=1}^{k} \sum_{m=1}^{k} E(u_{im} u_{jm}) E(r_{lm}^2) \text{ (*)} \\ &= \frac{1}{k} \sum_{l=1}^{k} k u_{im} u_{jm} \\ &= \sum_{l=1}^{k} u_{im} u_{jm} \\ &= \mathbf{u_i}.\mathbf{u_j} \end{aligned}$$

□

This theorem says that, on average at least, the dot-product is preserved under a random projection. However, it turns out that, unlike for distances, the variance in the dot-product can be quite large. We refer the reader to (Li *et al.*, 2006) for a treatment of the variance of the dot-product, which gives the following result.

LEMMA 12. *Let the entries of R have zero mean and unit variance. Then,*

$$Var(\mathbf{v_i}.\mathbf{v_j}) = \frac{1}{k} \left(||\mathbf{u_i}||^2 ||\mathbf{u_j}||^2 + (\mathbf{u_i}.\mathbf{u_j})^2 + (E(r_{ij}^4) - 3) \sum_m u_{im}^2 u_{jm}^2 \right)$$

PROOF. See (Li *et al.*, 2006). □

Note that in particular, for the case where the projection matrix is Gaussian or the Achlioptas matrix,

$$Var(\mathbf{v_i}.\mathbf{v_j}) = \frac{1}{k} \left(||\mathbf{u_i}||^2 ||\mathbf{u_j}||^2 + (\mathbf{u_i}.\mathbf{u_j})^2 \right) \tag{3.1}$$

We will study this variance, and the dot-product itself in more detail in Chapter 5.

3.3 A simple distance-preservation guarantee

We now show a simplified version of the key property of random projections, namely, that they manage to preserve pairwise distances. As with results for variance in this chapter, we look at the case where the matrix R has entries that are normally distributed.

The basic approach is similar to that of (Vempala, 2004), but we have extended the analysis to include an actual probabilistic bound that is similar to the one (Achlioptas, 2001).

THEOREM 4. *Let A be any $n \times d$ matrix whose rows are $\{\mathbf{u_i}\}_{i=1...n}$. Given any $\epsilon, \beta > 0$, suppose k is any number greater than*

$$k_0 = \frac{4 + 2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n$$

Let R be a $d \times k$ random matrix whose entries come from $\mathcal{N}(0, 1)$. Define

$$E = \frac{1}{\sqrt{k}} A.R$$

and let the rows of the $n \times k$ matrix E be $\{\mathbf{v_i}\}_{i=1...n}$. Then, with probability at least $1 - n^{-\beta}$, we have

$$(1 - \epsilon)||\mathbf{u_i} - \mathbf{u_j}||^2 \leq ||\mathbf{v_i} - \mathbf{v_j}||^2 \leq (1 + \epsilon)||\mathbf{u_i} - \mathbf{u_j}||^2$$

That is, with probability at most n^{β}, at least one pair of points get distorted by more than a factor of ϵ.

PROOF IDEA. The idea of this proof is to use Chernoff-bounds on the moment-generating function for the projected distance, and use this to get a highly concentrated bound that tells us the probability that the distance exceeds the original distance by more than a factor of ϵ.

To find this concentrated bound, we use the fact that the projection matrix is Gaussian, which makes it easy to compute the necessary moment-generating function. As a result of this tight bound, we can use the union bound to tell us when we can expect to preserve all pairwise distances. □

PROOF. Without loss of generality, consider the case $i = 1, j = 2$. We know the projection of the first point u_1 into $\mathbb{R}^k$ is

$$\mathbf{v_1} = \frac{1}{\sqrt{k}} \begin{bmatrix} \mathbf{u_1}.\mathbf{r_1^T} & \mathbf{u_1}.\mathbf{r_2^T} & \dots & \mathbf{u_1}.\mathbf{r_k^T} \end{bmatrix}$$

Certainly from §3.2, Lemma 10, we have

$$E(||\mathbf{v_1} - \mathbf{v_2}||^2) = ||\mathbf{u_1} - \mathbf{u_2}||^2$$

We can say that

$$Pr(\text{distortion}) = Pr(||\mathbf{v_1} - \mathbf{v_2}||^2 \geq (1+\epsilon)||\mathbf{u_1} - \mathbf{u_2}||^2 \vee ||\mathbf{v_1} - \mathbf{v_2}||^2 \leq (1-\epsilon)||\mathbf{u_1} - \mathbf{u_2}||^2)$$

since either of these cases means that we are more than ϵ away from the expected value. The method of proof is to use Chernoff-bounds, as outlined in Equation 2.10:

$$\begin{aligned} Pr(||\mathbf{v_1} - \mathbf{v_2}||^2 \geq (1+\epsilon)||\mathbf{u_1} - \mathbf{u_2}||^2) &= Pr(k||\mathbf{v_1} - \mathbf{v_2}||^2 \geq k(1+\epsilon)||\mathbf{u_1} - \mathbf{u_2}||^2) \\ &\leq \frac{E(e^{tk||\mathbf{v_1}-\mathbf{v_2}||^2})}{e^{tk(1+\epsilon)||\mathbf{u_1}-\mathbf{u_2}||^2}} \end{aligned}$$

This requires the use of the moment-generating function (see §2.9.4)

$$M_{k||\mathbf{v_1}-\mathbf{v_2}||^2}(t) := E(e^{tk||\mathbf{v_1}-\mathbf{v_2}||^2})$$

For simplicity, let us write

$$k||\mathbf{v_1} - \mathbf{v_2}||^2 = \sum_{i=1}^{k} \left(\sum_j (u_{1j} - u_{2j}) r_{ji} \right)^2 = \sum_{i=1}^{k} \alpha^2$$

where

$$\begin{aligned} \alpha &:= \sum_j (u_{1j} - u_{2j}) r_{ji} := \sum_j w_j r_{ji} \\ w_j &:= u_{1j} - u_{2j} \end{aligned}$$

Also without loss of generality, we can take $i = 1$, since the α_i's are independent. So, from now, we write

$$\alpha := \sum_j w_j r_{j1}$$

Then, by independence, we can write

$$\begin{aligned} M_{k||\mathbf{v_1}-\mathbf{v_2}||^2} &= M_{\sum \alpha^2} \\ &= \prod_i M_{\alpha^2} \text{ by Equation 2.9} \\ &= (M_{\alpha^2})^k \end{aligned}$$

Now, as demonstrated in Lemma 6, we can say that

$$\sum_{i=1}^{k} \alpha_i^2 \sim ||\mathbf{w}||^2 \chi_k^2$$

Therefore,

$$\begin{aligned} M_{k||\mathbf{v_1}-\mathbf{v_2}||^2}(t) &= M_{\sum a^2}(t) \\ &= E(e^{t||\mathbf{w}||^2\chi_k^2}) \\ &= M_{\chi_k^2}(||\mathbf{w}||^2 t) \\ &= \left(\frac{1}{\sqrt{1-2t||\mathbf{w}||^2}}\right)^k \text{ by Equation 2.24} \end{aligned}$$

So,

$$\begin{aligned} Pr(||\mathbf{v_1}-\mathbf{v_2}||^2 \geq (1+\epsilon)||\mathbf{u_1}-\mathbf{u_2}||^2) &\leq \frac{e^{-tk(1+\epsilon)||\mathbf{w}||^2}}{(1-2t||\mathbf{w}||^2)^{k/2}} \\ &= \left(\frac{e^{-2t(1+\epsilon)||\mathbf{w}||^2}}{1-2t||\mathbf{w}||^2}\right)^{k/2} \end{aligned}$$

We define

$$f(t) := \frac{e^{-2t(1+\epsilon)x}}{1-2tx}$$

Then, it can be shown that $f(t)$ has a minima at $t_0 := \frac{\epsilon}{2(1+\epsilon)x}$, where it takes the value $f(t_0) = (1+\epsilon)e^{-\epsilon}$.

So,

$$Pr(||\mathbf{v_1}-\mathbf{v_2}||^2 \geq (1+\epsilon)||\mathbf{u_1}-\mathbf{u_2}||^2) \leq \left((1+\epsilon)e^{-\epsilon}\right)^{k/2}$$

Using the Taylor series expansion, we get

$$(1+\epsilon)e^{-\epsilon} = 1 - \frac{\epsilon^2}{2} + \frac{\epsilon^3}{3} - O(\epsilon^4)$$

What this says is that

$$(1+\epsilon)e^{-\epsilon} < 1 - \frac{\epsilon^2}{2} + \frac{\epsilon^3}{3}$$

Therefore, if we let $x := -\epsilon^2/2 + \epsilon^3/3$, and consider that

$$e^x = 1 + x + O(x^2) > 1 + x$$

we see that we have

$$(1+\epsilon)e^{-\epsilon} < e^{-\epsilon^2/2+\epsilon^3/3}$$

This means that

$$Pr(||\mathbf{v_1}-\mathbf{v_2}||^2 \geq (1+\epsilon)||\mathbf{u_1}-\mathbf{u_2}||^2) \leq e^{k/2(-\epsilon^2/2+\epsilon^3/3)}$$

We follow the same approach to get the other inequality

$$Pr(||\mathbf{v_1} - \mathbf{v_2}||^2 \leq (1-\epsilon)||\mathbf{u_1} - \mathbf{u_2}||^2) \leq e^{k/2(-\epsilon^2/2+\epsilon^3/3)}$$

which tells us that

$$Pr(\text{distortion}) \leq 2e^{k/2(-\epsilon^2/2+\epsilon^3/3)}$$

This says that the probability that a given pair of points is distorted is $2e^{k/2(-\epsilon^2/2+\epsilon^3/3)}$. We are interested in a guarantee over *all* pairs of points. Consider that there are $\binom{n}{2} = \frac{n(n-1)}{2} < \frac{n^2}{2}$ such pairs. Therefore, we have

$$\begin{aligned} Pr(\text{Some pair distorted}) &= Pr(\text{Pair 1 distorted} \vee \text{Pair 2 distorted} \vee \ldots \vee \text{Pair } \binom{n}{2} \text{ distorted}) \\ &\leq \sum_i Pr(\text{Pair i distorted}) \\ &\leq \frac{n^2}{2} Pr(\text{Pair 1 distorted}) \end{aligned}$$

We are looking to make the probability that *any* pair is distorted very small - to be precise, we want

$$Pr(\text{Some pair distorted}) \leq \frac{1}{n^\beta}$$

where $\beta > 0$ is some constant. So, for large n and large β, the probability of a distortion becomes arbitrarily small. We can guarantee this, provided that we have

$$\frac{n^2}{2} 2e^{-k\epsilon^2/4+k\epsilon^3/6} \leq n^{-\beta}$$

We can easily invert this to deduce that this means

$$k \geq \frac{4+2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n$$

□

3.3.1 Remarks

The above result shows that random projections, at least with a Gaussian matrix, will preserve all pairwise distances with very high probability. It is remarkable that we can provide such a guarantee using such a simple procedure! This simplicty is one of the reasons why projections are a very appealing method of dimensionality reduction.

Achlioptas' insight was that the choice of a Gaussian matrix was "comfortable", in that it made the analysis simple, but not essential. He managed to show that we can get the same result as above even when one used his much simpler, discrete distribution. We give an alternate proof of his result in §8.2.

Also, note that the choice of bound we place on the probability of a distortion affects the lower bound on the reduced dimension, k. This means that there is potentially some scope for improving the bound, and hence sharpening the lower bound on k. We study this in more detail in Chapter 7.

Part 2

Data-streaming

CHAPTER 4

Projections and the streaming problem

The field of data-streams is an area of immense practical interest, as it captures a setting that is increasingly occuring in the real-world. This is because a lot of real-world data is not fixed, and is constantly changing with time. We can think of such data as a stream that is constantly updated, and in the general case, we can consider updates to be non-sequential (i.e. we do not necessarily update the pieces of the data in order).

Data-streams also suffer from the problem of high-dimensionality, and they are especially sensitive to a large number of dimensions since they are frequently updated. If, for instance, we want to do a perioidc analysis of a data-stream, then if the stream is high-dimensional, each analysis-step will prove to be a bottleneck for our entire process. It is natural to look to dimensionality reduction to try and ease the computational burden of the problem.

In this chapter, we look at applying random projections to a data-streaming problem, and provide a guarantee that projections can be used to keep what is known as a *sketch* of the stream, which is a high-quality approximation of it.

This chapter includes the following contributions:

- A proof that random projections can be used to keep an incremental sketch of a data-stream satisfying several desirable properties
- A proof that a well-known existing sketch is simply a special case of a random projection
- An argument as to why a general projection-based sketch is more desirable than the aforementioned existing sketch

4.1 Background

4.1.1 Streaming and analysis

A data stream, as the metaphor suggests, is the continuous flow of data generated at a source (or multiple sources) and transmitted to various destinations. Some examples of data streams

are the flow of data in the Internet, environmental data collection (through satellite and land-based stations), financial markets, telephony, etc. Since the number of applications that generate data streams are growing, there is a need for computational methods that can efficiently, and accurately, carry out sophisticated analysis on such sources of data.

Data-stream mining (DSM) and data-stream management systems (DSMS) are active areas of research which generalize and adapt existing techniques to handle streaming data. For example, in traditional database management systems (DBMS) the data is assumed to be fixed and persistent, and the set of queries on the database is assumed to be changing. But in contrast, the DSMS view is that the set of queries is fixed, and data is continuously flowing through the queries (Arasu *et al.*, 2003), (Chandrasekaran *et al.*, 2003). In DSM, the focus has been on efficient and incremental learning, i.e., a model is learnt on a batch of data, and as new data arrives, the model has to be updated to accommodate the new data without simply starting from scratch (as this would be too expensive). Further, the effect of "old" data has to be discounted from the model in order for it to be an accurate reflection of reality (Aggarwal *et al.*, 2005), (Gaber *et al.*, 2005).

However, most data-stream frameworks for data-mining assume that data arrives incrementally and sequentially (in-order), and develop solutions for this case. But in many situations, the data not only arrives incrementally, but also *asynchronously* - that is, at any point in time, we may be asked to update any arbitrary component or attribute of the existing data. The stream here cannot be mined with a one-pass algorithm, since the update of components is not (necessarily) sequential. Therefore, the challenge is to carry out our data mining task without having to instantiate the data vector - which may not be possible or computationally feasible. We explain this situation with the help of an example.

4.1.2 Prototype application

Consider a group of stock traders who trade on several equities listed in several stock markets. At a given moment in time, we associate a stock vector

$$\mathbf{s}(t) = (v_1, v_2, \ldots, v_n)$$

for each trader $\mathbf{s}$. Here v_i is the size of the holding for stock i, and n is number of all the stocks we consider (size of the universe of stocks). Now, the updates for each trader s arrive as tuples (j_s, u_s), where j_s is the index of the stock vector, and u_s is the value of the update. For example, if the stock vector represents the monetary value of stocks, then u_s can be a positive

or negative dollar value, depending upon whether the particular stock j_s was bought or sold. The actual update is performed as

$$v_i(t+1) = \begin{cases} v_i(t) + u_s & \text{if } i_s = j_s \\ v_i(t) & \text{otherwise} \end{cases}$$

Note that since u_s is negative, this a "turnstile-model"[1] of a data stream (Muthukrishnan, 2005).

Now, at a given point in time we would like to cluster the stock traders in real time, which will give us information about traders who are behaving similarly, or those who have moved from one cluster to another. The challenge is to do this efficiently, given that the stream is constantly being updated and that there is limited space. Theoretically, the stock vector can be very large, as large as the size of the union of all stocks that are traded in the world. So, if we want to use the standard k-means algorithm for clustering, then the complexity of each iteration is $O(nkd)$ where n, k and d are the number of stock traders, number of clusters and number of stocks respectively. As we will show in this chapter, using special data stream sketches, we can reduce the complexity to $O(nkD)$ where D is $O(\log n \frac{1}{\epsilon^2})$, and ϵ is the accuracy we want to achieve.

4.2 Problem definition

We define the problem by first considering a simple special case to illustrate the ideas clearly. Suppose we have an object that can be represented as an ordinary vector - that is, say it has a number of components with numerical values. However, let us also suppose that the object is dynamically changing, and so cannot simply be statically represented - each representation is only true for a particular time. Instead, at arbitrary points in time, we get information about the object that we use to update whatever information we already have.

Let us suppose that the information at each time-step is a simple update operation - it tells us to update a particular component of our vector by incrementing or decrementing it by some value. So, it might be a simple tuple

$$(i, c)$$

that tells us to update component i by adding onto it c (which might be negative).

[1] The motivation for the name is that if we imagine turnstiles in say a railway station, then they allow people to both enter and exit the station (we can "add" or "subtract" people from the station)

What is the initial value of our vector? If we have reliable information about what the initial state might be, then we can use that to initialize the vector. Otherwise, it can simply be set to be the zero vector. As such, it does not particularly affect the following analysis in this section.

For example, suppose we have a vector $\mathbf{a}$, and we get the sequence of updates

$$(1,2),(2,3),(1,3),(1,-1),(2,4),\ldots$$

Then, the state of the vector $\mathbf{a}$ at each of these time steps is

$$\begin{aligned}
\mathbf{a}(0) &= (0,0,0,\ldots)\\
\mathbf{a}(1) &= (2,0,0,\ldots)\\
\mathbf{a}(2) &= (2,3,0,\ldots)\\
\mathbf{a}(3) &= (5,3,0,\ldots)\\
\mathbf{a}(4) &= (4,3,0,\ldots)\\
\mathbf{a}(5) &= (4,7,0,\ldots)\\
&\vdots
\end{aligned}$$

In fact, we needn't constrain ourselves to a single object. Consider a collection of objects $\{\mathbf{u_1},\mathbf{u_2},\ldots,\mathbf{u_n}\}$, each with dimension d. A natural representation of this data is as an $n \times d$ matrix

$$A = \begin{bmatrix} \mathbf{u_1} \\ \mathbf{u_2} \\ \vdots \\ \mathbf{u_n} \end{bmatrix}$$

Suppose further that this matrix is updated asynchronously at various points in time, by updates of the form

$$(i,j,c)$$

which instructs us to update row i, column j with the value c (which may be negative). We can therefore consider A as holding n streams that may be updated at any time i.e. asynchronously, and denote the value of A at time t by $A(t)$. Therefore, $A(t)$ represents the state of each of our n streams at time t.

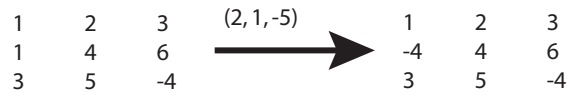


FIGURE 4.1: Example of the streaming process - we get a tuple that tells us to update a given row and column (in this case row 2, column 1) with a particular value (in this case, -5), so we have to decrement cell (1, 2) by -5

Now, in some problems, the value of d is very large (e.g. study of internet routers, stocks, etc.), in fact so large as to make instantiation of the matrix infeasible. However, we would still like to be able to extract some useful information about the data; for example, we might want to estimate the L_2 norm (Euclidean distance) of the individual rows/streams of the data, or perhaps the dot-product between streams.

To make such estimates, a *sketch* of the streams is usually kept. A sketch is a summary of a stream (or a series of streams) that has two desirable properties:

(1) It occupies very little space, and

(2) It allows us to give quick and accurate answers to queries on some quantity associated with the stream

A Euclidean distance-preserving sketch $\mathbf{c}$ for a stream $\mathbf{S}$, then, is some vector $\mathbf{c}(\mathbf{S})$ (or potentially a scalar) that occupies very little space and allows us to estimate $||\mathbf{S}||^2$ quickly and accurately.

What we want is some way of keeping an accurate, L_2-preserving sketch of a series of streams that occupies little space, and, ideally, can be updated efficiently. We show how a random projection can be used to construct such a sketch, and how the sketch can be updated incrementally as well.

4.3 Related work

In (Alon *et al.*, 1996), it was shown how to estimate frequency-moments F_k of a stream, where $F_k(\mathbf{a}) := \sum a_i^k$. In particular, for the case F_2, an efficient one-pass algorithm was proposed, using $O\left(\frac{1}{\epsilon^2}\right)$ space and giving a $1 \pm \epsilon$ approximation. However, this does not allow for asynchronous updates - if we think of the pass as a series of sequential updates to the components of the stream, then once we have updated the value of a particular component, we cannot go back and change it later.

An important sketch for preserving L_2 distance is the work of Indyk (Indyk, 2006), who proposed the use of a p-stable distribution in order to estimate the L_p norm for arbitrary $p \in (0, 2]$,

$$||\mathbf{x}||_p := \left(\sum_i |x_i|^p \right)^{1/p}$$

For $p = 2$, this requires the use of Gaussian random variables to maintain a sketch. The method used is essentially a random projection, though there are some minor differences regarding the way estimates are made; we look at the connection in more depth in §4.6. Surprisingly, while it has been noted that this sketch is inspired by random projection style embeddings, to our knowledge there has not been any work done using a sparse random matrix for the projection.

Since our work explicitly uses random projections as defined in the literature, we are able to access the theory of these projections. So, for instance, our bounds on the reduced dimension follow immediately from the Johnson-Lindenstrauss lemma (Johnson and Lindenstrauss, 1984), and our analysis does not have to deal with the accuracy of projection-based sketches. More importantly, we are able to justify the use of a sparse projection matrix with no loss in accuracy, and improved efficiency.

Of course, the L_p sketch offers us the ability to estimate L_p distances for general $p \in (0, 2]$, which makes it fairly versatile. An example of an interesting application of this sketch that our projection-based sketch cannot be used for is the estimation of the Hamming norm of streams (Cormode *et al.*, 2003).

4.4 Random projections and streams

We propose the use of random projections to help keep a sketch of a data-stream that allows us to answer distance queries with high accuracy. We provide theoretical analysis to prove the following claim, which is one of our main contributions in this chapter.

CLAIM 1. *Suppose we have an $n \times d$ matrix A that is updated at arbitrary points in time via tuples of the form*

$$(i, j, c)$$

which instruct us to update $A(i, j)$ with the (possibly negative) value c. Then, we can keep a sketch of the stream using random projections which satisfies the following properties:

- *The sketch occupies $\Theta(nk)$ space, where $k = O\left(\frac{\log n}{\epsilon^2}\right)$*

- *We can answer distance queries for the original data to up-to* $(1 \pm \epsilon)$ *accuracy*
- *We can incrementally update the sketch in* $\Theta(k)$ *time every time we get an update (using the sketch we have thus far)*
- *We can make our updates using* $\Theta(1)$ *space (the only significant space occupied is that of the sketch)*

In practise, we can use a $\frac{2}{3}$*rds sparse projection matrix that lets us use* $O\left(\frac{(4+2\beta)n\log n}{\epsilon^2/2-\epsilon^3/3}\right)$ *space, and provide* $(1 \pm \epsilon)$*-accurate answers to queries with probability at least* $1 - \frac{1}{n^\beta}$.

4.4.1 Setup

Suppose we have a data matrix $A(t) \in \mathbb{R}^{n \times d}$. Now, suppose that updates happen to this matrix in the following way. At time t, we get a tuple $d(t) = (i_t, j_t, c_t)$. When this happens,

$$a_{i_t j_t}(t) = a_{i_t j_t}(t-1) + c_t$$

That is, the entry in cell (i_t, j_t) gets incremented by the quantity c_t. We can express this as

$$A(t) = A(t-1) + C(t)$$

where $C(t) = \{C_{mn}\}$, such that $C_{mn} = c_t \delta_{i_t m} \delta_{j_t n}$ (where δ denotes the Kronecker delta function).

If d is large, it might be infeasible to instantiate the matrix A in memory. But we still want to keep the information associated with it, and we can try to do this with the help of a projection. Suppose we keep the projected matrix $E(t) \in \mathbb{R}^{n \times k}$, where k is a more manageable number. If we generate $E(t)$ via a random projection, then we have

$$E(t) = \frac{1}{\sqrt{k}} A(t) R$$

where R is, say, the Achlioptas matrix.

If we were able to do this, then clearly the theory of random projections guarantees that we will preserve pairwise distances in $A(t)$, and so, we could just use distances in $E(t)$ as an excellent approximation to those in $A(t)$.

But of course, $R \in \mathbb{R}^{d \times k}$, which is also comparable with A in terms of size. So, we can't really store a single random matrix R and use it for each of our projections. What can we do? An immediate idea is to try and do the generation on the fly. We show that such an approach is not likely to work, and show instead a different method to get around the space restriction of R.

4.4.2 On-the-fly random matrix generation

One option is to generate the matrix at every time step; of course, we don't generate the whole matrix due to space constraints, but only go one component at a time for the matrix multiplication. Still, for analysis' sake, let us assume we can remember the whole matrix for a time-step at least. Therefore, we have

$$E(t) = A(t)R(t)$$

Suppose that initially, $A(0) \equiv \mathbf{0}$ - we have no information about any component at all. It is clear that, based on the nature of the updates, we must have

$$A(t) = \sum_{n=1}^{t} C(n)$$

The projection of this matrix at time t will therefore be

$$E(t) = \sum_{n=1}^{t} C(n)R(t)$$

What happens the next time we get a tuple? Ideally, we would like to use the existing $E(t)$ matrix, and update it rather than doing a full recomputation. We can see that the expression for the new projected matrix is:

$$\begin{aligned}
E(t+1) &= \sum_{n=1}^{t+1} C(n)R(t+1) \\
&= C(t+1)R(t+1) + \sum_{n=1}^{t} C(n)R(t+1) \\
&= C(t+1)R(t+1) + E(t) + \sum_{n=1}^{t} C(n)(R(t+1) - R(t)) \\
&= E(t) + C(t+1)R(t+1) + \sum_{n=1}^{t} C(n)S(t+1)\ ,\ S(t) := R(t) - R(t-1)
\end{aligned}$$

4.4.3 Naïve approximation

Certainly one simple approximation for $E(t+1)$, based on the previous section, is

$$E(t+1) \approx E(t) + C(t+1)R(t+1)$$

This is easy to compute, but how bad is this as an approximation? The error at time $t+1$ will be

$$\delta(t+1) = \sum_{n=1}^{t} C(n)S(t+1)$$

To get a sense of the behaviour of this variable, suppose $X = r_{ij} - s_{ij}$, where r, s are from the Achlioptas distribution. Clearly, the quantity $S(t)$ must follow this distribution. By independence and linearity, we get

$$E(X) = 0, Var(X) = 2$$

Therefore,

$$E(\delta(t)_{ij}) = 0, Var(\delta(t)_{ij}) = 2\left(\sum C(n)\right)^2$$

So, while the expected error is zero, there is potentially quite a large variance, which only gets worse as t gets larger! This shows that the naïve approximation is too error-prone to be of much use, because we would expect the value of $C(n)$ to get arbitrarily large as we process more and more updates.

4.4.4 Repeatable generation

We have seen that we really do need to generate the same random matrix at every time-step, because otherwise the errors obtained get far too large. Before we suggest how to solve this problem, we observe that the matrix multiplication involved in the update step simplifies to the manipulation of a single vector alone.

4.4.4.1 Observation on multiplication by C

The fact that there are many zeros in the C matrix naturally suggests that we are saved a lot of work, and this is precisely the case. The situation we have is

$$C(t).R(t) = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{c_t} \\ \vdots \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{r_1}(t) & \mathbf{r_2}(t) & \dots & \mathbf{r_k}(t) \end{bmatrix}$$

$$= c_t \begin{bmatrix} 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ r_{j_t 1}(t) & r_{j_t 2}(t) & \dots & r_{j_t k}(t) \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

where

$$\mathbf{c_t} = [0 \quad 0 \quad \dots \quad c_t \quad 0 \quad \dots 0]$$

So, from the point of view of one update, we only need to compute k random numbers $r_{j_t i}(t)$. That is, every update only requires the generation of a single random row. From the point of view of this single update, it does not matter what the other elements of $R(t)$ are, since they are all cancelled by the zeros!

4.4.4.2 Pseudo-random generator

We can see that for a single update, we just need to generate $\Theta(k)$ random numbers - this is very much possible. The problem is that we want to generate the *same* random numbers each time we refer to the same row.

To get repeatable generation of the random rows, we use a pseudo-random number generator for generating the row values. Suppose that row i is generated using a seed $s(i)$ - then certainly if we fix $s(i)$ to be constant at all times, we can be guaranteed to get the same random row each time. However, since there are d rows, if we wanted to store d seeds we would need $d \log M$ bits of space, where M is the maximal seed value. In many cases, this might be comparable to simply storing the original data matrix.

But we can get around this in a very simple way, because the seed can be set to be the original column index, or the random row index, j_t (Indyk, 2006). With this seed, the pseudo-random number generator creates for us k values corresponding to the random row j_t. Since at times t_1 and t_2 where we refer to the same row, $j_{t_1} \equiv j_{t_2}$ trivially, we get the same random row.

With this method of generating R, the result of the earlier section reduces to the exact equation

$$E(t+1) = E(t) + C(t+1)R$$

This means that the update to the projected matrix is very simple - we just have to project the matrix $C(t+1)$ down (which is easy), and add this to our existing sketch.

Here, R only exists conceptually - to generate it one row at a time, we just use the above pseudo-random number generator with seed j_t. The above section on matrix multiplication shows that we really just need to generate a random row at each iteration, and further that this can be done on-the-fly, requiring $\Theta(1)$ space.

4.4.5 Complexity and comments

Recall that our sketch-update is

$$E(t+1) = E(t) + C(t+1)R$$

From the result of §4.4.4.1, we see that the matrix multiplication takes $\Theta(k)$ time. Hence, using random projections, we can keep a matrix $E(t)$ with space $\Theta(nk)$ that serves as an approximation to the original matrix $A(t)$, where updates take $\Theta(k)$ time (assuming that the matrix-row addition can be done in $\Theta(k)$ time as well - note that we don't have to update every row of $E(t)$, just row i_t).

Further, at any time t, the matrix $E(t)$ corresponds exactly with the matrix $A(t).R$, for some suitably chosen random-projection matrix R (that is, $E(t)$, which is incrementally updated, is the same as if we had done the expensive projection of the matrix $A(t)$ offline).

By projecting using a suitable k for the reduced dimension, we get the usual pairwise-distance preservation properties. The Johnson-Lindenstrauss lemma asserts that we only need space

$$nk = O\left(\frac{n \log n}{\epsilon^2}\right)$$

if we want to achieve at most ϵ distortion in our L_2 estimation.

In practise, we can use the Achlioptas projection matrix, which is known to be $\frac{2}{3}$rds sparse, to do our projection for us. This requires $O\left(\frac{(4+2\beta)n \log n}{\epsilon^2/2 - \epsilon^3/3}\right)$ space (although, as we will later see, the bound is known to be quite conservative).

Therefore, we have shown the claim presented at the start of the section, and we see that, theoretically, a random projection can be used to keep a sketch satisfying the properties mentioned in the claim.

4.4.6 Note on pseudo-random generators

It is very reasonable to pick up on the idea of pseudo-random generators in the previous section and ask whether this makes our choice of matrix random after all. The method proposed essentially means that, for the same implementation of a random number generator (e.g. MATLAB's `rand` function), if we ran the above on two different days, we would get the exact same random matrix! Recall that we are setting the seed to be the row index, and so there is really a minimal amount of randomness associated with the matrix.

To answer this dilemma, consider that the choice of matrix R is inherently prone to being good for some data and bad for some - and this leads us onto the third point, which is that the updates are themselves random. By Yao's minimax principle (Yao, 1977), this gives the same behaviour as if we had truly random matrices, but a fixed set of updates. Therefore, using the seeds as outlined above causes no problems.

4.4.7 Summary

To summarize, we use projections to keep a sketch of a series of data-streams as follows.

(1) We initialize the sketch with n rows and k columns, where $k = O(\log n/\epsilon^2)$, ϵ being the accuracy we want to achieve
(2) We get an update of the form (i, j, c) for one of our data-streams
(3) We use the value j as a seed for a pseudo-random generator
(4) One element at a time, we generate the random row $[c \times r_{j1} \quad \dots \quad c \times r_{jk}]$, where r_{ij} follows the Achlioptas distribution
(5) As we generate each element in the above row, we add this to the corresponding column in the ith row of our existing sketch (hence there is no extra space overhead)

4.5 Why the naïve approach fails

We already saw that if we consider random matrices that are updated at each time step, we get an error that is proportional to the square of the sum of all the updates. The above section on the projection of C highlights its sparsity, and the impact that has on the matrix multiplication - does that have any effect on the error term? Is it really far more lenient than our previous analysis suggets? Unfortunately, the answer is no. It is illustrative to see why this is the case - the randomness of our matrices does not mean that we can use such tricks and still have guarantees!

In fact, the sparsity of C just means that it is not *always* a problem to do a complete re-computation of the random matrix R at each time-step - but in general, there are 'bad' updates that will cause us to incur a heavy error. To be more precise, the problem happens when we have two updates to the same cell in A at two different times, or even two updates to the same column. Even if the update happens to the same row, but a different column, on-the-fly generation will suffice. But in this special case where the same cell is updated, we will have a problem if we are using a different random vector. Consider the simple example where we

have got three stream values:

$$(1, 1, c_1), (2, 2, c_2), (1, 1, c_3)$$

The third stream value asks us to update the same cell as the first, but this time with a different value. The precise indices chosen are not important (they just make the working easier), but note that the second stream value must refer to a different row. The state of A will be

$$A(3) = \begin{bmatrix} c_1 + c_3 & 0 & \dots & 0 \\ 0 & c_2 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \end{bmatrix}$$

Therefore,

$$||\mathbf{u_1} - \mathbf{u_2}||^2 = (c_1 + c_3)^2 + c_2^2$$

For the projections, we basically need three vectors of length k - suppose these are $\mathbf{r}, \mathbf{s}, \mathbf{q}$. We will see that we would need $\mathbf{r} \equiv \mathbf{q}$ for the projected length to match the original. The projected matrix will be

$$E(3) = \begin{bmatrix} c_1 r_1 + c_3 q_1 & c_1 r_2 + c_3 q_2 & \dots & c_1 r_k + c_3 q_k \\ c_2 s_1 & c_2 s_2 & \dots & c_2 s_k \\ \vdots & \vdots & \dots & \vdots \end{bmatrix}$$

The expected length is

$$\begin{aligned} E(||\mathbf{v_1} - \mathbf{v_2}||^2) &= E\left(\sum_{l=1}^{k} (c_1 r_l + c_3 q_l - c_2 s_l)^2\right) \\ &= \sum_{l=1}^{k} E\left[(c_1 r_l + c_3 q_l - c_2 s_l)^2\right] \\ &= \sum_{l=1}^{k} E\left(c_1^2 r_l^2 + c_3^2 q_l^2 + c_2^2 s_l^2 + 2c_1 c_3 r_l q_l - 2c_1 c_2 r_l s_l - 2c_2 c_3 q_l s_l\right) \\ &= c_1^2 + c_2^2 + c_3^2 \text{ assuming independence of } \mathbf{r}, \mathbf{s}, \mathbf{q} \end{aligned}$$

noting that we have taken the $\frac{1}{\sqrt{k}}$ scaling factor inside the random matrix for simplicity.

Had $\mathbf{r} \equiv \mathbf{q}$, independence would not apply and the result would be the length of the original vectors (as we might expect).

4.6 Comparison with L_p sketch

We compare the aforementioned "L_p sketch" with the projection-based sketch that we have outlined above.

4.6.1 The L_p sketch.

In (Indyk, 2006), it was proposed to use a p-stable distribution in order to estimate the L_p norm,

$$||\mathbf{x}||_p := \left(\sum_i |x_i|^p \right)^{1/p}$$

In particular, (Indyk, 2006) deals with the problem of a single vector **a** that gets updates of the form (i_t, a_t) at time t, indicating that index i_t should be updated by the value a_t. We can therefore represent the update vector as

$$C(t) = [0 \quad 0 \quad \dots \quad a_t \quad 0 \quad \dots \quad 0]$$

and thus the state of the true/underlying vector A at time t is just

$$\mathbf{a}(t) = \sum_{n=1}^{t} C(n)$$

We can show the following interesting result that hints at why the projection sketch is a more powerful technique.

CLAIM 2. *The L_2 sketch of Indyk is a special case of a random projection. In particular, it is a random projection with a normal projection matrix, and with no scaling by $\frac{1}{\sqrt{k}}$.*

PROOF. We intepret the sketch using analysis similar to the one employed for defining our projection sketch. As a result, the connection between this sketch and random projections will hopefully become obvious.

A sketch of this vector, $S(t)$, is maintained by using a matrix

$$X = [\mathbf{x_1} \quad \mathbf{x_2} \quad \dots \quad \mathbf{x_k}]$$

where $\mathbf{x_i}$ is a $d \times 1$ column vector, the entries x_{ij} come from a p-stable distribution Z:

$$\sum a_i Z_i \sim \left(\sum |a_i|^p\right)^{1/p} Z$$

and the value k is shown to be $O(\log \frac{1}{\delta}/\epsilon^2)$ for giving $1 \pm \epsilon$ accurate estimates with probability at least $1 - \delta$.

At every update, the sketch is updated with

$$S(t+1) = S(t) + a(t+1)\mathbf{r}(t+1)$$

where

$$\mathbf{r}(t+1) = [X_{i_{t+1}1} \quad X_{i_{t+1}2} \quad \dots \quad X_{i_{t+1}k}]$$

The L_p approximation is given by

$$\text{est}_{L_p}(||\mathbf{a}||^p) = \frac{\text{median}(s_1^2, s_2^2, \dots, s_k^2)}{\text{median}(|Z|^p)}$$

where Z is a variable from a p-stable distribution, and this is guaranteed to be a $1 \pm \epsilon$ approximation with probability at least $1 - \delta$. The key fact here is that we can get strong guarantees on the closeness of the median of the components of the sketch to the norm of the original stream/vector.

So, for $p = 2$, this means we can estimate the L_2 norm using a 2-stable distribution, such as a Gaussian distribution. As a result, we are essentially doing a projection with a matrix whose entries are i.i.d. Gaussian variables, and so this is essentially a classical random projection matrix. Therefore, the L_p sketch can be seen as a *special case of a random projection*.

One important difference exists, namely, the lack of the $\frac{1}{\sqrt{k}}$ scaling we normally associate with projections. This is because the estimate of the L_2 norm is given by

$$\text{est}_{L_2}(||\mathbf{a}||^2) = \frac{\text{median}(s_1^2, s_2^2, \dots, s_k^2)}{\text{median}(Z^2)}$$

where $\mathbf{s} = (s_1, s_2, \dots, s_k)$ denotes the sketch of the vector $\mathbf{a}$, and $Z \sim \mathcal{N}(0, 1)$. This differs from the projection estimate, which is the explicit computation of the norm:

$$\text{est}_{\text{proj}}(||\mathbf{a}||^2) = ||\mathbf{s}||^2$$

but gives the same guarantee, namely that with probability at least $1 - \delta$, we have

$$(1-\epsilon)\text{est}(||\mathbf{u_i}|^2) \leq ||\mathbf{u_i}||^2 \leq (1+\epsilon)\text{est}(||\mathbf{u_i}|^2)$$

□

So, we see that the L_2 sketch is nothing more than a projection, even though we get our estimates slightly differently. We now argue that using the general form of random projections is advantageous.

4.6.2 Advantage of sparse projections

Our insight is that the computation of 2-stable values is *not* essential to make accurate estimates of the L_2 norm of a stream, and that we can use the simpler Achlioptas matrix to do our projection. This has no impact on the accuracy of the sketch, and leads to greatly improved performance.

One potential problem with using 2-stable distributions in this streaming context is that we are required to regenerate the random rows every time there is an update - were it possible to store the random matrix once at the start, there would likely not be much difference between the two approaches. However, since every regeneration of a random row requires the generation of $\Theta(k)$ Gaussian variables, in streaming problems the effect of this cost can become noticeably large after long periods of time. Of course, depending on implementation, the precise difference between generating Gaussians and uniform random variables may differ, but with a high-volume of updates, even a small difference can prove to be very costly.

Further, the update itself is simplified by the fact that the Achlioptas matrix is, on average, $\frac{2}{3}$rds sparse - this means that we essentially don't have to do $\frac{2}{3}$rds of the multiplication of the update value with our random row (whereas for the Gaussian case, we would have to multiply this with every element in our random row of size k).

As for the claim about accuracy, this follows simply by Achlioptas' theorem, which asserts that using his sparse projection matrix, we get results that are at least as good as using a Gaussian projection matrix.

Therefore, we are able verify the following important claim, which is the final contribution in this chapter.

CLAIM 3. *One can use random projections with the $\frac{2}{3}$rds sparse Achlioptas matrix to keep a sketch of n data-streams that occupies $\Theta(nk)$ space, where $k = O(\log n/\epsilon^2)$. Further, this sketch can be updated quicker than the L_2 sketch of (Indyk, 2006), and produces results that are at least as accurate. This means that the projection sketch can achieve the same level of accuracy as the L_2 sketch using less space, and in (a constant factor) less time.*

4.7 Sketch and dot-products

While we know that projections give us guarantees on the preservation of L_2 distances, it is also a known fact that, on average, they preserve the dot-product (Lemma 11). As a result,

it is natural to ask whether we can use the projection sketch when we want to compute dot-products between objects in the original space. It turns out that we cannot do this with the same level of accuracy, but nonetheless we can get a certain guarantee on the error incurred by the dot-product, which can perhaps be used in special cases e.g. when it is known that the original objects have a fixed length. We study the preservation of the dot-product in Chapter 5.

CHAPTER 5

Dot-product distortion

The dot-product of two vectors, denoted by $\mathbf{u}.\mathbf{v}$, is defined by

$$\mathbf{u}.\mathbf{v} = \sum_i u_i v_i$$

Equivalently, we can write

$$\mathbf{u}.\mathbf{v} = ||\mathbf{u}||||\mathbf{v}|| \cos\theta_{uv}$$

where θ_{uv} is the "angle" between the high-dimensional points.

This is an interesting quantity that is used to calculate e.g. the cosine similarity between two vectors. It is therefore something that arises very naturally in streaming scenarios, and so it is of interest to know how it behaves under random projections, and in particular, whether we can give any guarantees on their preservation. In this chapter, we provide a new (high-probability) bound on the error incurred by the dot-product, and discuss how it relates to existing work.

The contributions made in this chapter include:

- A novel analysis of why the dot-product is not well-behaved in general under random projections, and an experimental illustration of this
- A new guarantee on sums of vectors being preserved under random projections, and an experimental verification of this
- A novel upper-bound on the error incurred by the dot-product using the above guarantee, and experimental verification of this fact

5.1 Why the dot-product is not well-behaved

In §3.2, we saw that we have

$$E(\mathbf{v_1}.\mathbf{v_2}) = \mathbf{u_1}.\mathbf{u_2}$$

So certainly the dot-product in the new space is an estimator of the dot-product in the original space, but we would like to know if it is a good one, i.e. whether we have any useful guarantees. It is known that for the Achlioptas projection matrix (see 3.1),

$$Var(\mathbf{v_1}.\mathbf{v_2}) = \frac{||\mathbf{u_1}||^2||\mathbf{u_2}||^2 + (\mathbf{u_1}.\mathbf{u_2})^2}{k}$$

(Li *et al.*, 2006) notes that one problem with the straightforward estimator is that there are no practical tail bounds that can be derived from them using Chernoff bounds, unlike for distances. In fact, the paper shows that one can use a different dot-product estimator that gives smaller variance. Unfortunately, it requires access to $||\mathbf{u_1}||^2$, $||\mathbf{u_2}||^2$, and is defined by:

$$\text{est}(\mathbf{u_1}.\mathbf{u_2}) := \frac{1}{2}(||\mathbf{u_1}||^2 + ||\mathbf{u_2}||^2 - 2||\mathbf{v_1} - \mathbf{v_2}||^2)$$

This is called a *margin-based* estimator, since it uses the so-called *margins*, $||\mathbf{u_1}||^2, ||\mathbf{u_2}||^2$. It makes intuitive sense that we should be able to get better results from this estimator, as we are using more information than is provided in the original space. In applications where these margins are known, the above is a natural choice for estimating the dot-product. However, there will still be scenarios where the quantities are not known - the data-streaming problem is certainly one such example. In such cases, the straightforward estimator is the best we can do, and therefore, it is of interest to know what results we can derive related to it.

The eager reader might notice that the margin-estimator uses $||\mathbf{u_1}||^2, ||\mathbf{u_2}||^2$, and ask quite reasonably why we cannot use the estimates to this, $||\mathbf{v_1}||^2, ||\mathbf{v_2}||^2$, instead, and get similarly tight variance. In fact, it turns out that this is precisely what the simple estimator does! The reason is that we can rewrite $\mathbf{v_1}.\mathbf{v_2}$ as

$$\mathbf{v_1}.\mathbf{v_2} = \frac{||\mathbf{v_1} + \mathbf{v_2}||^2 - ||\mathbf{v_1} - \mathbf{v_2}||^2}{4}$$

The margin-estimator, therefore, replaces the imprecise margins with the exact ones.

We will focus on those cases where we use the simple estimator to derive dot-product estimates. We can explore why the dot-product is not particularly well-behaved (using this estimate) under a random projection using the formula for the variance.

5.1.1 Dot-product variance

Using the fact that

$$\mathbf{u_1}.\mathbf{u_2} = ||\mathbf{u_1}||||\mathbf{u_2}|| \cos\theta$$

we can certainly say that

$$\begin{aligned} Var(\mathbf{v_1}.\mathbf{v_2}) &= \frac{1}{k}(||\mathbf{u_1}||^2||\mathbf{u_2}||^2 + (\mathbf{u_1}.\mathbf{u_2})^2) \\ &= \frac{1}{k}(||\mathbf{u_1}||^2||\mathbf{u_2}||^2 + ||\mathbf{u_1}||^2||\mathbf{u_2}||^2\cos^2\theta) \\ &= \frac{||\mathbf{u_1}||^2||\mathbf{u_2}||^2(1+\cos^2\theta)}{k} \end{aligned}$$

$$\leq \frac{2}{k}||\mathbf{u_1}||^2||\mathbf{u_2}||^2$$

An alternate form for the variance can be deduced using the arithmetic mean - geometric mean (AM-GM) inequality, which for two positive real numbers says that

$$\frac{x+y}{2} \geq \sqrt{xy}$$

In particular, then,

$$\frac{x^4+y^4}{2} \geq x^2y^2$$

This means that

$$Var(\mathbf{v_1}.\mathbf{v_2}) \leq \frac{1}{k}(||\mathbf{u_1}||^4 + ||\mathbf{u_2}||^4)$$

So, the variance of the dot-product in the projected space is deeply dependent on the lengths of the vectors in the original space. We see that if the lengths of the vectors are very large, the variance can get quite large (since there is a quartic dependence on the vector length). This by itself does not necessarily mean that the dot-product is not well-behaved, but it turns out that the variance does not get any better when the original dot-product gets small (unlike for distances). We look at why this is below.

5.1.2 Unbounded dot-product ratio

Ideally, we would like to have a guarantee similar to the one for distances, telling us that, with high probability,

$$(1-\epsilon)\mathbf{u_1}.\mathbf{u_2} \leq \mathbf{v_1}.\mathbf{v_2} \leq (1+\epsilon)\mathbf{u_1}.\mathbf{u_2}$$

If this were true, the term $\frac{\mathbf{v_1}.\mathbf{v_2}}{|\mathbf{u_1}.\mathbf{u_2}|}$ would have bounded variance. We can calculate the variance:

$$\begin{aligned} Var\left(\frac{\mathbf{v_1}.\mathbf{v_2}}{\mathbf{u_1}.\mathbf{u_2}}\right) &= \frac{1}{(\mathbf{u_1}.\mathbf{u_2})^2} Var(\mathbf{v_1}.\mathbf{v_2}) \\ &= \frac{\frac{||\mathbf{u_1}||^2||\mathbf{u_2}||^2}{(\mathbf{u_1}.\mathbf{u_2})^2} + 1}{k} \\ &= \frac{\sec^2\theta + 1}{k} \text{ since } \mathbf{u_1}.\mathbf{u_2} = ||\mathbf{u_1}||||\mathbf{u_2}||\cos\theta \end{aligned}$$

This suggests that when $\theta \to \pi/2$, namely, when the original vectors become nearly orthogonal, then the variance of the ratio of the dot-products increases without bound. The best

case is when the vectors are simply scaled values of each other i.e. $\theta = 0$, in which case the variance is $\frac{1}{k}$.

We can interpret this as saying that the angle between vectors, θ, is fairly well-preserved when $\theta \approx 0$, but starts to get arbitrarily distorted when $\theta \approx \frac{\pi}{2}$. So, the dot-product behaves badly because as the dot-product in the original space tends to zero, the dot-product in the projected space starts to become arbitrarily large with respect to it (that is, it does not strongly tend to zero as well - there is still a large variance in the results).

To summarize, we have shown the following claim, which is to our knowledge novel.

CLAIM. *The straightforward estimator for dot-products is not well behaved because when the dot-products in the original space tend to zero, we do not observe a corresponding decrease of the value of the dot-product in the projected space.*

5.1.3 Experimental verification

We can verify that as the dot-product tends to zero, the error ratio $\left|\frac{\mathbf{v_i}.\mathbf{v_j} - \mathbf{u_i}.\mathbf{u_j}}{\mathbf{u_i}.\mathbf{u_j}}\right|$ starts to get arbitrarily large.

We ran experiments to observe how the original dot-product affects the error ratio, plotting the two against each other. Even for a simple 100×100 data-set where the maximal length of each vector is constrained to $\sqrt{100} = 10$, we can observe the theoretical prediction. A graph is presented in Figure 5.1. We can see from the graph that as the original dot-product tends to zero, the error ratio starts to increase towards a peak.

In fact, we get more pronounced results when we increase the number of points, n, as the graph smoothens out (since there are more dot-products to consider). Figure 5.2 shows the graph for the case $n = 1000$, where the peak near $\mathbf{u_1}.\mathbf{u_2} = 0$ becomes even clearer.

5.2 An extended projection guarantee

Random projections guarantee that, with high probability,

$$(1-\epsilon)||\mathbf{u_i} - \mathbf{u_j}||^2 \leq ||\mathbf{v_i} - \mathbf{v_j}||^2 \leq (1+\epsilon)||\mathbf{u_i} - \mathbf{u_j}||^2$$

It is interesting to ask, however, whether

$$(1-\epsilon)||\mathbf{u_i} + \mathbf{u_j}||^2 \leq ||\mathbf{v_i} + \mathbf{v_j}||^2 \leq (1+\epsilon)||\mathbf{u_i} + \mathbf{u_j}||^2$$

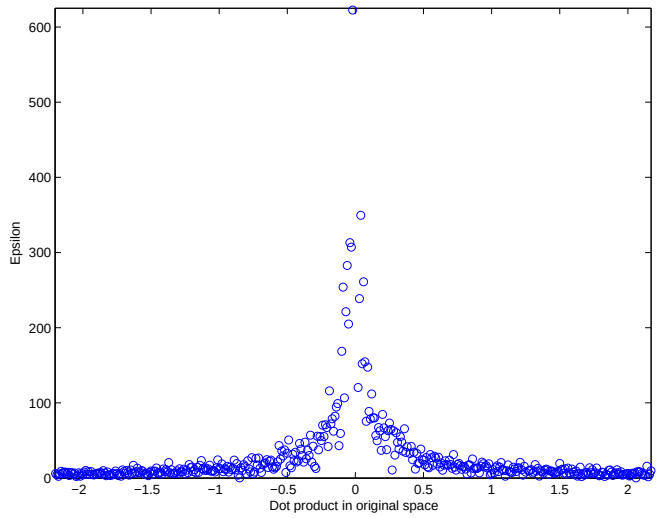


FIGURE 5.1: The original dot-product versus the error observed in the dot-product, $n = 100$

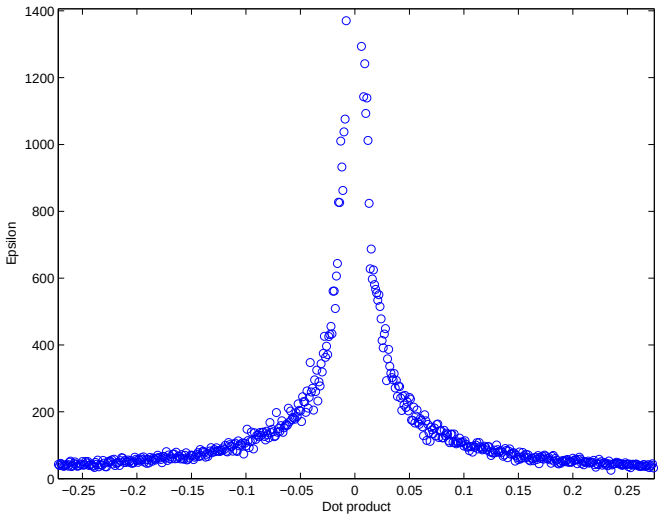


FIGURE 5.2: The original dot-product versus the error observed in the dot-product, $n = 1000$

This is obviously interesting at a theoretical level, since it says that both sums and differences of vectors in the original space get their lengths preserved. However, as we will see in the next section, this result is also interesting in the sense that it lets us get a bound on the error in the dot-product.

So, we present the following simple extension to the random projection guarantee, which serves as the crux of our new bound for the dot-product.

THEOREM 5. *Suppose the vectors* $\{\mathbf{u_i}\}_{i=1...n} \in \mathbb{R}^d$ *are projected to give us the vectors* $\{\mathbf{v_i}\}_{i=1...n} \in \mathbb{R}^k$, *where* $k = O(\log n/\epsilon^2)$, *such that, with high probability,*

$$(1-\epsilon)||\mathbf{u_i} - \mathbf{u_j}||^2 \leq ||\mathbf{v_i} - \mathbf{v_j}||^2 \leq (1+\epsilon)||\mathbf{u_i} - \mathbf{u_j}||^2$$

Then, we must also have, with high probability,

$$(1-\epsilon)||\mathbf{u_i} + \mathbf{u_j}||^2 \leq ||\mathbf{v_i} + \mathbf{v_j}||^2 \leq (1+\epsilon)||\mathbf{u_i} + \mathbf{u_j}||^2$$

PROOF IDEA. Noting that $\mathbf{u_i} + \mathbf{u_j} = \mathbf{u_i} - (-\mathbf{u_j})$, we construct a new matrix and apply the given projection guarantee to it. □

PROOF. [1] Let us consider the given vectors as a matrix $A \in \mathbb{R}^{n\times d}$. We are told that we can apply a projection that takes the rows $\mathbf{u_i}$ of A to a row $\mathbf{v_i} \in \mathbb{R}^k$, where $k = O(\log n/\epsilon^2)$, such that with high probability,

$$(1-\epsilon)||\mathbf{u_i} - \mathbf{u_j}||^2 \leq ||\mathbf{v_i} - \mathbf{v_j}||^2 \leq (1+\epsilon)||\mathbf{u_i} - \mathbf{u_j}||^2$$

Now, form the data-set B with $2n$ points, consisting of the points in A and their negatives: $B = \{\mathbf{u_i}, -\mathbf{u_i}\}$. Suppose we use the same projection matrix to do the projection of B - then, our projection guarantees tell us that, with high probability, we must have

$$(1-\epsilon)||\mathbf{u_i} + \mathbf{u_j}||^2 \leq ||\mathbf{v_i} + \mathbf{v_j}||^2 \leq (1+\epsilon)||\mathbf{u_i} + \mathbf{u_j}||^2 \tag{5.1}$$

by writing $\mathbf{u_i} + \mathbf{u_j} = \mathbf{u_i} - (-\mathbf{u_j})$. Note that $\log(2n) = O(\log n)$, meaning we are still reducing to an $O(\log n/\epsilon^2)$ subspace.

Why are we allowed to fix the choice of the projection matrix? This is because this matrix is inherently random when we choose it for A - therefore, the matrix is not fixed over all possible matrices A. Since the Achlioptas guarantees work for a randomly chosen projection matrix, they must in particular work for whatever matrix we have chosen for A.

Also, note that the independence, in terms of generation, of the random rows in B means that we don't really have to project the set B down to k dimensions - we can just project A, and provided k is large enough, the guarantee (Equation 5.1) must still hold anyway. What we are arguing here is that the projection of B is just the projection of A and $-A$ pasted together. Certainly the projection of $-A$ is the negative of the projection of A - this means that by just projecting A, we get the above guarantee.

[1] Credit goes to Anh Pham for providing the motivation for this proof

To make this concrete, we are saying that

$$E_B = \frac{1}{\sqrt{k}} BR = \begin{bmatrix} E_A \\ --- \\ -E_A \end{bmatrix}$$

which means that we are not doing anything special by projecting B - we can just use the projection for A itself to make this new projection. Therefore, we do not have to actually construct B, and the guarantees will hold anyway.

Therefore, we have, with high probability,

$$(1-\epsilon)||\mathbf{u_1} \pm \mathbf{u_2}||^2 \leq ||\mathbf{v_1} \pm \mathbf{v_2}||^2 \leq (1+\epsilon)||\mathbf{u_1} \pm \mathbf{u_2}||^2$$

□

5.2.1 Experimental verification

We experimentally verified that the above theorem does in fact hold. We took a set of random data with 100 points in 100 dimensions, and ran experiments with a large (10^6) number of iterations, where on each iteration the data was randomly projected to k dimensions. The dimension k was varied, and we observed the effect it had on the ratio of the sums. The results are presented in Table 5.1.

The results indicate that, as expected, the mean ratio of the sums of vectors is around 1 (Figure 5.3); that is, sums are on average preserved. Even for small values of k, the mean is very close to 1, and this only improves as k increases. Further, we see that the sums of vectors are preserved with a low variance (Figure 5.4) - in fact, we see that the variance behaves like $\frac{1}{k}$, which is as expected, since we basically have the same guarantee as Equation 10. Again, even for small k, the variance can be seen to be quite low. Therefore, the extended projection guarantee can be experimentally validated.

We can also see a histogram of the sum ratios for $k = 2$ (Figure 5.5), which shows that they follow a symmetric distribution that appears to be normal in shape. This is consistent with theory of distances of vectors, which are known to be approximately normal due to a central limit theorem style result (Li *et al.*, 2006).

We get similar results when $n = 1000$, presented in Table 5.2 and Figures 5.6 and 5.7. As expected, the results are slightly worse than that of the $n = 100$ case (since the reduced

k	Ratio of sum
2	1.0009 ± 0.0193
10	1.0004 ± 0.0040
20	0.9998 ± 0.0020
30	1.0002 ± 0.0013
40	0.9993 ± 0.0010
50	0.9994 ± 0.0008
60	1.0003 ± 0.0006
70	0.9998 ± 0.0006
80	0.9999 ± 0.0005
90	1.0001 ± 0.0004
100	1.0000 ± 0.0004

TABLE 5.1: Mean and variance for the ratio of sums, $n = 100$

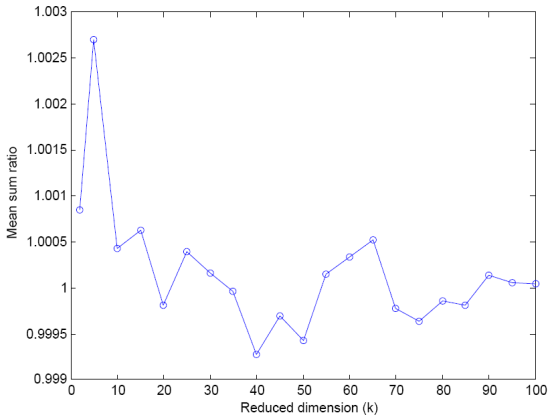


FIGURE 5.3: The mean of the sum of vectors is very close to 1 even for small k values, and it gets closer as we increase the reduced dimension

dimension k, theoretically, should increase accordingly with n). We still observe, however, that the results are as theoretically predicted even for low dimensions.

5.3 A new bound

We saw in §5.1 that the dot-product can have quite a large variance, which led us to conclude that it is not well behaved under a random projection. Nonetheless, it is of interest to know how precisely much error we can expect in practise (which is akin to asking what sort of distribution the error follows).

The extended projection guarantee allows to make an interesting theorem on the maximal error that is likely to be incurred. This is not as strong a guarantee as we have for distance, but

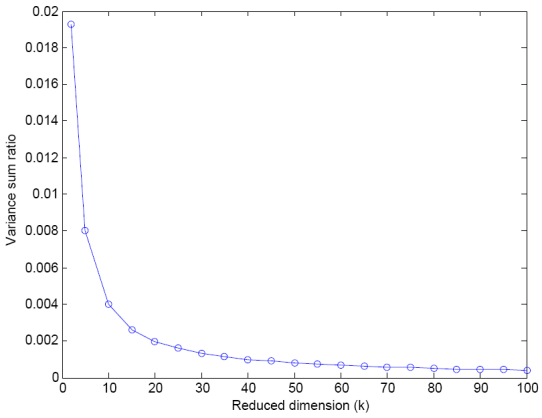


FIGURE 5.4: The variance of the sum of vectors behaves like $\frac{1}{k}$, and so gets very small as k gets large

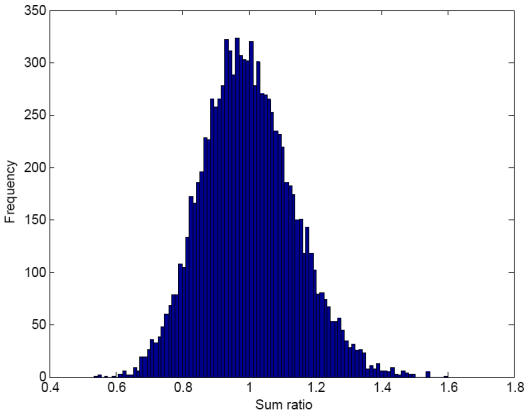


FIGURE 5.5: The distribution approaches that of a normal distribution if we let the data take on positive and negative values. The mean can be observed to be 1, and the variance is fairly small

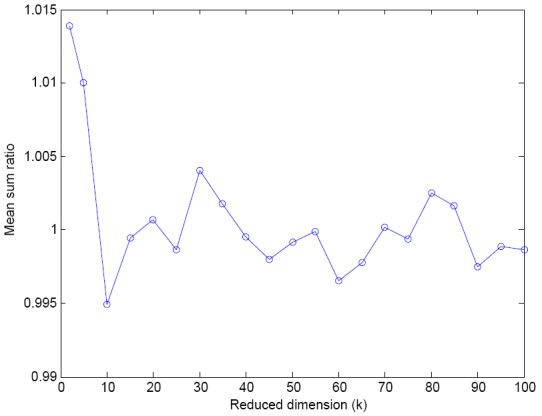


FIGURE 5.6: The mean of the sum of vectors is very close to 1 again when $n = 1000$

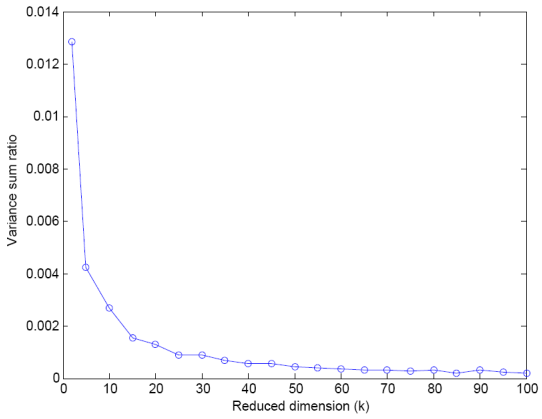


FIGURE 5.7: The variance of the sum of vectors behaves like $\frac{1}{k}$ when $n = 1000$

k	Ratio of sum
2	1.0139 ± 0.1132
10	0.9949 ± 0.0517
20	1.0007 ± 0.0358
30	1.0040 ± 0.0298
40	0.9995 ± 0.0235
50	0.9992 ± 0.0207
60	0.9966 ± 0.0186
70	1.0002 ± 0.0177
80	1.0025 ± 0.0171
90	0.9975 ± 0.0179
100	0.9987 ± 0.0135

TABLE 5.2: Mean and variance for the ratio of sums, $n = 1000$

it is nonetheless of interest, since it is provably tight. The theorem is a new result, and is given below.

THEOREM 6. *If the points* $\{\mathbf{u_i}\}_{i=1\ldots n}$ *are projected to* $\{\mathbf{v_i}\}_{i=1\ldots n}$*, where* $k = O(\log n/\epsilon^2)$*, then, with high probability,*

$$|\mathbf{v_i}.\mathbf{v_j} - \mathbf{u_i}.\mathbf{u_j}| \leq \frac{\epsilon}{2}(||\mathbf{u_i}||^2 + ||\mathbf{u_j}||^2)$$

PROOF IDEA. We use the extended projection guarantees and the fact that

$$\mathbf{x}.\mathbf{y} = \frac{||\mathbf{x}+\mathbf{y}||^2 - ||\mathbf{x}-\mathbf{y}||^2}{4}$$

This connects the dot-product to distances, and so our distance related guarantees can be molded into dot-product related guarantees. □

PROOF. The following fact that relates the dot-product to distances:

$$\mathbf{x}.\mathbf{y} = \frac{||\mathbf{x}+\mathbf{y}||^2 - ||\mathbf{x}-\mathbf{y}||^2}{4}$$

This result follows because

$$||\mathbf{x} \pm \mathbf{y}||^2 = (\mathbf{x} \pm \mathbf{y}).(\mathbf{x} \pm \mathbf{y}) = ||\mathbf{x}||^2 \mp 2\mathbf{x}.\mathbf{y} + ||\mathbf{y}||^2$$

Therefore, using the results from the previous section, we know that with high probability,

$$\begin{aligned}
(1-\epsilon)||\mathbf{u_i}+\mathbf{u_j}||^2 - (1+\epsilon)||\mathbf{u_i}-\mathbf{u_j}||^2 &\leq ||\mathbf{v_i}+\mathbf{v_j}||^2 - ||\mathbf{v_i}-\mathbf{v_j}||^2 \\
&\leq (1+\epsilon)||\mathbf{u_i}+\mathbf{u_j}||^2 - (1-\epsilon)||\mathbf{u_i}-\mathbf{u_j}||^2 \\
\implies (1-\epsilon)\frac{||\mathbf{u_i}+\mathbf{u_j}||^2}{4} - (1+\epsilon)\frac{||\mathbf{u_i}-\mathbf{u_j}||^2}{4} &\leq \frac{||\mathbf{v_i}+\mathbf{v_j}||^2}{4} - \frac{||\mathbf{v_i}-\mathbf{v_j}||^2}{4} \\
&\leq (1+\epsilon)\frac{||\mathbf{u_i}+\mathbf{u_j}||^2}{4} - (1-\epsilon)\frac{||\mathbf{u_i}-\mathbf{u_j}||^2}{4} \\
\implies \mathbf{u_i}.\mathbf{u_j} - \frac{\epsilon}{4}(||\mathbf{u_i}-\mathbf{u_j}||^2 + ||\mathbf{u_i}+\mathbf{u_j}||^2) \leq \mathbf{v_i}.\mathbf{v_j} &\leq \mathbf{u_i}.\mathbf{u_j} + \frac{\epsilon}{4}(||\mathbf{u_i}-\mathbf{u_j}||^2 + ||\mathbf{u_i}+\mathbf{u_j}||^2)
\end{aligned}$$

From here, we can say

$$\begin{aligned}
-\frac{\epsilon}{4}(||\mathbf{u_i}-\mathbf{u_j}||^2 + ||\mathbf{u_i}+\mathbf{u_j}||^2) \leq \mathbf{v_i}.\mathbf{v_j} - \mathbf{u_i}.\mathbf{u_j} &\leq \frac{\epsilon}{4}(||\mathbf{u_i}-\mathbf{u_j}||^2 + ||\mathbf{u_i}+\mathbf{u_j}||^2) \\
\implies -\frac{\epsilon}{2}(||\mathbf{u_i}||^2 + ||\mathbf{u_j}||^2) \leq \mathbf{v_i}.\mathbf{v_j} - \mathbf{u_i}.\mathbf{u_j} &\leq \frac{\epsilon}{2}(||\mathbf{u_i}||^2 + ||\mathbf{u_j}||^2)
\end{aligned}$$

So, we get

$$|\mathbf{v_i}.\mathbf{v_j} - \mathbf{u_i}.\mathbf{u_j}| \leq \frac{\epsilon}{2}(||\mathbf{u_i}||^2 + ||\mathbf{u_j}||^2)$$

with high probability. □

An immediate consequence is the following important corollary.

COROLLARY. *If the points* $\{\mathbf{u_i}\}_{i=1\ldots n}$ *are projected to* $\{\mathbf{v_i}\}_{i=1\ldots n}$, *where* $k = O(\log n/\epsilon^2)$, *then, with high probability,*

$$\left|\frac{\mathbf{v_i}.\mathbf{v_j}}{\mathbf{u_i}.\mathbf{u_j}} - 1\right| \leq \frac{\epsilon}{2|\mathbf{u_i}.\mathbf{u_j}|}(||\mathbf{u_i}||^2 + ||\mathbf{u_j}||^2)$$

This gives us a high-probability upper bound on the error ratio, which, although not as strong as the one for distances, is certainly of theoretical interest.

5.3.1 Tightness of bound

It is unlikely that one can find a tighter bound on the error incurred by the dot-product, simply because of the tightness of the distance results. Indeed, if we had

$$|\mathbf{v_1}.\mathbf{v_2} - \mathbf{u_1}.\mathbf{u_2}| \leq A\epsilon$$

for some $A > 0$, then we could rewrite this as

$$\mathbf{u_1}.\mathbf{u_2} - A\epsilon \leq \mathbf{v_1}.\mathbf{v_2} \leq \mathbf{u_1}.\mathbf{u_2} + A\epsilon$$

and as we did in the proof, write the dot-product in terms of distance. If we then claim that the bound of $A\epsilon$ is tight, using the fact that the distance bounds are tight, we would be forced to choose A to be the quantity we have given in the above theorem. Hence, by the tightness of the distance bounds, this dot-product bound must also be tight.

5.3.2 "Not well-behaved" revisited

As a quick note, we can see that when $\mathbf{u_1}.\mathbf{u_2} = 0$, and in particular when $\theta = \frac{\pi}{2}$ (that is, we are not taking the dot-product with a zero vector), we have that, with high probability

$$|\mathbf{v_i}.\mathbf{v_j}| \leq \frac{\epsilon}{2}(||\mathbf{u_i}||^2 + ||\mathbf{u_j}||^2)$$

The previous section asserts this bound is essentially tight, and so we see here why the dot-product is not well-behaved - when $\mathbf{u_1}.\mathbf{u_2} = 0$, we do not have a strong bound on $\mathbf{v_1}.\mathbf{v_2}$, and instead it is allowed to get quite large when the norms of the vectors are large. By contrast, for distances, we would see the projected distance correspondingly go to 0 as well.

5.3.3 Interpreting with reduced dimension

By the Johnson-Lindenstrauss lemma, the reduced dimension k must be $O(\log n/\epsilon^2)$. Equivalently, the error ϵ must be

$$\epsilon = O\left(\sqrt{\frac{\log n}{k}}\right)$$

Therefore, we are saying that

$$|\mathbf{v_1}.\mathbf{v_2} - \mathbf{u_1}.\mathbf{u_2}| \leq A\sqrt{\frac{\log n}{k}}$$

for some constant $A > 0$. This gives an explicit relationship with the reduced dimension, k, which is present in the variance formulae. This is probably a more convenient bound to use

in practise, since the reduced dimension is directly controlled when we do a projection, and we tend to interpret ϵ as being implicitly defined by k (since we do not know the constant A). It tells us that as we change the reduced dimension k, we can expect a $\frac{1}{\sqrt{k}}$ growth in the error incurred by the dot-product.

5.3.4 Experimental verification

We computed the quantity $\frac{|\mathbf{v_i}.\mathbf{v_j}-\mathbf{u_i}.\mathbf{u_j}|}{||\mathbf{u_i}||^2+||\mathbf{u_j}||^2}$, which, by our predictions, should get quite small as $\epsilon \to 0$. The results are provided in Table 5.3.

k	Ratio
2	0.2523 ± 0.0616
10	0.1238 ± 0.0099
20	0.0887 ± 0.0048
30	0.0728 ± 0.0032
40	0.0631 ± 0.0024
50	0.0565 ± 0.0019
60	0.0517 ± 0.0016
70	0.0479 ± 0.0013
80	0.0448 ± 0.0012
90	0.0422 ± 0.0010
100	0.0401 ± 0.0009

TABLE 5.3: Mean and standard deviation of the ratio of the quantity in the new bound

We notice that as k increases, the average value of the ratio decreases, as does the variance, which goes down dramatically. These results are presented in Figures 5.8 and 5.9.

This is as theoertically predicted - we are observing that as the error ϵ decreases, so too does the ratio, and further, that the ratio is becoming "tightly" zero i.e. the variance also becomes very close to 0 along with the mean. As a further observation, we can verify that the growth is like $\frac{1}{\sqrt{k}}$ - Figure 5.10 shows the behaviour of the mean ratio and compares it to $\frac{1}{\sqrt{k}}$. Clearly, the results indicate that the dot-product ratio is just a scaled version of the graph of $\frac{1}{\sqrt{k}}$.

5.4 Comparison with existing bound

Note that we have given an explicit bound on how much the dot-product is distorted. Earlier results have only focussed on the variance of the dot-product, which gives us some sense of its behaviour, but is incomplete. In particular, we do not know what portion of samples of the dot-product fall within one standard deviation of the mean, which is something that the probability density function would help us capture.

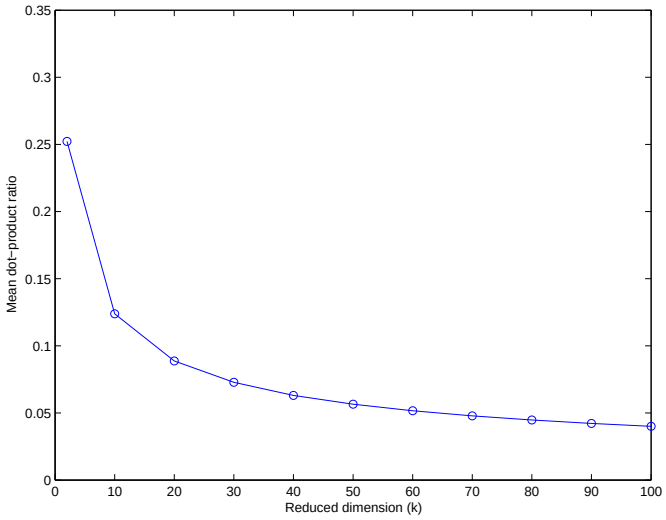


FIGURE 5.8: Mean of the quantity in the bound

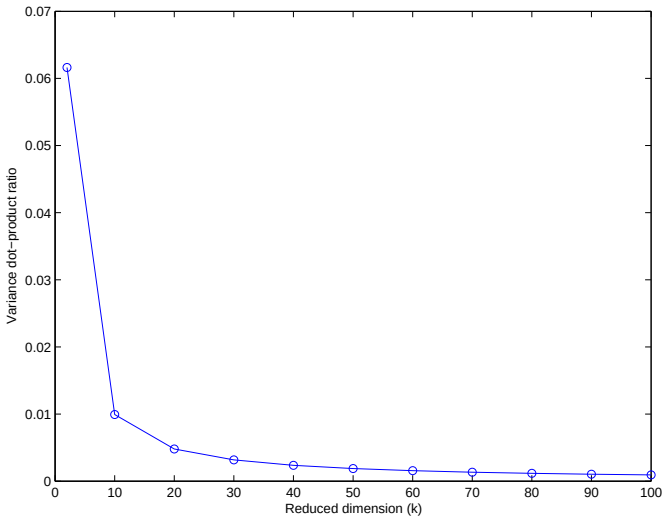


FIGURE 5.9: Variance of the quantity in the bound

This is usually not simple to compute for dot-products, however. For example, consider the case where the projection matrix is Gaussian. Then, we have that

$$v_{1i} = \frac{1}{\sqrt{k}} \sum_j u_{1j} r_{ij} \sim \frac{1}{\sqrt{k}} ||\mathbf{u_1}|| \mathcal{N}(0,1)$$

Consider that the dot-product is

$$\mathbf{v_1}.\mathbf{v_2} = \sum_i v_{1i} v_{2i}$$

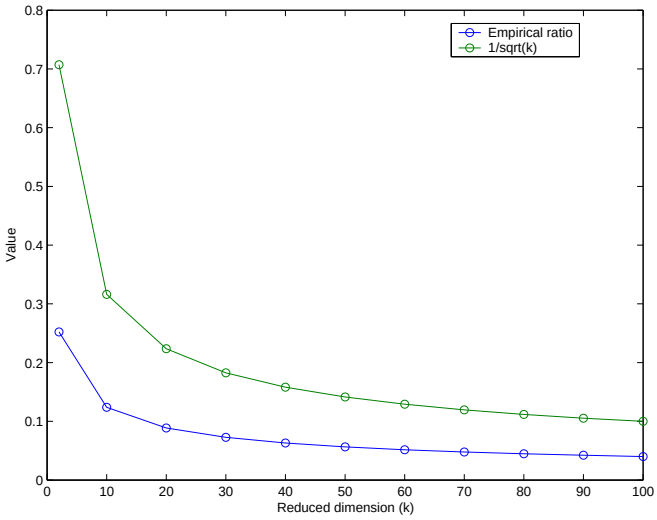


FIGURE 5.10: Comparing the behaviour of the mean ratio and $\frac{1}{\sqrt{k}}$

However, the term $v_{1i}v_{2i}$ is distributed like the product of two Gaussians - this is not a trivial distribution, and in particular, the probability density function of the product of two $\mathcal{N}(0,1)$ variables, as mentioned in our analysis of §8.3, is

$$f(x) = \frac{K_0(|x|)}{\pi}$$

where K_0 is a modified Bessel function of the second kind (Abramowitz and Stegun, 1964). The *sum* of k such variables does not, to our knowledge, have a closed form!

To get from results on variance to something similar to our result, we can try to use some well-known, but necessarily crude, inequalities to get a bound on the probability we exceed some amount, but as we will demonstrate, we cannot get as tight a result as the one we have provided.

We can apply Chebyshev's inequality, which says that

$$Pr(|X| \geq A) \leq \frac{Var(X)}{A^2}$$

Let $X := \mathbf{v_1}.\mathbf{v_2} - \mathbf{u_1}.\mathbf{u_2}$. Then, we know that

$$Var(X) = \frac{1}{k}(||\mathbf{u_1}||^2||\mathbf{u_2}||^2 + (\mathbf{u_1}.\mathbf{u_2})^2)$$

Of course,

$$||\mathbf{u_1}||^2||\mathbf{u_2}||^2 + (\mathbf{u_1}.\mathbf{u_2})^2 = ||\mathbf{u_1}||^2||\mathbf{u_2}||^2(1 + \cos^2\theta)$$

which means

$$Var(X) = \frac{1}{k}||\mathbf{u_1}||^2||\mathbf{u_2}||^2(1+\cos^2\theta)$$

Therefore,

$$Pr(|X| \geq A) \leq \frac{||\mathbf{u_1}||^2||\mathbf{u_2}||^2(1+\cos^2\theta)}{kA^2}$$

So, let $A = \frac{\epsilon}{2}(||\mathbf{u_1}||^2 + ||\mathbf{u_2}||^2)$. Then, we get

$$\begin{aligned} Pr\left(|X| \geq \frac{\epsilon}{2}(||\mathbf{u_1}||^2 + ||\mathbf{u_2}||^2)\right) &\leq \frac{4||\mathbf{u_1}||^2||\mathbf{u_2}||^2(1+\cos^2\theta)}{k\epsilon^2(||\mathbf{u_1}||^2 + ||\mathbf{u_2}||^2)^2} \\ &\leq \frac{2(1+\cos^2\theta)}{k\epsilon^2} \\ &\leq \frac{4}{k\epsilon^2} \end{aligned}$$

The ϵ^2 term in the bottom means that, unless $\epsilon^2 \geq \frac{2}{\sqrt{k}}$, the probability we get is too large to be of any use. If we want the probability to become extremely small, then, what we need is for the k value, and the ϵ value, to be large. Therefore, for certain values of k, ϵ, we are able to get the same result as outlined above, but the probability is not as tight as ours.

5.5 Bound for L_2 sketch

We can use the Indyk's L_2 sketch to get a dot-product estimate, as with projections. In particular, one can prove the following theorem.

THEOREM 7. *For the L_2 sketch, let*

$$e(\mathbf{u_1}.\mathbf{u_2}) := (1-\epsilon^2)\frac{median_l(v_{il}v_{jl})}{median(X^2)}$$

be the estimate of the dot-product. Then, as with the theorem for projections in §5.3, with high probability,

$$|\mathbf{u_i}.\mathbf{u_j} - e(\mathbf{u_i}.\mathbf{u_j})| \leq \frac{\epsilon}{2}(||\mathbf{u_i}||^2 + ||\mathbf{u_j}||^2)$$

This theorem is due to Anh Pham, and is as of yet unpublished[2]. We include his proof of the theorem in Appendix A.1.

As a consequence, we see that even the L_2 sketch can be used to estimate dot-products with at least some guarantees as to the accuracy.

[2]The proof has been submitted as part of a paper to the 2007 SDM conference

CHAPTER 6

Experiments on data-streams

In the previous chapter, we claimed that random projections can be used to keep a L_2-preserving sketch of of a series of streams, and interestingly that this sketch is quicker to update and at least as accurate as the existing sketch of Indyk (Indyk, 2006). We can experimentally test whether the projection-based sketch does indeed give us better results compared to the existing L_2 sketch of (Indyk, 2006). In this chapter, we give such an experimental test of the theoretical guarantees by applying both sketches to an online clustering problem, and comparing results.

It should be noted that the actual application to online clustering is not a major contribution as of itself (by which we mean that the idea of using a sketch to do clustering is not novel (Guha *et al.*, 2000)), but rather the contribution is a verification that the projection sketch is indeed quicker, and at least as accurate as the existing sketch.

The contributions in this chapter include:

- A comparison of the accuracy results of projections and the L_2 sketch in an online clustering problem, showing that the projection sketch is at least as accurate
- A comparison of the runtime costs of projections and the L_2 sketch, showing that the projection sketch is (marginally) quicker

6.1 Experiments on clustering

We compared the accuracy of the two sketches (projection and L_2) by applying them to an online clustering problem. The basic scenario is the same as the previous chapter - we have a set of data points, each of which can be thought of as a stream. The points are updated asynchronously - that is, at every update, an arbitrary coordinate is varied. However, at given moments in time (say every 100 time-steps), we would like to cluster the data as it stands at that instant - we can use this information to find groups points that, at this instant, are "similar" to one another. The problem is that if the streams are high-dimensional, clustering them can be quite expensive. A natural idea is to use a sketch of the streams to help us do clustering in low dimensions, and relate the answer to the original space.

We focussed on clustering that uses Euclidean distances (L_2 norm) and dot-products to judge the "closeness" of two points, since the theoretical work of the previous chapters suggests that these quantities are roughly preserved under the sketches. In particular, we used the k-means (MacQueen, 1967) (which uses distances) and kernel k-means (Shawe-Taylor and Christianini, 2004) (which uses dot-products) algorithms to do clustering. We kept a sketch of the streams and clustered these low-dimensional sketches, and measured how "good" the clustering was. Of course, to test how good the clustering was, we had to actually cluster the original data itself, and compare the results. We would not do this in practise, of course, since we cannot instantiate the data!

Note that the primary aim of the experiments was to check that the projection sketch was at least as accurate as the L_2 sketch, and quicker by some factor - the accuracy compared to the offline algorithm as of itself was only of secondary interest.

6.1.1 Existing work

While our aim is to compare the projection and L_2 sketch, and the clustering itself is tertiary, it is worthwhile nonetheless to look at some existing work that has tried clustering using of streams and sketches. An important paper in the field of clustering of data streams is the work of (Guha *et al.*, 2000), which provides an algorithm for doing clustering of a stream of data using only $O(nk)$ time, where k is the number of clusters. This however does not focus on a stream with asynchronous updates, and instead provides a one-pass algorithm for a stream where we get new points that we have to cluster, but never updates to existing points.

In (Fern and Brodley, 2003), random projections were used to cluster data using the expectation maximization (EM) algorithm in a non-streaming setting. It was found that the results from the use of a single projection were unsatisfactory for their purposes[1], and it was recommended that an *ensemble* of projections be used to try and overcome stability issues related to projections. Our concern is not so much with the accuracy of clustering, but rather with the accuracy of our sketch as compared the "true" value of the underlying stream, and how that compares to the L_2 sketch. Clustering is merely a vehicle towards seeing this, and certainly whether projections produce a bad clustering or not is dependent on the nature of the clustering algorithm e.g. if it is extremely sensitive to perturbations in the data. An ensemble-based approach, however, might be a natural extension to the sketch we propose here.

[1]Note that they required the sketch to discover the number of clusters from the data

6.1.2 Data-set

We used a series of Gaussian clusters as our data-set for clustering. We generated the data as follows - we chose m random cluster centre vectors $\boldsymbol{\mu}_i$, and some variances σ_i^2 for each of them. We then created a mixture of high-dimensional Gaussians based on this:

$$X = \sum_{i=1}^{m} \alpha_i \mathcal{N}(\boldsymbol{\mu_i}, \boldsymbol{\sigma_i^2})$$

where $\sum_{i=1}^{m} \alpha_i = i$.

This forms fairly distinct clusters - examples of these clusters are provided in Figures 6.1 and 6.2. We made n points, where n was varied, which were randomly assigned to one of these clusters.

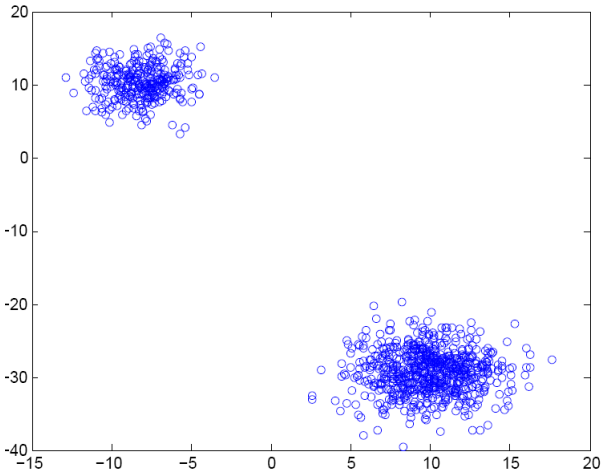


FIGURE 6.1: A 2D Gaussian mixture, $\sigma^2 = 9$. Notice how well separated and densely packed the clusters are

We then generated a new mixture, and for each point in the original clustering, chose a random point in the new clustering as its "partner". The data-stream updates then simply made the original points transform into the new points. For example, say we the first point in the original mixture was $(1, 2, 3)$, and its partner in the new mixture was judged to be $(0, 5, 4)$. Then, the set of generated updates would be $\{(1, 1, -1), (1, 2, 3), (1, 3, 1)\}$. This was done for every pair of partners in the data.

The desired number of clusters C in our clustering step was chosen to be precisely m, the number of clusters in the original data, so that the offline algorithm would be expected to give excellent results, which the streaming algorithms could be compared to.

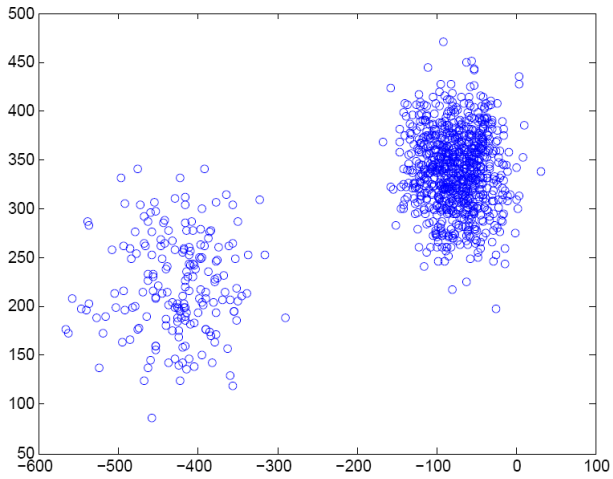


FIGURE 6.2: A 2D Gaussian mixture, $\sigma^2 = 2500$. The clusters now start to overlap, and lose heterogenity

6.1.3 Our implementation

We ran our simulations using MATLAB, on a Pentium-D 3.2 GHz machine with 3.5 GB of RAM. We used Achlioptas' sparse matrix to do our projections, and the inbuilt `randn` to generate the Gaussian variables required for the L_2 sketch. The code is provided in Appendix A.2.

The updates were generated as described above, and we looked to do 50 periodic clusterings of the data. We chose to make the data have 1000 dimensions, noting that by the Johnson-Lindenstrauss lemma, this does not have any impact on the reduced dimension we project to, or the accuracy of the projection. The number of points n was taken to be 100 and 1000, and we also looked at both 2 and 5 Gaussian clusters in the data.

The reduced dimension k was varied, and we observed the effect on the solution quality. We did not follow Achlioptas' bound (Theorem 3) because of literature suggesting that these bounds, while theoretically important, are somewhat weak in practise (Bingham and Mannila, 2001), (Lin and Gunopulos, 2003), (Li *et al.*, 2006), and we can get away with using much lower bounds for our reduction.

6.1.4 Measuring the solution quality

To assess the cluster quality, we used two metrics - an intra-cluster centroid distance metric, and a similarity metric.

The first measure is that of the distance of points to the centroids of the clusters they are in. Suppose we have m clusters $c_1, c_2, \ldots, c_m$ of our points $x_1, x_2, \ldots, x_n$. These clusters are defined by centroids, $r_1, r_2, \ldots, r_m$. The objective of k-means (and kernel k-means) clustering is to minimize

$$\sum_{i=1}^{m} \sum_{x_j \in c_i} ||\mathbf{x_j} - \mathbf{r_i}||^2$$

A natural measure of cluster quality is therefore the value of this objective function - a clustering that has a smaller objective function value than another clustering would be called a "better" clustering. Note that this measure is probably the "truest" measure of (kernel) k-means clustering, since all the algorithm guarantees is that it will try to find a local (but hopefully global) minima for this value.

We took the ratio of this objective function value that was produced by the of the offline algorithm with that of the ones given by the sketches. A value close to 1 indicates that the sketch produces a result that is very close to being that of the offline algorithm, while a value close to 0 indicates that the sketch is almost arbitrarily larger (and hence worse) than the offline algorithm. Note that a value greater than 1 would mean that the sketch gives us a better answer than the offline one!

The second measure, which we henceforth refer to as the *similarity* of two clusterings, finds how many pairs of points lie in the same/different cluster for two given clusterings. That is, we want to see how many pairs are either in the same cluster (no matter what the label of the clusters is), or in different clusters (again no matter what the label of the clusters is).

This is useful for k-means clustering, because it is possible that we get clusterings that "look" different, but are actually the same. For instance, suppose we have two points that we want to put into two clusters, 1 and 2. Are the cluster allocations (1, 2) and (2, 1) the same? Even though they are not identically the same in terms of labelling, clearly they are just *relabellings* of each other, and so for all intents and purposes, they are the same allocations. This similarity measure will report all pairs to be clustered the "same", since in both cases the points are in distinct clusters.

For example, suppose we have the following cluster indicator matrices:

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Now look at pairs of points:

- (1, 2), (1, 3), (2, 4), (3, 4) are in different clusters in both cases
- (1, 4) are in the same clusters in both cases (disregard the fact that they are in cluster 1 in the first case, but cluster 2 in the second case)
- (2, 3) are in different clusters in the first case, but move to the same cluster in the second case. So, this is counted as a 'bad' pair

We say the second clustering is $\frac{14}{16} \times 100 = 87.5\%$ similar to the first - only two pairs (namely, (3, 4) and (4, 3)) are 'bad' out of a possible 16. Note that we have side-stepped the problem that we may label clusters in a different way.

As with the centroid ratio, we computed the similarity between the clusterings produced by the sketches to that of the offline algorithm, to give us the similarity between the online and offline clusterings.

6.2 Incremental k-means clustering

Our first set of experiments was with the k-means clustering algorithm. This algorithm works by computing cluster centroids using Euclidean (L_2) distances, which we know are preserved under random projections. We looked at the solution quality obtained when a number of such clusterings were done on the sketch, which we would expect is a good approximation to the results for clustering on the original data.

6.2.1 Results

6.2.1.1 $n = 100$

For the case where $n = 100, d = 1000$, we see that for two clusters, the results are very accurate as we increase the reduced dimension k (the centroid sum is $\approx 98\%$ of the offline case for projections), but when we generate 5 clusters, as we might expect, the results get slightly less accurate. This is because with 5 clusters, even though they are densely packed, we would expect there to be some overlap and hence uncertainity as to which cluster a point belongs to.

	Similarity		Centroid	
Reduced dimension	Projections	L_2	Projections	L_2
2	52.9091 ± 3.5849	53.1818 ± 3.9464	0.7596 ± 0.2837	0.7515 ± 0.2882
50	96.0606 ± 20.3787	60.1818 ± 16.6496	0.9700 ± 0.3217	0.7758 ± 0.3059
100	98.0000 ± 20.7599	67.6768 ± 20.7246	0.9883 ± 0.2375	0.7947 ± 0.3164
150	76.2626 ± 21.5192	78.5152 ± 18.9006	0.9761 ± 0.3247	0.9274 ± 0.3197
200	87.1212 ± 21.3466	69.5859 ± 16.7517	0.9823 ± 0.3375	0.9059 ± 0.2699

TABLE 6.1: Clustering accuracy results, $n = 100, d = 1000, \sigma^2 = 1$, 2 clusters

	Similarity		Centroid	
Reduced dimension	Projections	L_2	Projections	L_2
2	65.0101 ± 6.9937	65.6162 ± 7.8549	0.6793 ± 0.2437	0.6342 ± 0.2146
50	74.8081 ± 10.1382	73.5758 ± 10.9792	0.8907 ± 0.2570	0.8607 ± 0.2158
100	74.6465 ± 9.6792	72.4040 ± 10.0497	0.8795 ± 0.2540	0.8333 ± 0.2496
150	77.5455 ± 9.7465	73.0707 ± 9.0328	0.9305 ± 0.2289	0.9205 ± 0.3133
200	77.6667 ± 11.9243	75.5758 ± 9.7882	0.9131 ± 0.2345	0.9115 ± 0.2241

TABLE 6.2: Clustering accuracy results, $n = 100, d = 1000, \sigma^2 = 1$, 5 clusters

When we increased σ^2 to a larger value of 9, we get slightly less accurate results, as expected. This is because the data becomes more scattered, and the clusters become less well-defined.

	Similarity		Centroid	
Reduced dimension	Projections	L_2	Projections	L_2
2	51.8333 ± 8.1086	51.6061 ± 9.6518	0.7341 ± 0.2374	0.7355 ± 0.2327
50	76.1061 ± 22.7292	69.5556 ± 17.0110	0.9159 ± 0.2563	0.8245 ± 0.2420
100	92.5859 ± 21.7867	74.8131 ± 18.9967	0.9766 ± 0.2454	0.8447 ± 0.2559
150	93.4242 ± 21.7019	83.9646 ± 21.0888	0.9906 ± 0.2654	0.9227 ± 0.2429
200	94.9545 ± 21.6794	83.1414 ± 20.8421	1.0000 ± 0.2616	0.9174 ± 0.2414

TABLE 6.3: Clustering accuracy results, $n = 100, d = 1000, \sigma^2 = 9$, 2 clusters

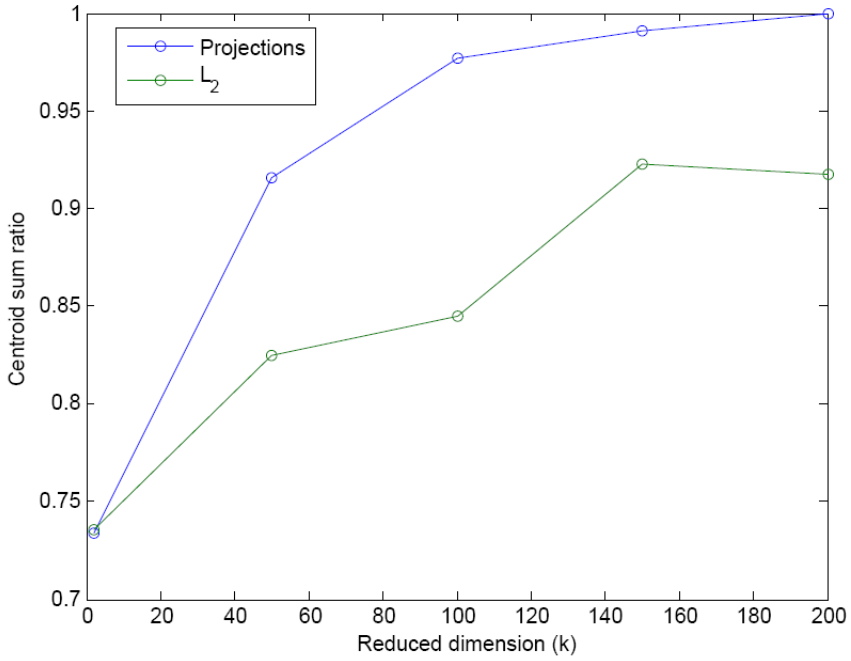


FIGURE 6.3: Centroid sum ratio, $n = 100, \sigma^2 = 9$, 2 clusters

	Similarity		Centroid	
Reduced dimension	Projections	L_2	Projections	L_2
2	65.1010 ± 5.6006	66.5758 ± 5.6794	0.6972 ± 0.1831	0.6877 ± 0.1704
50	76.4141 ± 10.5057	70.2424 ± 10.1072	0.7979 ± 0.2245	0.7476 ± 0.2081
100	81.1313 ± 11.0703	76.9192 ± 11.6677	0.8079 ± 0.2908	0.7878 ± 0.2451
150	76.3131 ± 11.2852	78.1515 ± 11.7283	0.9046 ± 0.2872	0.8530 ± 0.2825
200	78.3535 ± 10.2692	79.7576 ± 9.8988	0.8692 ± 0.2382	0.8522 ± 0.2519

TABLE 6.4: Clustering accuracy results, $n = 100, d = 1000, \sigma^2 = 9$

6.2.1.2 $n = 1000$

When the number of points increased, the results were a bit different to the $n = 100$ case. In particular, we see that the accuracy becomes quite low when the reduced dimension is small, but as this reduced dimension increases the results approach that of the $n = 100$ case. This is in keeping with the Johnson-Lindenstrauss lemma, because as we increase the number of points, we would expect that the reduced dimension k would have to be made correspondingly larger.

Reduced dimension	Similarity		Centroid	
	Projections	L_2	Projections	L_2
2	60.8032 ± 11.5492	53.5324 ± 15.3135	0.7368 ± 0.2723	0.7282 ± 0.2134
50	52.6438 ± 22.5916	68.5490 ± 19.2462	0.8488 ± 0.2660	0.7818 ± 0.2488
100	65.8707 ± 23.4064	67.9973 ± 21.5126	0.8858 ± 0.3394	0.7758 ± 0.3073
150	97.2757 ± 24.0969	62.3288 ± 20.2169	1.0000 ± 0.3174	0.7879 ± 0.3238
200	79.6296 ± 24.1815	52.0328 ± 20.2287	0.9287 ± 0.3412	0.7732 ± 0.3424

TABLE 6.5: Clustering accuracy results, $n = 100, d = 1000, \sigma^2 = 9$, 2 clusters

Reduced dimension	Similarity		Centroid	
	Projections	L_2	Projections	L_2
2	64.1744 ± 6.6183	61.9279 ± 5.5459	0.6727 ± 0.1980	0.6491 ± 0.2027
50	78.6201 ± 10.1359	72.1879 ± 9.8855	0.8739 ± 0.2716	0.7475 ± 0.2296
100	78.4050 ± 11.9351	71.2059 ± 11.9053	0.9002 ± 0.2831	0.7475 ± 0.2860
150	79.6142 ± 12.1918	74.2481 ± 11.1874	0.9100 ± 0.2795	0.8519 ± 0.2447
200	79.1936 ± 11.8065	76.1932 ± 10.3753	0.9170 ± 0.2647	0.8654 ± 0.2648

TABLE 6.6: Clustering accuracy results, $n = 1000, d = 1000, \sigma^2 = 1$, 5 clusters

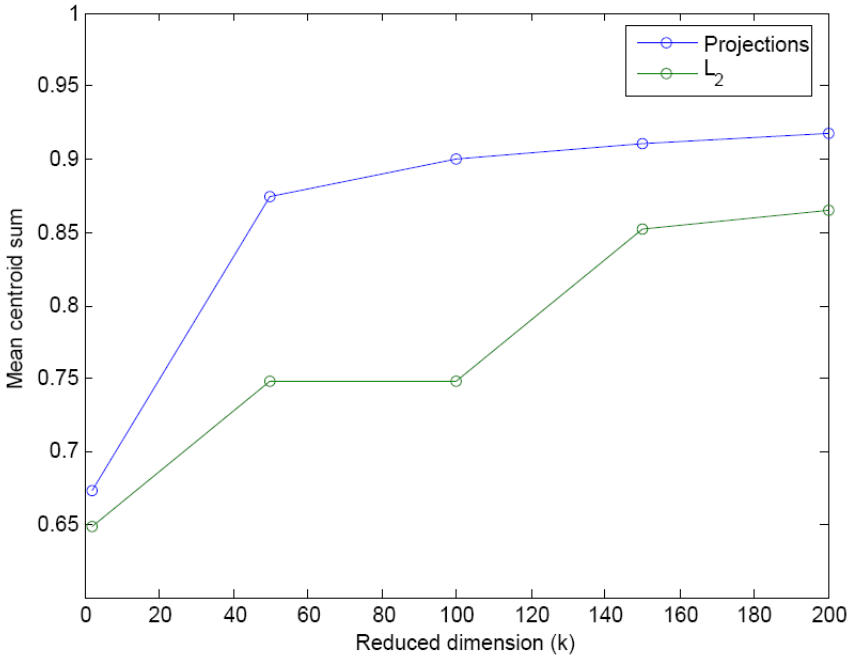


FIGURE 6.4: Centroid sum ratio, $n = 1000, \sigma^2 = 1$, 5 clusters

6.3 Incremental kernel k-means clustering

6.3.1 Kernel matrix and distances

Kernel k-means clustering works using a *kernel matrix*, K, where

$$K(i,j) =< \mathbf{x_i}, \mathbf{x_j} >$$

	Similarity		Centroid	
Reduced dimension	Projections	L_2	Projections	L_2
2	54.4152 ± 10.3066	52.9534 ± 14.7356	0.7235 ± 0.2820	0.7032 ± 0.2718
50	63.9985 ± 22.4360	67.6101 ± 21.4261	0.8277 ± 0.3296	0.8256 ± 0.2808
100	78.2684 ± 23.4060	62.2993 ± 22.3521	0.9185 ± 0.2465	0.7639 ± 0.3265
150	73.0516 ± 21.1794	61.8087 ± 21.7738	0.9271 ± 0.3365	0.8626 ± 0.3064
200	72.6275 ± 24.6650	63.8384 ± 23.2786	0.8701 ± 0.3248	0.8544 ± 0.3129

TABLE 6.7: Clustering accuracy results, $n = 1000, d = 1000, \sigma^2 = 9$, 2 clusters

	Similarity		Centroid	
Reduced dimension	Projections	L_2	Projections	L_2
2	63.9349 ± 7.7109	65.0684 ± 8.4074	0.7042 ± 0.2387	0.7614 ± 0.2367
50	77.0378 ± 9.5556	71.0085 ± 11.1355	0.9474 ± 0.2800	0.8314 ± 0.2556
100	84.0634 ± 12.3335	75.3517 ± 13.7827	0.9838 ± 0.2237	0.9174 ± 0.2256
150	86.0173 ± 10.6433	73.4006 ± 13.2800	0.9485 ± 0.2201	0.8950 ± 0.2235
200	85.6849 ± 8.5192	86.2184 ± 10.7646	0.9706 ± 0.2673	0.9496 ± 0.2081

TABLE 6.8: Clustering accuracy results, $n = 1000, d = 1000, \sigma^2 = 9$, 2 clusters

gives us the dot-product between two points. The idea of the algorithm is simply that we can express a normal k-means clustering problem in terms of distances, and that distances can in turn be expressed in terms of dot-products:

$$||\mathbf{x} - \mathbf{y}||^2 = (\mathbf{x} - \mathbf{y}).(\mathbf{x} - \mathbf{y}) = ||\mathbf{x}||^2 - 2\mathbf{x}.\mathbf{y} + ||\mathbf{y}||^2$$

Of course, dot-products correspond to entries in the kernel matrix, and therefore we can replace the computations of the dot-products by simply using the kernel matrix entries.

$$||\mathbf{x} - \mathbf{y}||^2 = K(\mathbf{x}, \mathbf{x}) - 2K(\mathbf{x}, \mathbf{y}) + K(\mathbf{y}, \mathbf{y})$$

In fact, if we use kernel k-means clustering, then as mentioned earlier, we can express the problem in terms of entries from the kernel matrix - namely, dot-products of points. We show how we can keep an incremental approximation of the kernel as well.

6.3.2 Projections for kernels

To do kernel k-means clustering, we keep not only a sketch for the data, but an approximation for the kernel - we do not want to run the cost of recomputing the kernel matrix at every time step, because updating a single entry takes $\Theta(d)$ time, and we are considering very large d. But, if we instead use an approximation $\widehat{K}$ for the kernel, then the updates will take much less time. The approximation we propose is very simple, and is derived from the sketch of the

stream:

$$\widehat{K}_{ij} := \text{est}(\mathbf{u_i}.\mathbf{u_j})$$

where est is the sketch-based estimate.

Now, we can set the values of the approximate kernel matrix either:

- Incrementally - when we get an update for the original matrix, we affect the dot-products of n pairs, each of which we recompute. The work required is $\Theta(nk)$
- "Brute-force" i.e. a $\Theta(n^2)$ recomputation from scratch

The right approach depends on how often we want to cluster. Let us say we want to cluster the points with a periodicity of p. Then, after T steps, the incremental approach will require nkT work, whereas the former will require $\frac{n^2kT}{p}$ work. So, the brute-force approach will be better if

$$\frac{n^2}{p} < n \iff p > n$$

So, we have a potentially quick way to do kernel k-means clustering on-the-fly, using a sketch to help estimate the true kernel matrix.

6.3.3 Accuracy guarantees

Since it is not a standard guarantee, we recap the accuracy result for dot-products - note that by Chapter 5, for either sketch, we have that, with high probability,

$$|\mathbf{v_i}.\mathbf{v_j} - \mathbf{u_i}.\mathbf{u_j}| \leq \frac{\epsilon}{2}(||\mathbf{u_i}||^2 + ||\mathbf{u_j}||^2)$$

The error ratio is therefore, with high probability

$$\left|\frac{\mathbf{v_i}.\mathbf{v_j} - \mathbf{u_i}.\mathbf{u_j}}{\mathbf{u_i}.\mathbf{u_j}}\right| \leq \frac{\epsilon}{2|\mathbf{u_i}.\mathbf{u_j}|}(||\mathbf{u_i}||^2 + ||\mathbf{u_j}||^2)$$

Therefore, we know that either sketch will on average preserve dot-products, and the above upper bound indicates that when ϵ is small enough (or equivalently, when the reduced dimension k is large enough), our estimates will be fairly well concentrated about the expected value. Of course, we cannot expect as good results as with distances, because there are going to be more "bad" cases for dot-products that we cannot get around with simple estimates.

6.3.4 Results

6.3.4.1 $n = 100$

As with the k-means algorithm, we first tried the algorithm on a 100×1000 data-set, and compared the accuracy of the two sketches.

We see that for $\sigma^2 = 1$, the results are only slightly worse than the case for the k-means algorithm (the decrease is no surprise given that the guarantees are not as strong).

	Similarity		Centroid	
Reduced dimension	Projections	L_2	Projections	L_2
2	52.4848 ± 7.2812	52.1717 ± 5.6581	0.7642 ± 0.2858	0.7678 ± 0.2816
50	73.1717 ± 19.1737	51.5152 ± 5.6306	0.8641 ± 0.3154	0.8021 ± 0.2809
100	82.4747 ± 21.1076	50.6364 ± 5.5649	0.8958 ± 0.3165	0.7806 ± 0.2856
150	74.8687 ± 20.8541	51.1515 ± 5.5049	0.9258 ± 0.3426	0.8585 ± 0.2854
200	75.6768 ± 21.3726	51.1313 ± 5.8781	0.9419 ± 0.3229	0.7642 ± 0.2825

TABLE 6.9: Clustering accuracy results, $n = 100, d = 1000, \sigma^2 = 1$, 2 clusters

	Similarity		Centroid	
Reduced dimension	Projections	L_2	Projections	L_2
2	64.8283 ± 5.5566	64.0101 ± 5.9354	0.6925 ± 0.1835	0.6806 ± 0.1828
50	70.3737 ± 10.6541	61.3838 ± 6.2871	0.8519 ± 0.2446	0.6706 ± 0.1920
100	71.7879 ± 10.0429	64.1616 ± 4.7472	0.8654 ± 0.2799	0.6550 ± 0.1955
150	72.8384 ± 11.1226	64.3535 ± 5.3314	0.8714 ± 0.3238	0.6701 ± 0.2447
200	72.4242 ± 11.1658	63.3131 ± 5.2586	0.8079 ± 0.2070	0.6530 ± 0.1386

TABLE 6.10: Clustering accuracy results, $n = 100, d = 1000, \sigma^2 = 1$, 5 clusters

When $\sigma^2 = 9$, again as expected, the results decrease somewhat.

	Similarity		Centroid	
Reduced dimension	Projections	L_2	Projections	L_2
2	67.8586 ± 17.1542	57.9899 ± 10.8304	0.7477 ± 0.2013	0.7114 ± 0.2622
50	74.9091 ± 21.5442	59.0808 ± 11.8157	0.8887 ± 0.2686	0.7140 ± 0.2596
100	69.3131 ± 20.7244	57.2222 ± 10.9195	0.9328 ± 0.3279	0.7068 ± 0.2649
150	82.4747 ± 20.9869	57.9899 ± 13.4011	0.9272 ± 0.2454	0.7413 ± 0.2627
200	80.0606 ± 22.3009	57.1818 ± 12.9029	0.9694 ± 0.2437	0.7381 ± 0.2709

TABLE 6.11: Clustering accuracy results, $n = 100, d = 1000, \sigma^2 = 9$, 2 clusters

6.3.4.2 $n = 1000$

When the number of point is increased, we see that the results are again mostly commensurate with the ones from the k-means algorithm, though there is a slight decrease in the accuracy, again as expected.

	Similarity		Centroid	
Reduced dimension	Projections	L_2	Projections	L_2
2	68.5556 ± 5.7098	66.4444 ± 4.6156	0.6625 ± 0.1471	0.6310 ± 0.1379
50	76.5657 ± 11.0360	65.1414 ± 4.9313	0.7687 ± 0.2019	0.6535 ± 0.2058
100	77.3737 ± 11.1476	66.1717 ± 5.4500	0.8205 ± 0.2402	0.6283 ± 0.2078
150	76.0202 ± 12.0258	65.4949 ± 6.3961	0.8222 ± 0.2860	0.6616 ± 0.1823
200	78.1818 ± 11.2836	66.3434 ± 6.0104	0.8455 ± 0.3152	0.6244 ± 0.1809

TABLE 6.12: Clustering accuracy results, $n = 100, d = 1000, \sigma^2 = 9$, 5 clusters

	Similarity		Centroid	
Reduced dimension	Projections	L_2	Projections	L_2
2	52.7087 ± 9.2750	56.2347 ± 8.9585	0.7724 ± 0.2808	0.7158 ± 0.2767
50	75.6526 ± 18.8054	54.6071 ± 8.8600	0.8997 ± 0.3216	0.7103 ± 0.2809
100	71.7038 ± 21.3313	52.2428 ± 8.7759	0.9200 ± 0.3211	0.7207 ± 0.2734
150	82.9353 ± 15.4645	55.1801 ± 8.7550	0.8771 ± 0.3143	0.7320 ± 0.2789
200	88.0825 ± 18.4925	53.6004 ± 8.8658	0.9405 ± 0.3146	0.7148 ± 0.2783

TABLE 6.13: Clustering accuracy results, $n = 1000, d = 1000, \sigma^2 = 9$, 2 clusters

	Similarity		Centroid	
Reduced dimension	Projections	L_2	Projections	L_2
2	65.1590 ± 5.3797	62.5070 ± 4.6603	0.6531 ± 0.1449	0.6211 ± 0.1592
50	81.9962 ± 11.1301	60.8544 ± 5.7097	0.9082 ± 0.2322	0.6129 ± 0.1916
100	80.0811 ± 12.6400	61.3219 ± 5.9844	0.8898 ± 0.2358	0.6426 ± 0.1783
150	81.7725 ± 12.0323	59.8452 ± 6.2278	0.9028 ± 0.2465	0.6367 ± 0.2010
200	78.0733 ± 9.4549	60.1058 ± 4.6022	0.9480 ± 0.2147	0.6400 ± 0.1905

TABLE 6.14: Clustering accuracy results, $n = 1000, d = 1000, \sigma^2 = 9$, 5 clusters

6.4 Analysis of clustering experiments

If we compare the two sketches (which was the main point of the experiments), we see that in nearly all cases, as we prediced theoretically, the projection-based sketch gives us better results than the L_2 sketch, despite having lower complexity in terms of the type of matrix used. We can see that the results become more pronounced when we increase the number of points to 1000 - for instance, in the case $n = 1000, \sigma^2 = 9$, we see that the L_2 sketch does not touch a centroid sum ratio of 0.9 (or 90% accuracy), whereas the projection sketch easily reaches this when it gets to $k = 150$. This suggests that the projection sketch can give the same accuracy as the L_2 sketch *using a smaller reduced dimension*, and hence with less space (since the space required is $\Theta(nk)$).

We notice that for both the k-means and kernel k-means algorithm, the results, as expected, get better when we increase the reduced dimension. The results for small k ($k = 2$)

are markedly different from those for $k = 50$, and so we would recommend that if the sketches are to be applied to clustering, they be used for k at least as large as this. We note that in many applications, $k = 2$ can actually produce decent results, but in this case, the results get quite poor when the data starts getting slightly non-trivial!

However, we do also observe that there is a fairly large variance in the results, which is similar to the observations made in (Fern and Brodley, 2003), indicating that either sketch can be quite unstable. This suggests that perhaps an ensemble approach, as suggested in the paper, might similarly produce better results (and this is certainly some potential future work).

We notice that the cluster similarity measure does *not* seem to correlate well with the centroid sum measure - the latter is a "truer" measure of the quality of the clustering, since it is the objective function for the k-means and kernel k-means algorithm, and so we are led to conclude that the cluster similarity measure is perhaps not all that useful a measure, at least in the online clustering problem. The results essentially say that it is possible for either sketch to produce results that are not especially similar to those of the offline algorithm, but still produce "good" clusters as recognized by a high centroid sum ratio.

Regardless of the efficacy of the cluster similarity measure, we see that projections still outperform the L_2 sketch in this aspect, which was the purpose of the experiments. We next look at a runtime comparison of the two sketches.

6.5 Experiments on runtime

We mentioned in §4.6 that one advantage of projections over the L_2 sketch is in the nature of the sparse matrix used, which allows for more efficient updates. In this section, we experimentally test this claim, and also compare ways in which we can generate variables for the L_2 sketch.

6.5.1 Generating 2-stable values

Certainly the bottleneck for the update time of the L_2 sketch, or for that matter any L_p sketch, is the generation of the stable distribution. For $p = 2$, we need to make Gaussian variables, which can be time consuming. The standard method to do this is the Box-Muller transform, that takes uniform random variables and makes two independent Gaussian variables (BoxMuller, 2006):

$$z_0 = \cos(2\pi\phi)\sqrt{-2\log r}$$

$$z_1 = \sin(2\pi\phi)\sqrt{-2\log r}$$

for $r, \phi \in (0, 1]$. The variable z_1 can be fed into an algorithm that uses the normal variable to make a new variable, and the process continues.

In (Cormode *et al.*, 2003), the following method is suggested for p-stable distributions, also using two uniform random variables r_1, r_2. We define $\theta := \pi r_1 - \frac{\pi}{2}$, and then generate

$$X = \text{stable}(r_1, r_2, p) = \frac{\sin p\theta}{\cos^{1/p}\theta} \left(\frac{\cos(\theta(1-p))}{\log \frac{1}{r_2}} \right)^{\frac{1-p}{p}}$$

For $p = 2$, this becomes

$$\begin{aligned} X &= \frac{\sin 2\theta}{\sqrt{\cos\theta}} \sqrt{\frac{\log \frac{1}{r_2}}{\cos\theta}} \\ &= \frac{2\sin\theta\cos\theta}{\cos\theta} \sqrt{\log \frac{1}{r_2}} \\ &= 2\sin\theta \sqrt{\log \frac{1}{r_2}} \\ &= 2\sin\left(\pi r_1 - \frac{\pi}{2}\right) \sqrt{\log \frac{1}{r_2}} \text{ by definition of } \theta \\ &= -2\cos \pi r_1 \sqrt{\log \frac{1}{r_2}} \end{aligned}$$

This is not a particularly bad method in terms of conceptual simplicity (all we need to do is make two uniform random variables and apply some well-known functions on them), but it is clearly going to be expensive, as we are using three non-trivial operations - cos, $\sqrt{}$, and log. Certainly there are clever ways of computing these (with lookup tables, etc.), but we would still expect there to be significant expenses since there are three of these operations.

There is also a more complicated, but more efficient, approach known as the Ziggurat method for making Gaussian variables (Marsaglia and Tsang, 2000). This method is employed by MATLAB's `randn` function for generating Gaussian random variables (Moler, 2001). This method does not require the use of non-trivial functions, and so, theoretically, is a more efficient option.

We checked to see what performance impact the choice of 2-stable value generation has on the results of update times for t_s. We made the usual updates to the sketch by generating stable values by first the (Cormode *et al.*, 2003) method (which we call the "Suggested L_2" method), and also tried using the Ziggurat method. Our results are shown in Table 6.15 and Figure 6.5.

Reduced dimension	Suggested L_2 time	Ziggurat time
100	0.2453 ± 0.0104	0.0298 ± 0.0049
200	0.4266 ± 0.0076	0.0390 ± 0.0082
300	0.6110 ± 0.0049	0.0454 ± 0.0047
400	0.8001 ± 0.0068	0.0530 ± 0.0082
500	0.9907 ± 0.0080	0.0625 ± 0.0005
600	1.1891 ± 0.0114	0.0719 ± 0.0081
700	1.3969 ± 0.0406	0.0953 ± 0.0186
800	1.6280 ± 0.0697	0.1065 ± 0.0143

TABLE 6.15: Mean time and standard deviation, in seconds, required for 1000 sketch updates, suggested L_2 vs. Ziggurat method

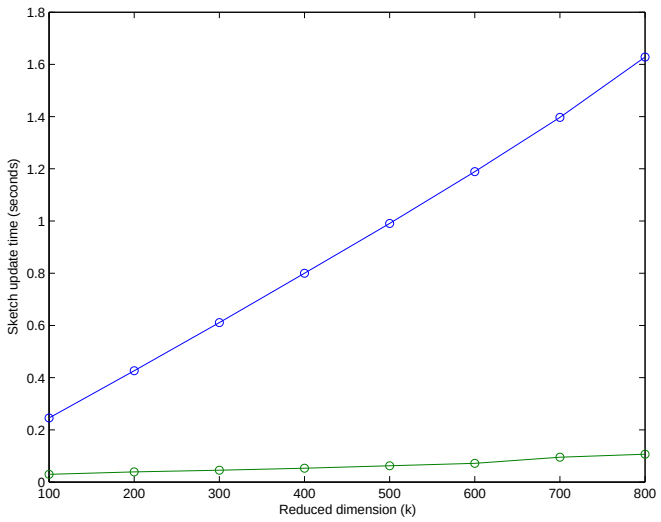


FIGURE 6.5: The sketch update time increases linearly with the reduced dimension, as we would expect. The Ziggurat sketch can be seen to have a constant factor improvement over the Hamming sketch approach, which might prove useful in applications where we have to do a lot of sketch updates in very quick succession

The Ziggurat method can be seen to be significantly faster than the suggested method in (Cormode *et al.*, 2003), and for large k we can see that it seems to be around 10 times quicker! It would be advisable then to use the Ziggurat approach, even if the implementation might involve more work (this is probably not an issue in a lot of settings where there is already an implementation available). So, the (Cormode *et al.*, 2003) approach does not appear to be particularly useful for the special case $p = 2$, but it certainly is useful for general p!

6.5.2 Gaussian vs. uniform random

It is intuitive that the generation of a Gaussian random variable would, on average, take longer than the generation of a uniform random variable. Of course, the precise difference depends a lot on implementation; for instance, if we use the Box-Muller transform to generate a Gaussian, we would expect worse results than if we use the Ziggurat method (Marsaglia and Tsang, 2000).

MATLAB implements an optimized version of the Ziggurat method to generate Gaussian variables (Moler, 2001). Our results on MATLAB indicated that, surprisingly, the generation of Gaussian variables via the built-in `randn` function was faster than the generation of uniform random variables via the built-in `rand`. These tests were performed on a Pentium-D 3.2 GHz machine with 3.5 GB of RAM.

# values	Uniform	Gaussian
10^5	0.006074 ± 0.000050	0.004282 ± 0.000056
10^6	0.062306 ± 0.000252	0.044472 ± 0.000120
10^7	0.625017 ± 0.006305	0.441509 ± 0.000677
10^8	6.22510 ± 0.017631	4.415166 ± 0.022836

TABLE 6.16: Average time taken and standard deviation, in seconds, to generate uniform and Gaussian random variables over 10,000 runs, with MATLAB

We also tried the GNU Scientific Library [2] for C, which provides a Mersenne-twister implementation for uniform random variables (`gsl_rng_mt19937`), and a Ziggurat-based method for generating normal random variables (`gsl_ran_gaussian_ziggurat`). Our results here indicated that uniform random generation was on average faster than the generation of a Gaussian.

# variables	Uniform	Gaussian
10^5	0.0043 ± 0.0049	0.0165 ± 0.0050
10^6	0.0441 ± 0.0048	0.1572 ± 0.0045
10^7	0.4726 ± 0.0267	1.6406 ± 0.0945
10^8	4.4054 ± 0.1700	17.8785 ± 0.5173

TABLE 6.17: Average time taken and standard deviation, in seconds, to generate uniform and Gaussian random variables over 10,000 runs, with C and GSL

This suggests that the precise difference in generation time depends a lot on the specific implementation. It is unclear why the results in MATLAB are markedly different to those in C++; it might be a caching issue, or it could even be that the Gaussian generator uses a different uniform random generator than the standard one.

[2] `http://www.gnu.org/software/gsl/`

Regardless, in practise we would expect most real applications to be in C++ rather than in a simulation environment like MATLAB, so the C++ results can be taken as being the more realistic of the two.

6.5.3 Sketch update time - projections vs. L_2

We measured the time taken for the update of each of the projection and L_2 sketches, to see what savings, if any, the projection approach provides. The results of §6.5.2 indicate that the time required for the generation of the projection matrix itself, which is either sparse or Gaussian, is highly implementation dependent. As a result, for measuring the sketch update, we only measured the time taken for the actual update i.e. multiplication and addition of the update value into the sketch.

We varied the number of reduced dimensions k, and tried a large volume of updates of each of the sketches. Our results are presented in Table 6.18.

Updates	k	Projections	L_2
10^4	5	0.0588 ± 0.0007	0.0594 ± 0.0006
10^4	100	0.0772 ± 0.0006	0.0870 ± 0.0007
10^4	200	0.0981 ± 0.0007	0.1096 ± 0.0005
10^4	300	0.1195 ± 0.0009	0.1265 ± 0.0007
10^4	400	0.1312 ± 0.0019	0.1438 ± 0.0013
10^4	500	0.1580 ± 0.0007	0.1608 ± 0.0012
10^5	5	0.6061 ± 0.0037	0.5977 ± 0.0036
10^5	100	0.8055 ± 0.0048	0.8623 ± 0.0046
10^5	200	1.0476 ± 0.0079	1.0928 ± 0.0058
10^5	300	1.2738 ± 0.0047	1.2683 ± 0.0057
10^5	400	1.4301 ± 0.0051	1.4159 ± 0.0074
10^5	500	1.4601 ± 0.0040	1.5827 ± 0.0035
10^6	5	5.8064 ± 0.0066	5.8430 ± 0.0146
10^6	100	7.7333 ± 0.0119	8.6419 ± 0.0311
10^6	200	10.6300 ± 0.0353	10.6701 ± 0.0513
10^6	300	11.7557 ± 0.0575	12.5707 ± 0.0777
10^6	400	13.0989 ± 0.0941	14.1790 ± 0.0220
10^6	500	15.6372 ± 0.0197	15.9051 ± 0.0276

TABLE 6.18: Average time and standard deviation, in seconds, for sketch updates

6.6 Remarks on runtime experiments

Our experiments indicated that the update time for the projection sketch was faster than that of the L_2 sketch, which is to be expected, since our sketch involves a $\frac{2}{3}$rds sparse row, allowing

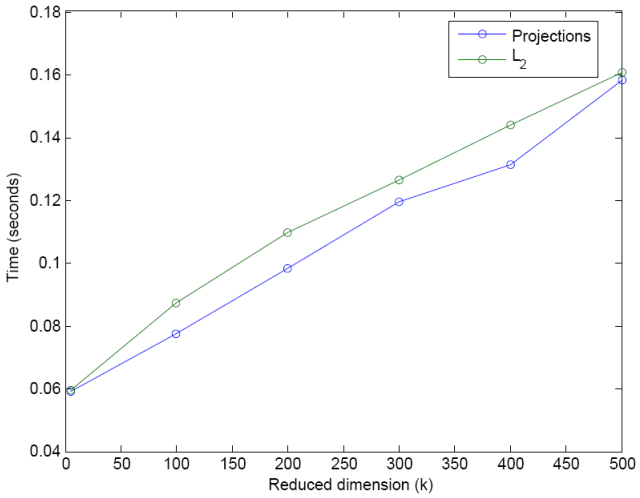


FIGURE 6.6: The time taken to make 10^4 updates for the sketch, projections vs. L_2

for a reduction in the number of multiplications that are required. However, the factor of improvement was less than expected - we were expecting something roughly 3 times quicker (since we are using a sparse row). We were not able to discern exactly what the reason for this was, but it certainly warrants a closer investigation.

Also, there is a linear growth in the difference as k increases, which is again as expected (as each update requires $\Theta(k)$ time). Surprisingly, though, the variance of the results of the projection sketch seemed to be larger than that for the L_2 sketch - this is possibly again due to the specifics of the implementation.

Combining the accuracy results of §6.4, we can make the following claim that serves as an experimental verification of the one made in §4.6.2.

CLAIM. *For the online clustering problem, the projection based sketch can achieve the same accuracy as the L_2 sketch using less space, and is (marginally) quicker to update.*

Part 3

Theoretical results

CHAPTER 7

Dimension bounds

In this chapter, we look at Achlioptas' dimension bounds, and explore their tightness and applicability. We show that in some cases, his bounds become "meaningless" (in a sense which will be defined later). We also show why his bounds are essentially tight in general, but provide a significant improvement over the bounds for some special cases.

This chapter includes the following contributions:

- A novel result that gives a necessary and sufficient condition for the Achlioptas bound to be "meaningless"
- A significant improvement of the tightness of the Achlioptas bound for cases where the desired error parameter ϵ is above 0.75, and a note on how this threshold value may be decreased
- A discussion of why the Achlioptas bound cannot be improved for general ϵ, unless we remove the union bound

7.1 The limits of Achlioptas' lowest dimension

Recall that in §2.3.1, we detailed Achlioptas' projection theorem, where he introduced the "lowest reduced dimension"

$$k_0 = \frac{4 + 2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n$$

The theorem says that for $k \geq k_0$, we are able to make guarantees on the projection preserving pairwise distances. Therefore, k_0 becomes the lowest dimension to which we can reduce our data, while still keeping projection guarantees.

Note that there is no mention of the original dimension, d in the above equation. Does that mean we need not pay heed to the original dimension when doing our projection? It should be clear that this is not quite accurate, because it is certainly possible for us to choose values of n, ϵ, β, and of course d, so that $k_0 > d$ - that is, the reduced dimension is larger than the original dimension! In most cases, this is probably not desired[1], and so we would call the bound in such cases "meaningless".

[1] What might be exceptions to this rule? Perhaps a study of the use of projections to reduce noise by projecting onto higher-dimensions, but we are not aware of such work

We are led to ask if it is possible to have a situation where the above bound can *never* be satisfactory, i.e., it always gives us an answer that is larger than the original dimension. We show that in fact it is possible to have such bad cases, but that they are probably unlikely to arise in practise. The following lemma, which is a novel result, gives a necessary and sufficient condition for the bound to be "meaningless".

LEMMA 13. *Suppose* $\epsilon \in \left(0, \frac{3}{2}\right)$. *Then,*

$$k_0 \geq d(\forall \epsilon, \beta > 0) \iff n \geq e^{d/24}$$

That is, we cannot find any values of ϵ *or* β *that give us a reduced dimension that is smaller than the original dimension if, and only if, the number of points is at least exponential in the number of dimensions (plus scaling).*

PROOF. From the above bound,

$$k_0 \geq d(\forall \epsilon, \beta > 0) \iff d \leq \frac{4 + 2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n(\forall \epsilon, \beta > 0)$$

Note that our restriction on ϵ means that the right-hand side is always positive. So, we must require

$$\epsilon^2/2 - \epsilon^3/3 \leq \frac{4 + 2\beta}{d} \log n(\forall \epsilon, \beta > 0)$$

Consider the left-hand side as a function of ϵ, and find its maximum:

$$g(\epsilon) := \epsilon^2/2 - \epsilon^3/3$$
$$g'(\epsilon) = \epsilon - \epsilon^2 = \epsilon(1 - \epsilon)$$
$$g''(\epsilon) = 1 - 2\epsilon$$

We see that we have a maxima at $\epsilon = 1$, where $g'(\epsilon) = 0, g''(\epsilon) < 0$. Here, $g(1) = 1/6$. So, if we want the inequality to hold for all ϵ, then we see that we must have

$$\epsilon^2/2 - \epsilon^3/3 \leq \frac{1}{6} \leq \frac{4 + 2\beta}{d} \log n(\forall \beta > 0)$$

Again, if we want this to hold for all $\beta > 0$, we must have

$$\frac{1}{6} \leq \frac{4}{d} \log n \leq \frac{4 + 2\beta}{d} \log n$$

This will hold iff

$$\log n \geq \frac{d}{24} \iff n \geq e^{d/24}$$

□

Remark 1 The above restriction on ϵ is implicit in Achlioptas' theorem, and so we have not introduced a new constraint here. If it is not satisfied, Achlioptas' bound is negative!

Remark 2 This lemma does **not** say that if $n \geq e^{d/24}$, we cannot reduce the dimension of the data to within ϵ accuracy; it merely says that the Achlioptas prediction fails us. That is, it says **we cannot make *any* choice** of $\epsilon, \beta > 0$ that will give us a reduced dimension k that is actually *smaller* than the original dimension, d. As we will verify in §7.2, the Achlioptas prediction is not a tight-bound, and so this does not say that we cannot do a distance preserving projection at all.

So, in summary, the lemma says that if the number of points in our data-set is more than exponential in the number of dimensions of the data, we cannot find any assignment of ϵ, β that makes the Achlioptas bound useful/meaningful. In such cases, we would have to look at using an empirical bound to do our projection.

7.2 Achlioptas' vs. true bounds

How does the Achlioptas bound compare to the true lowest bounds? It was shown in (Li *et al.*, 2006) that the bounds were sometimes up to 50% weak. We ran simple experiments that confirmed the weakness of the bounds.

Figure 7.1 shows a graph that compares the Achlioptas prediction to the empirically discovered lowest dimension bound. This experiment was run for the case $n = 1000, \epsilon = 0.25$. The empirical bound was calculated via a binary search in MATLAB.

We can see that there is a vaguely logarithmic growth to the lowest dimension, which is consistent with the Johnson-Lindenstrauss embedding theorem. However, it appears that the Achlioptas bound is quite conservative, giving an answer that is roughly twice that of the "true" bound.

Clearly, the behaviour of the function of ϵ in the bottom has a large impact on the nature of the bound. In Figure 7.2, we see a graph for the case $\epsilon = 0.25$, and the fraction of points that are allowed to be distorted by more than ϵ is chosen to be 12.5%. We observe the behaviour of this lowest dimension as the number of points n increases, and include graphs of the functions $\frac{1}{\epsilon}$, $\frac{1}{\epsilon - \epsilon^2}$, and Alon's lower bound (Alon, 2003) $\frac{1}{\epsilon^2 \log 1/\epsilon}$.

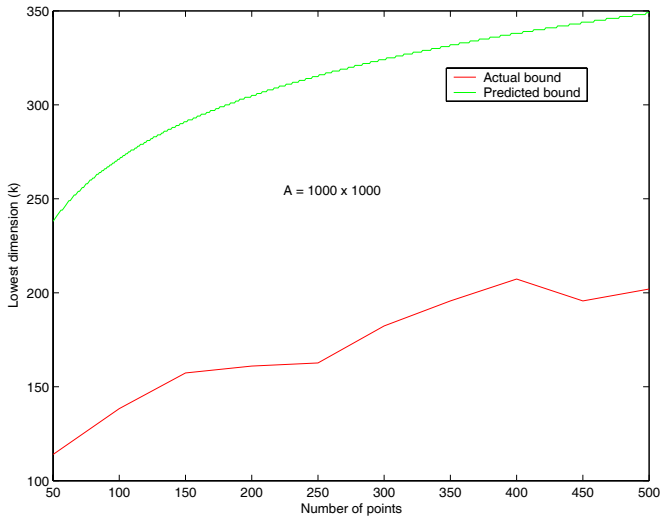


FIGURE 7.1: Weakness of the Achlioptas bound

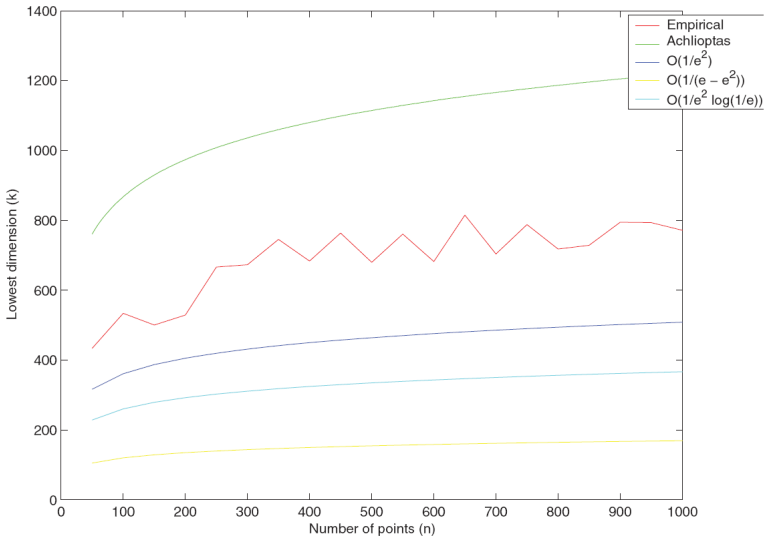


FIGURE 7.2: Lowest dimension, with bounds

We see that the empirical bound appears to be closest (in growth shape) to the $O\left(\frac{1}{\epsilon-\epsilon^2}\right)$ term. This is insufficient evidence to conclude that this must be "true" practical bound, however!

What we *can* conclude from these experiments is that the Achlioptas bound does *not* correspond exactly with the "true" lowest dimension bound, and that it is an approximation (albeit a useful one). With this in mind, we would like to ask whether we can get a better result

that more closely matches the true lowest dimension bound, and this is the focus of the next section.

7.3 A sometimes tighter version of Achlioptas' bound

We look closer at Achlioptas' bound, and see whether we can make any improvements to it as far as tightness goes.

Achlioptas derives the essentially tight results[2]

$$Pr(||\mathbf{v_1} - \mathbf{v_2}||^2 \geq (1+\epsilon)||\mathbf{u_1} - \mathbf{u_2}||^2) \leq \left((1+\epsilon)e^{-\epsilon}\right)^{k/2} = \left(\sqrt{1+\epsilon}e^{-\epsilon/2}\right)^k$$

$$Pr(||\mathbf{v_1} - \mathbf{v_2}||^2 \leq (1-\epsilon)||\mathbf{u_1} - \mathbf{u_2}||^2) \leq \left(1 - \frac{\epsilon}{2(1+\epsilon)} + \frac{3\epsilon^2}{8(1+\epsilon)^2}\right)^k \exp\left(\frac{\epsilon(1-\epsilon)k}{2(1+\epsilon)}\right)$$

What he then does is bound these functions with $\exp\left(-\frac{k\epsilon^2}{4} + \frac{k\epsilon^3}{6}\right)$. We see if we can get a better result than this, at least in some cases. In particular, we show the following theorem.

THEOREM 8. *If $\epsilon \in \left[\frac{3}{4}, \frac{3}{2}\right]$, then Achlioptas' statements hold for a modified k value of*[3]

$$k \geq k_0 = \frac{16 + 8\beta}{\epsilon^2} \log n$$

PROOF IDEA. We follow Achlioptas' method, but demonstrate that we can choose a different bound for the moment-generating function that behaves better for larger values of ϵ (specifically, $\epsilon \geq \frac{3}{2}$). We explain how this new bound can be easily derived, and how similar bounds can be obtained if desired. □

PROOF. First, let us convince ourselves that

$$\left(1 - \frac{\epsilon}{2(1+\epsilon)} + \frac{3\epsilon^2}{8(1+\epsilon)^2}\right) \exp\left(\frac{\epsilon(1-\epsilon)}{2(1+\epsilon)}\right) \leq \sqrt{1+\epsilon}e^{-\epsilon/2}$$

which is easy to verify using a sketch of the two functions, as provided in Figure 7.3.

What we are aiming for is a bound on both functions, and therefore we really just need to bound $\sqrt{1+\epsilon}e^{-\epsilon/2}$. Ideally, since it makes deriving an explicit bound easier, we would like a bound of the form $e^{-g(\epsilon)}$, where, to avoid a non-trivial bound, we want $g(\epsilon) \geq 0$ for all ϵ.

[2]The first result is tight, and the second one is tight up to lower order terms

[3]The restriction $\epsilon \leq \frac{3}{2}$ is done only for convenience - the true upper bound is closer to 1.6

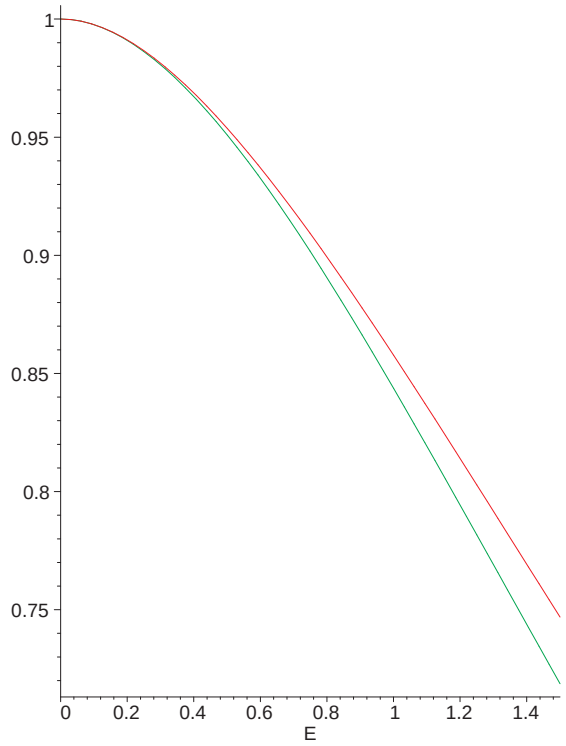


FIGURE 7.3: The graphs of the functions $y = \sqrt{1+\epsilon}e^{-\epsilon/2}$ (red) and $\left(1 - \frac{\epsilon}{2(1+\epsilon)} + \frac{3\epsilon^2}{8(1+\epsilon)^2}\right) \exp\left(\frac{\epsilon(1-\epsilon)}{2(1+\epsilon)}\right)$ (green)

Equivalently, we can derive the condition

$$\sqrt{1+\epsilon}e^{-\epsilon} = e^{\frac{1}{2}\log(1+\epsilon)-\frac{\epsilon}{2}} \leq e^{-\frac{\epsilon}{2}} \iff 0 \leq g(\epsilon) \leq \frac{\epsilon}{2} - \frac{1}{2}\log(1+\epsilon)$$

A sketch of $\frac{\epsilon}{2} - \frac{1}{2}\log(1+\epsilon)$ reveals it to be a quadratic-shaped function. In fact, we have the expansion

$$\frac{\epsilon}{2} - \frac{1}{2}\log(1+\epsilon) = \frac{\epsilon^2}{4} - \frac{\epsilon^3}{6} + O(\epsilon^4)$$

This is the motivation for Achlioptas' bound

$$\sqrt{1+\epsilon}e^{-\epsilon/2} \leq e^{-\frac{\epsilon^2}{4}+\frac{\epsilon^3}{6}}$$

as he has chosen

$$g(\epsilon) = \frac{\epsilon^2}{4} - \frac{\epsilon^3}{6}$$

However, it should be noted that the quadratic nature is such that, in the interval $\epsilon \in \left[0, \frac{3}{2}\right]$, we have

$$\frac{\epsilon^2}{8} \leq \frac{\epsilon}{2} - \frac{1}{2}\log(1+\epsilon)$$

and thus

$$\sqrt{1+\epsilon}e^{-\epsilon/2} \leq e^{-\epsilon^2/8}$$

This can be verified from a graph of the three functions, provided in Figure 7.4.

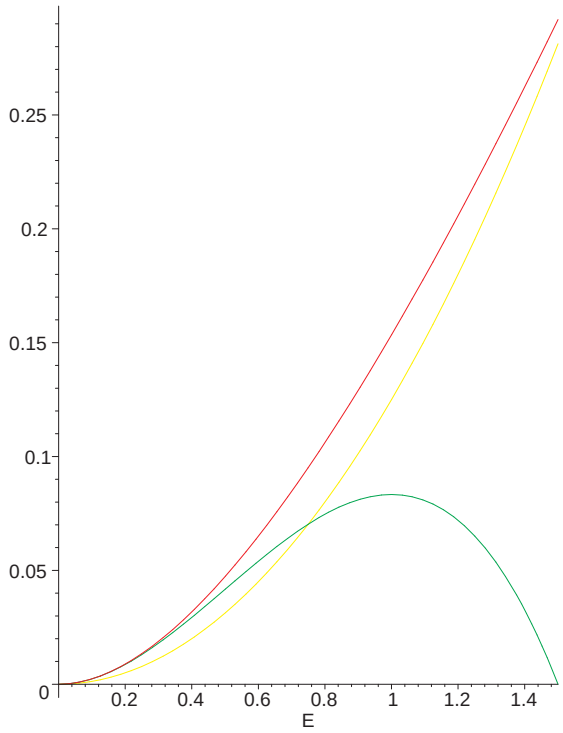


FIGURE 7.4: The graphs of the functions $y = \frac{\epsilon}{2} - \frac{1}{2}\log(1+\epsilon)$ (red), $y = \frac{\epsilon^2}{4} - \frac{\epsilon^3}{6}$ (green), and $y = \frac{\epsilon^2}{8}$ (yellow)

In fact, notice that, for $\epsilon > \frac{3}{4}$, we have[4]

$$\frac{\epsilon^2}{4} - \frac{\epsilon^3}{6} < \frac{\epsilon^2}{8}$$

and so, $\frac{\epsilon^2}{8}$ is a much tighter bound to the function $\frac{\epsilon}{2} - \frac{1}{2}\log(1+\epsilon)$.

[4]The value $\frac{3}{4}$ is the solution to $\frac{\epsilon^2}{4} - \frac{\epsilon^3}{6} = \frac{\epsilon^2}{8}$

So, to complete the proof of the theorem, we simply follow Achlioptas' steps, but note that we can bound

$$Pr(||\mathbf{v_1} - \mathbf{v_2}||^2 \geq (1+\epsilon)||\mathbf{u_1} - \mathbf{u_2}||^2) \leq e^{-k\epsilon^2/8}$$

instead of the usual $e^{-k\epsilon^2/4+k\epsilon^3/6}$. □

7.3.1 Comparison to Achlioptas' bound

It is clear that this bound is better as $\epsilon \to \frac{3}{2}$, since Achlioptas' bound blows up here (i.e. it goes to $+\infty$!), but ours only goes to a constant, calculated by $\frac{64+32\beta}{9}\log n$. Of course, for smaller ϵ, Achlioptas' bound is indeed superior. We can explicitly compute the ratio of the two, which will be

$$\frac{\text{New}}{\text{Achlioptas}} = \frac{\epsilon^2/2 - \epsilon^3/3}{\epsilon^2/4} = 2 - \frac{4\epsilon}{3}$$

In particular, then, as $\epsilon \to 0^+$, we give a result that is twice that of Achlioptas' bound, whereas as $\epsilon \to \frac{3}{2}^-$, we give results that are completely neglible with respect to his bound. One value of interest is $\epsilon = \frac{9}{8} = 1.125$, where we give a result that is half that of Achlioptas' bound.

Therefore, for $\epsilon \in \left[\frac{3}{4}, \frac{3}{2}\right]$, we can give better bounds to the lowest reduced dimension than Achlioptas. In fact, as $\epsilon \to \frac{3}{2}^-$, our improvements over his bound become arbitrarily large. Since there are e.g. algorithmic problems for which $\epsilon \approx \frac{3}{4}$ is considered "reasonable", this a bound that is certainly of some practical interest.

7.3.2 Generic tightness of Achlioptas' bound

We should make a quick note that we can't hope to do better than Achlioptas near $\epsilon = 0$, because of the result

$$\frac{\epsilon}{2} - \frac{1}{2}\log(1+\epsilon) = \frac{\epsilon^2}{4} - \frac{\epsilon^3}{6} + O(\epsilon^4)$$

that guarantees that, as $\epsilon \to 0^+$, we have the tight result (up to lower order terms)

$$\frac{\epsilon}{2} - \frac{1}{2}\log(1+\epsilon) \geq \frac{\epsilon^2}{4} - \frac{\epsilon^3}{6}$$

This in fact means that we cannot derive a better polynomial result for small ϵ, at least if we use the standard method of finding concentration bounds. The reason for this is that Achlioptas' approximations are

- The (tight) series expansion of the MGF of $||\mathbf{v_1} - \mathbf{v_2}||^2$
- The use of $\exp\left(-\frac{\epsilon^2}{4} + \frac{\epsilon^3}{6}\right)$ as an upper bound

- The use of the union bound to get the probability that any pair is distorted

So, *as long as we use the union bound, we cannot do polynomially-better than Achlioptas' result in general*. To get around the union bound, we would need to have more information on the probability of two pairs getting distorted *together*; there has thus far not been any work on this, and so it seems that if the union bound is used, Achlioptas' bound is essentially tight for small ϵ (but not for larger ϵ, as we have just demonstrated above).

This should not be seen as a contradiction with §7.2, because what we are saying is that the probabilistic method of proof using Chernoff bounds cannot be improved unless we get rid of the union bound. This is not to say that we cannot do better using *any* technique, but the problem is that, to our knowledge, there are no other techniques that have been successfully employed to derive projection guarantees, except for the complicated geometric approach of Johnson and Lindenstrauss' original work (Johnson and Lindenstrauss, 1984). The complexity of this approach is such that it is unlikely that there will be any opportunity to try and significantly improve it.

7.3.3 Decreasing the threshold

It is natural to ask whether we can decrease the threshold value of $\epsilon \geq \frac{3}{4}$, for instance to something like $\frac{1}{2}$. The answer to this turns out to be "Yes, but not without a corresponding decrease in the upper threshold"!

Consider first that the intersection of $\frac{\epsilon^2}{4} - \frac{\epsilon^3}{6}$ and $\frac{\epsilon^2}{a}$ is

$$\begin{aligned} \frac{1}{4} - \frac{\epsilon}{6} &= \frac{1}{a} \\ \implies \frac{3-2\epsilon}{12} &= \frac{1}{a} \\ \implies a &= \frac{12}{3-2\epsilon} \end{aligned}$$

So, if we wanted $\epsilon_0 = \frac{1}{2}$, then our a would be 6. In particular, by making ϵ smaller, we are making the value of a larger.

But note that we also have the upper threshold given by the solution to

$$\frac{\epsilon^2}{a} = \frac{\epsilon}{2} - \frac{1}{2}\log(1+\epsilon)$$

It is evident from Figure 7.4 that if we increase the value of a, the value of this upper threshold must then decrease. It is not easy to give an analytic solution for the threshold, but

in particular for the case where $a = 6$, the solution is roughly $\epsilon = 0.8$, which means that, for $\epsilon \in \left[\frac{1}{2}, \frac{4}{5}\right]$, we are able to do slightly better than Achlioptas' bounds. As we keep decreasing the lower threshold, however, we will bring down the upper threshold too, meaning our results become better only on an increasingly small interval!

Therefore, it is possible to do better than Achlioptas with a better lower threshold, but at the cost of a lower upper threshold as well. This means that very quickly, the results become better on a very small interval, with minimal savings.

CHAPTER 8

Distribution-based dimension bounds

We mentioned in §2.3 that Achlioptas (Achlioptas, 2001) provided an explicit relationship between the error of the projection and the reduced dimension, given by

$$k \geq k_0 := \frac{4 + 2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n$$

As discussed in §7.2, these bounds are quite weak in practise, but at the same time, it was argued in §7.3.2 that one probably cannot improve the details of his proof without losing some generality. In §7.3, we removed one aspect of generality by restricting the error parameter ϵ, and saw that we could get sharper bounds in this case.

In this chapter, we remove a different aspect of generality and see whether it can help us derive tighter bounds. Specifically, we look at data that comes from some known distribution, and see whether it can help improve the dimension bounds. Intuitively, if we can say something specific about the input data, it seems reasonable to suggest that we might get better bounds on the reduced dimension.

The contributions made in this chapter are:

- A simple extension of one of Achlioptas' results relating to the moment-generating function (MGF) of the projection
- An alternate, somewhat simpler derivation of Achlioptas' theorem
- An analysis of why e.g. normal input data does not easily yield tighter bounds than Achlioptas' general result

8.1 A new MGF bound

Recall that with a random projection, we have a guarantee on pairwise distances being preserved upto a factor of ϵ. The probability that our projection causes a distortion for the first pair is therefore simply

$$Pr(||\mathbf{v_1} - \mathbf{v_2}||^2 \geq (1+\epsilon)2\sigma^2 d) + Pr(||\mathbf{v_1} - \mathbf{v_2}||^2 \leq (1-\epsilon)2\sigma^2 d)$$

To analyze this probability, we follow the standard technique of using Markov's inequality to the moment-generating function, as was detailed in §3.3:

$$Pr(||\mathbf{v_1} - \mathbf{v_2}||^2 \geq (1+\epsilon)2\sigma^2 d) = Pr(k||\mathbf{v_1} - \mathbf{v_2}||^2 \geq (1+\epsilon)2k\sigma^2 d)$$
$$\leq \frac{M_{k||\mathbf{v_1}-\mathbf{v_2}||^2}(t)}{e^{2tk\sigma^2 d(1+\epsilon)}}$$

where $M_X(t)$ denotes the moment-generating function of the random variable X.

As with §3.3, we have

$$M_{k||\mathbf{v_1}-\mathbf{v_2}||^2} = (M_{\alpha^2})^k$$

where

$$\alpha := \sum_j (u_{1j} - u_{2j}) r_{ji}$$

So, the key is to be able to compute M_{α^2}. Consider that

$$M_{\alpha^2}(t) = E(e^{t\alpha^2}) = \sum_{n=0}^{\infty} \frac{t^n}{n!} E(\alpha^{2n})$$

Therefore, equivalently, we would like to compute, or at least bound, $E(\alpha^{2n})$. To do this, we have the following (fairly) tight theorem, which is useful when we want to study the case where the input data comes from a certain distribution.

THEOREM 9. *When the entries of R come from the Achlioptas matrix, we have that*

$$E(\alpha^{2n}) \leq E\left[(T_1 w_1 + \ldots T_d w_d)^{2n}\right]$$

where $T_i \sim \mathcal{N}(0,1), w_i := u_{1i} - u_{2i}$.

This result is similar to one provided by Achlioptas, except that we explicitly incorporate the input vector in our results, and our analysis is simpler from the use of the multinomial theorem. In order to prove this result, we first need the following simple lemma. Achlioptas states this lemma without proof, and we show here that it is very simple to prove.

LEMMA 14. *If X comes from the Achlioptas distribution, and* $Y \sim \mathcal{N}(0,1)$, *then*

$$E(X^n) \leq E(Y^n)$$

PROOF. Certainly when n is odd, the two terms are equal, since by Lemma 3 and Equation 2.19, the moments are both 0. For n even, we have

$$E(X^{2n}) = (\sqrt{3})^{2n-2} = 3^{n-1}$$

$$E(Y^{2n}) = \frac{2^n \Gamma\left(n + \frac{1}{2}\right)}{\sqrt{\pi}}$$

So, we need to prove

$$3^{n-1} \leq \frac{2^n \Gamma\left(n + \frac{1}{2}\right)}{\sqrt{\pi}}$$

To do this, we can use the duplication formula (Equation 2.20), which says we equivalently need to prove

$$3^{n-1} \leq 2^{1-n} \frac{\Gamma(2n)}{\Gamma(n)} = 2^{1-n} \frac{(2n-1)!}{(n-1)!}$$

This is true iff

$$6^{n-1} \leq \frac{(2n-1)!}{(n-1)!}$$

We can prove this by induction. Certainly it is true for $n = 1$, since $1 \leq 1$. So, assume that it is true for $n = k$, that is,

$$6^{k-1} \leq \frac{(2k-1)!}{(k-1)!}$$

So, we want to prove that

$$6^k \leq \frac{(2k+1)!}{k!}$$

To do this, note that

$$\begin{aligned}
6^k &= 6 \times 6^{k-1} \\
&\leq 6 \times \frac{(2k-1)!}{(k-1)!} \\
&= 6 \times \frac{(2k+1)!}{(2k+1)(2k)} \times \frac{k}{k!} \\
&= \frac{(2k+1)!}{k!} \frac{3}{(2k+1)} \\
&\leq \frac{(2k+1)!}{k!}
\end{aligned}$$

as required. □

With this result in place, we can now show Theorem 9.

PROOF IDEA. We use a multinomial expansion of α^{2n}, and tightly bound the moments of the Achlioptas matrix. This simplifies to the given expression. □

PROOF OF THEOREM 9. By definition, we have

$$\alpha := \sum_{i=1}^{d} w_i r_{i1}$$

where $w_i = u_{1i} - u_{2i}$.

So, we may expand $E(\alpha^{2n})$ using the multinomial theorem as follows:

$$\begin{aligned}
E(\alpha^{2n}) &= E\left((w_1 r_{11} + \ldots + w_d r_{d1})^{2n}\right) \\
&= E\left(\sum_{k_1+\ldots+k_d=2n} \binom{2n}{k_1, k_2, \ldots k_d} w_1^{k_1} r_{11}^{k_1} \ldots w_d^{k_d} r_{d1}^{k_d}\right) \\
&= \sum_{k_1+\ldots+k_d=2n} \binom{2n}{k_1, k_2, \ldots k_d} E(w_1^{k_1}) E(r_{11}^{k_1}) \ldots E(w_d^{k_d}) E(r_{d1}^{k_d}) \text{ by linearity and independence}
\end{aligned}$$

Now we use Lemma 14 to argue that

$$E(r_i^{k_i}) \leq E(T_i^{k_i})$$

where $T_i \sim \mathcal{N}(0, 1)$.

This will give us

$$\begin{aligned}
E(\alpha^{2n}) &\leq \sum_{k_1+\ldots+k_d=2n} \binom{2n}{k_1, k_2, \ldots k_d} E(w_1^{k_1}) E(T_1^{k_1}) \ldots E(w_d^{2k_d}) E(T_d^{k_d}) \\
&= E\left[(T_1 w_1 + \ldots T_d w_d)^{2n}\right] \text{ by linearity}
\end{aligned}$$

This completes the proof. □

This bound on the moment-generating function is useful because it explicitly incorporates the input vector. This means that we can try to use information about the input when it comes from a distribution to get a tight bound.

We also did some work of independent interest relating to input distributions and the variance of error. In particular, we proved the following novel theorems. Since we did not study them further, we leave the proofs in the appendix.

THEOREM 10. *Even when the input comes from a distribution, if we use a projection matrix whose entries have mean zero and variance one, the distances in the original and projected spaces must agree.*

THEOREM 11. *When the input data comes from some distribution, and we use the Achlioptas projection matrix, then,*

$$Var(||\mathbf{v_1} - \mathbf{v_2}||^2) = \frac{3\sum_i Var(w_i^2) + 2(E(||\mathbf{w}||^2))^2}{k}$$

8.2 Alternative derivation of Achlioptas' result

Theorem 9 is useful because since it explicitly deals with the input vector, it lets us apply the analysis not only to the case where the input comes from a certain distribution, but also for the standard case where the input does not follow any distribution. In fact, using Theorem 9, we can derive Achlioptas' result in a different way. As with the previous section, we get a slightly different result to Achlioptas because we explicitly incorporate the input vector in our analysis. What we wish to show here is how we can simplify the anaysis of Achlioptas somewhat; the result itself is unchanged from his own theorem.

To be precise, we show the following.

THEOREM 12. *If our random projection matrix R has entries following the Achlioptas distribution, then, as with the case where the entries come from a normal distribution we have*

$$E(\alpha^{2n}) \leq ||\mathbf{w}||^{2n} E(T^{2n})$$

where $T \sim \mathcal{N}(0,1)$. *As a result,*

$$M_{\alpha^2}(t) \leq \frac{1}{\sqrt{1-2t||\mathbf{w}||^2}}$$

PROOF IDEA. Simply use Theorem 9 and the 2-stability of the Gaussian distribution to simply the expression provided there. □

PROOF. From Theorem 9, we know that

$$\begin{aligned} E(\alpha^{2n}) &\leq E\left[(w_1T_1 + \ldots + w_dT_d)^{2n}\right] \\ &= E\left[(||\mathbf{w}||T)^{2n}\right] \text{ , } T \sim \mathcal{N}(0,1) \text{ by 2-stability 2.16} \\ &= ||\mathbf{w}||^{2n} E(T^{2n}) \text{ by Equation 2.15} \end{aligned}$$

So,

$$M_{\alpha^2}(t) \leq \sum_{n=0}^{\infty} \frac{t^n}{n!} \times ||\mathbf{w}||^{2n} E(T^{2n}) = \frac{1}{\sqrt{1-2t||\mathbf{w}||^2}}$$

□

Note that with this moment-generating function in place, we can simply use the analysis of §3.3 to derive Achlioptas' theorem.

CLAIM. *Using Theorem 12, we can deduce the exact same embedding theorem as Theorem 4.*

Our derivation is much simpler than Achlioptas' since we do not have to consider "worst-case" vectors and the like, but instead bundle the norm of the vector explicitly in the moment-generating function. So, we can claim the following.

8.3 Bounds for normal input

We can try to see if we can improve our reduced dimension bounds for the case of normal input. From Theorem 9, we know that

$$E(\alpha^{2n}) \leq E\left[(T_1 w_1 + \dots T_d w_d)^{2n}\right]$$

where $T_i \sim \mathcal{N}(0,1), w_i := u_{1i} - u_{2i}$.

In our case, $u_{ij} \sim \mathcal{N}(0, \sigma^2)$, and so by Equation 2.15,

$$w_i := u_{1i} - u_{2i} \sim \mathcal{N}(0, 2\sigma^2)$$

This means that

$$T_i w_i \sim \mathcal{N}(0,1) \times \mathcal{N}(0, 2\sigma^2)$$

So, when we run into the problem of having to find the moments of the sum of the product of two normal variables: by Equation 2.8, we have

$$E(\alpha^{2n}) \leq E\left[(T_1 w_1 + \dots T_d w_d)^{2n}\right] = M_{(\sum T_i w_i)^{2n}}(0)$$

where M^n denotes the nth derivative of $M(t)$.

This variable, unfortunately, does not readily yield its moments. It is known that its probability density function of the product of two normals is is

$$f(x) = \frac{K_0\left(\frac{|x|}{\sqrt{2}\sigma}\right)}{\sqrt{2}\sigma\pi}$$

where K_0 is a modified Bessel function of the second kind. We can try to get the moment-generating function for the product of two normals using

$$M_X(t) = E(e^{tX}) = \int_{-\infty}^{-\infty} e^{tx} f(x)$$

$$= \frac{1}{\sqrt{1-2\sigma^2 t^2}} \text{ via MAPLE}$$

Therefore, by Equation 2.9, we have that

$$M_{\sum T_i w_i} = \left(\frac{1}{\sqrt{1-2\sigma^2 t^2}}\right)^d$$

Unfortunately, this is not easy to try and invert - what we would like is to get the probability density function of this, and use that to derive a tight bound on the moments of α. Therefore, we see that the reason that Achlioptas' theorem is so simple to derive is because of the fact that we can easily study the mixture of a Gaussian and a discrete random variable, but studying the product of normal variables is much more complicated. It should be noted, however, that future work in this area might involve apprxoimate, series-based analysis of the above function, which might lead to some sort of tighter bounds for normal input.

8.4 Other bounds

Even though the normal data case does not easily yield new answers, we can look at another type of data that occurs frequently in practise, namely, sparse data. This is the focus of the next chapter, where we develop a model of sparsity, and use it to make some basic statements about projections and how they are affected by sparse data.

CHAPTER 9

Sparsity

A lot of high-dimensional data is in practise sparse, especially when the data refers to the relationship between two objects. We might attribute this to the fact that there are many applications that deal with high-dimensional data that represents in some sense the "neighbourhood" of a point. With this interpretation, sparsity arises naturally because most points are sufficiently far away from other points to be essentially isolated, implying a very sparse neighbourhood (the curse of dimensionality essentially says that points in high-dimensions start becoming "lost in space").

For instance, consider the *document-term matrix*, which is used in textual analysis. The matrix here has rows representing documents versus columns representing terms. If we consider an average cross-section of documents, it is reasonable to expect that there be some words that appear in only a handful of documents. As we consider more documents and more terms, we would expect that this matrix be very sparse, with many clusters of zeros.

We have already seen the sparse projection matrix of Achliopats being used for projections, and the work of (Li *et al.*, 2006), which takes the sparsity even further has been mentioned a few times already. It is interesting therefore to see how random projections fare with sparse matrices - both for the input, and for the projection matrix. This chapter attempts to give a preliminary treatment of this issue by providing a model of sparse data, and studying the effects that such data has on projections.

The contributions made in this chapter are:

- A simple yet powerful model of data-sparsity and results for standard projection quantities using this model
- Preliminary analysis on the impact that input sparsity has on the error of a projection
- Simple analysis on the effect that varying projection sparsity has on the error of a projection
- Experiments showing the nature of the effect of projection sparsity

9.1 Modelling input sparsity

To study the distortion of a distance vector from a known sparse matrix, we need a model of sparsity. We do this by considering two vectors - a "true" vector, and a sparse vector representation of it. That is, we suppose we have some vector $\mathbf{x}$, and then we make it sparse by only choosing to pick some fraction of the dimensions of it. Without loss of generality (and to make analysis simpler), let us suppose that the vector $\mathbf{x}$ does not have any zeros in it. So, this gives us a sparse vector $\mathbf{y}$ with the distribution

$$y_i = \begin{cases} x_i & \text{prob } = 1-p \\ 0 & \text{prob } = p \end{cases}$$

$\mathbf{x}$ is nothing more than a convenience vector that exists only conceptually. Notice that this is a fairly strong model of sparsity, because we are allowing $x_i \in \mathbb{R}$.

If we now consider the distance between two such sparse vectors $\mathbf{w}$ (w.l.o.g. consider the difference between the first two vectors), this must have distribution

$$w_j = \begin{cases} x_{1j} - x_{2j} & \text{prob } = (1-p)^2 \\ x_{1j} & \text{prob } = p(1-p) \\ -x_{2j} & \text{prob } = p(1-p) \\ 0 & \text{prob } = p^2 \end{cases}$$

If we look at the distribution of the component squared, it is

$$w_j^2 = \begin{cases} (x_{1j} - x_{2j})^2 & \text{prob } = (1-p)^2 \\ x_{1j}^2 & \text{prob } = p(1-p) \\ x_{2j}^2 & \text{prob } = p(1-p) \\ 0 & \text{prob } = p^2 \end{cases}$$

We can make the following observations.

THEOREM 13. *With w_j as defined above, we have*

$$E(w_j) = (1-p)(x_{1j} - x_{2j})$$
$$E(w_j^2) = (1-p)(x_{1j}^2 + x_{2j}^2) - 2x_{1j}x_{2j}(1-p)^2$$

$$Var(w_j) = p(1-p)(x_{1j}^2 + x_{2j}^2)$$

PROOF. The expectation is

$$\begin{aligned} E(w_j) &= (1-p)^2(x_{1j} - x_{2j}) + p(1-p)(x_{1j} - x_{2j}) \\ &= (x_{1j} - x_{2j})((1-p)^2 + p(1-p)) \\ &= (x_{1j} - x_{2j})(1-p) \end{aligned}$$

Also,

$$\begin{aligned} E(w_j^2) &= (1-p)^2(x_{1j} - x_{2j})^2 + p(1-p)(x_{1j}^2 + x_{2j}^2) \\ &= ((1-p)^2 + p(1-p))(x_{1j}^2 + x_{2j}^2) - 2(1-p)^2 x_{1j} x_{2j} \\ &= (1-p)(x_{1j}^2 + x_{2j}^2) - 2(1-p)^2 x_{1j} x_{2j} \end{aligned}$$

It follows by definition then that

$$\begin{aligned} Var(w_j) &= E(w_j^2) - (E(w_j))^2 \\ &= (1-p)^2(x_{1j} - x_{2j})^2 + p(1-p)(x_{1j}^2 + x_{2j}^2) - (1-p)^2(x_{1j}^2 - x_{2j}^2)^2 \\ &= p(1-p)(x_{1j}^2 + x_{2j}^2) \end{aligned}$$

□

9.1.1 Remarks

We notice that for the case $p = 0$, our results are consistent with those from the case where the input does not follow any distribution (note in particular that the variance is zero). Also, as $p \to 1$, which means the matrix gets more sparse, we see that each of the three terms go to zero, and in particular the variance of each component in the distance vector again tends to zero. This is because, in the limiting case, we deal with the zero matrix, which has no variance at all.

The variance of each component in the distance vector is at its maximum when $p = \frac{1}{2}$, which is the case where we have a 50% sparse matrix. In this case, the variance is $\frac{x_{1j}^2 + x_{2j}^2}{4}$.

9.2 Projected distance

We are interested in the preservation of distances of sparse data under random projection. It is clear that the distance between the projections $\mathbf{v_1}, \mathbf{v_2}$ of vectors $\mathbf{u_1}, \mathbf{u_2}$ can be considered to be the same as the projection of the original distance vector $\mathbf{w} = \mathbf{u_1} - \mathbf{u_2}$. That is,

$$\mathbf{v_1} - \mathbf{v_2} = \text{proj}(\mathbf{w})$$

So, as usual, we have

$$\mathbf{v_1} - \mathbf{v_2} = \frac{1}{\sqrt{k}}[\mathbf{w}.\mathbf{r_1^T} \quad \mathbf{w}.\mathbf{r_2^T} \quad \dots \quad \mathbf{w}.\mathbf{r_k^T}]$$

We can try to evaluate the error of the projection explicitly, with the help of the following theorem.

THEOREM 14. *With w_i as defined above,*

$$Var(w_i^2) = p(1-p)\left[x_{1i}^4 - 4(1-p)x_{1i}x_{2i}(x_{1i}^2 + x_{2i}^2) + 4(1-p)(2-p)x_{1i}^2x_{2i}^2 + x_{2i}^4\right]$$

PROOF. Consider that

$$Var(w_i^2) = E(w_i^4) - (E(w_i^2))^2$$

Now,

$$E(w_i^4) = (1-p)^2(x_{1i} - x_{2i})^4 + p(1-p)(x_{1i}^4 + x_{2i}^4)$$
$$E(w_i^2) = (1-p)^2(x_{1i} - x_{2i})^2 + p(1-p)(x_{1i}^2 + x_{2i}^2)$$

Notice that

$$\begin{aligned}(E(w_i^2))^2 &= (1-p)^4(x_{1i} - x_{2i})^4 + 2p(1-p)^3(x_{1i} - x_{2i})^2(x_{1i}^2 + x_{2i}^2) + p^2(1-p)^2(x_{1i}^2 + x_{2i}^2)^2 \\ &= (1-p)^4(x_{1i} - x_{2i})^4 + p^2(1-p)^2(x_{1i}^4 + x_{2i}^4) \\ &+ 2p(1-p)^3(x_{1i} - x_{2i})^2(x_{1i}^2 + x_{2i}^2) + 2p^2(1-p)^2x_{1i}^2x_{2i}^2\end{aligned}$$

Therefore,

$$\begin{aligned}Var(w_i^2) &= E(w_i^4) - (E(w_i^2))^2 \\ &= (1-p)^2\left[1 - (1-p)^2\right](x_{1i} - x_{2i})^4 + p(1-p)(x_{1i}^4 + x_{2i}^4) \\ &- 2p(1-p)^3(x_{1i} - x_{2i})^2(x_{1i}^2 + x_{2i}^2) - p^2(1-p)^2(x_{1i}^2 + x_{2i}^2) \\ &= p(1-p)\left[x_{1i}^4 - 4(1-p)x_{1i}x_{2i}(x_{1i}^2 + x_{2i}^2) + 4(1-p)(2-p)x_{1i}^2x_{2i}^2 + x_{2i}^4\right] \textit{ via } \text{MAPLE}\end{aligned}$$

□

9.2.1 Maximizing the variance

Suppose that $p \to 0$ - this means that the matrix becomes very dense, as there are no zero terms. In this case, the equations collapse to give us the usual non-sparse results, namely

$$Var(w_i^2) = 0$$

$$E(w_i^2) = (x_{1i} - x_{2i})^2 = w_i^2$$
$$Var(||\mathbf{v_1} - \mathbf{v_2}||^2) = \frac{2||\mathbf{w}||^4}{k}$$

Suppose that $p \to 1$ - this means that the matrix becomes very sparse, with very little non-zero terms. Here, we get

$$Var(||\mathbf{v_1} - \mathbf{v_2}||^2) = 0$$

What we can conclude from this is that the cases of very sparse or very dense vectors do not cause the variance of distortion to be suddenly increased, because as we have seen here, very dense vectors are (naturally) "normal" input that does not follow a distribution, and very sparse vectors are practically the zero vector.

It is certainly true, of course, that if we choose say $p = 1 - 10^{-5}$, then we will have a non-zero variance of w_i^2; further, it is possible that we might be able to choose x_{1i}, x_{2i} large enough such that the variance gets very large. However, in this case, we will also have $(E(w_i^2))^2$ as being large, and therefore the error ratio

$$Var\left(\frac{||\mathbf{v_1} - \mathbf{v_2}||^2}{E(||\mathbf{u_1} - \mathbf{u_2}||^2)}\right)$$

would be expected to be small.

9.2.1.1 Experimental maximization

It seems intuitive that we maximize the variance for $p \approx \frac{1}{2}$. In fact, by using specific values for x_{1i}, x_{2i} (and noting the symmetry in the two terms, and the scaling invariance of the relative terms), we can observe experimentally at least that $p \approx \frac{1}{2}$ is usually the maxima for the variance.

In fact, among the maximas, the maximum p values occurs around the case $x_{1i} = 1, x_{2i} = 2$. Unfortunately, we were not able to conduct an analysis of the behaviour for the general case,

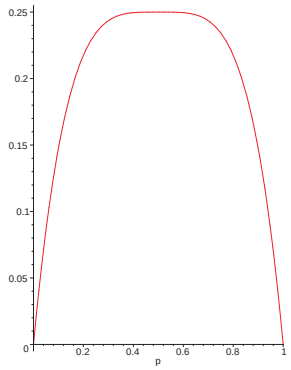


FIGURE 9.1: When $x_{1i} = x_{2i} = 1$, the variance is maximized at $p = \frac{1}{2}$

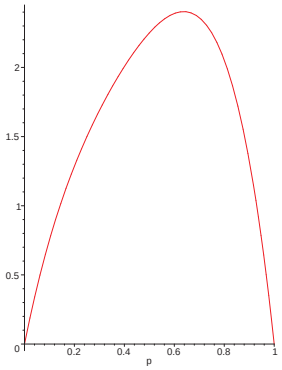


FIGURE 9.2: When $x_{1i} = 1, x_{2i} = 2$, the variance is maximized at $p \approx 0.63774$

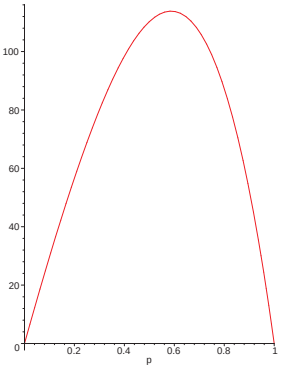


FIGURE 9.3: When $x_{1i} = 1, x_{2i} = 5$, the variance is maximized at $p \approx 0.58709$

since it requires a delicate handling of the "true" input vector. This behaviour does suggest, however, that there is a deeper pattern even for these simple cases of input sparsity.

9.3 Projection sparsity

It was mentioned in §2.5 that one of the major breakthroughs in random projections was the formulation of sparse projection matrices. Achlioptas provided the first such result with a $\frac{2}{3}$rds sparse matrix (Achlioptas, 2001), but the work was subsequently improved on by Li et al. (Li *et al.*, 2006), who proposed a matrix that was $\frac{s-1}{s}$ sparse (see §2.5), where $s \geq 3$, showing that asymptotically we get the same results as if we use a classical normal matrix.

However, Ailon and Chazelle claimed that sparsity of the projection matrix could only take us so far; they claimed that for a sparse input vector, if we used a sparse projection matrix, then we would incur a large distortion (Ailon and Chazelle, 2006). This makes some intuitive sense; suppose we have an $n \times d$ matrix that has only a fixed number, say c of non-zero entries, where $c << nd$. Then, if we have a very sparse projection matrix, it is quite likely that we will end up destroying these non-zero entries when we do our projection, replacing them with zeros. But of course "non-zero" does not imply small, and hence we could be incurring a very large distortion indeed!

9.3.1 Projection sparsity analysis

We can explicitly analyze the moment-generating function when we have a projection matrix of sparsity s. Recall that from Lemma 14, we know that if X is an Achlioptas variable, and $Y \sim \mathcal{N}(0, 1)$, then

$$E(X^n) \leq E(Y^n)$$

Suppose Z is a Li-variable with sparsity s. Then, using similar analysis to that for the Achlioptas matrix (Lemma 3), it is true that

$$E(Z^n) = \begin{cases} 1 & n = 0 \\ 0 & \text{when } n \text{ is odd} \\ (\sqrt{s})^{n-2} & \text{otherwise} \end{cases}$$

Therefore, for $n \neq 0$,

$$E(Z^{2n}) = s^{n-1} = 3^{n-1} \times \left(\frac{s}{3}\right)^{n-1}$$

This means that we can say

$$E(Z^n) \leq E(Y_0^n)$$

where $Y_0 \sim \mathcal{N}(0, \frac{s}{3})$.

If we then follow Theorem 12, it is not hard to see that we will get

$$M_{a^2}(t) \leq \frac{1}{\sqrt{1 - \frac{2ts||\mathbf{w}||^2}{3}}}$$

At our minimization step, recall that in the normal case, we minimize (see §3.3)

$$f(t) := \frac{e^{-2t(1+\epsilon)x}}{1 - 2tx}$$

Here, the extra s term means that we have to minimize

$$f(u) := \frac{e^{-2us(1+\epsilon)x}}{1 - 2ux}$$

using the substitution $u = st$.

This means that we will require a non-negative restriction on ϵ - it is not hard to see that we will in fact need

$$\epsilon > \frac{s}{3} - 1 = \frac{s-3}{3}$$

So, for example, when we go one step higher than Achlioptas to $s = 4$, then we will require that $\epsilon > \frac{1}{3}$; when $s = 9$, then we will require $\epsilon > 2$. This means that the lower bound on the error increases linearly with the projection sparsity, and so when we make the projection sparsity sufficiently high, the error becomes far too large to be useful for practical purposes.

In practise, we would not expect the error to be as large as $\frac{s-3}{3}$, due to the weakness of the union bound, but what we would expect is for the error to linearly increase as the projection sparsity increases.

9.3.2 Experiments

We ran experiments for a 100×1000 matrix with the projection sparsity parameter s ranging from 3 to 150 (in increments of 25). We also varied the reduced dimension, k, which is intimately connected with the error parameter ϵ.

The results for $k = 50$ are shown in Table 9.1. We see that there is a linear trend in the observed as the projection sparsity increases, as clearly indicated in Figure 9.4.

As the reduced dimension gets smaller, we notice that the average error starts to increase, as expected. For $k = 25$, the average error starts to increase by a fair amount (around 3 times compared to the $k = 50$ case), and we observe even more pronounced results when $k = 5$ (Table 9.3).

Projection sparsity	ϵ
3	0.0416
25	0.0495
50	0.0612
75	0.0718
100	0.0794
125	0.0977
150	0.1017

TABLE 9.1: Results for ϵ, $d = 100, k = 50$

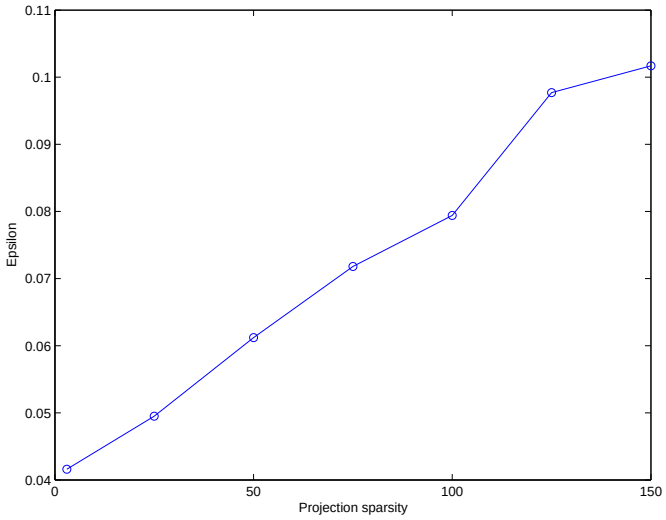


FIGURE 9.4: Varying projection sparsity when $d = 100, k = 50$

Projection sparsity	ϵ
3	0.0878
25	0.1060
50	0.1234
75	0.1516
100	0.1685
125	0.1967
150	0.2307

TABLE 9.2: Results for ϵ, $d = 100, k = 25$

So, we can see that our theoretical prediction has been validated, since as the projection sparsity increases, we observe a linear increase in the error of the projection. Further, as k decreases, we note that for a fixed sparsity, the error starts to increase as we would expect.

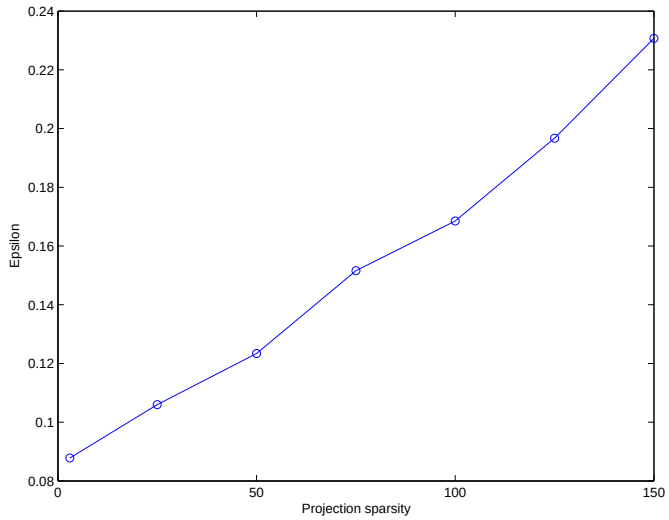


FIGURE 9.5: Varying projection sparsity when $d = 100, k = 25$

Projection sparsity	ϵ
3.0000	0.4837
25.0000	0.5115
50.0000	0.5504
75.0000	0.5886
100.0000	0.6214
125.0000	0.6460
150.0000	0.7137

TABLE 9.3: Results for ϵ, $d = 100, k = 5$

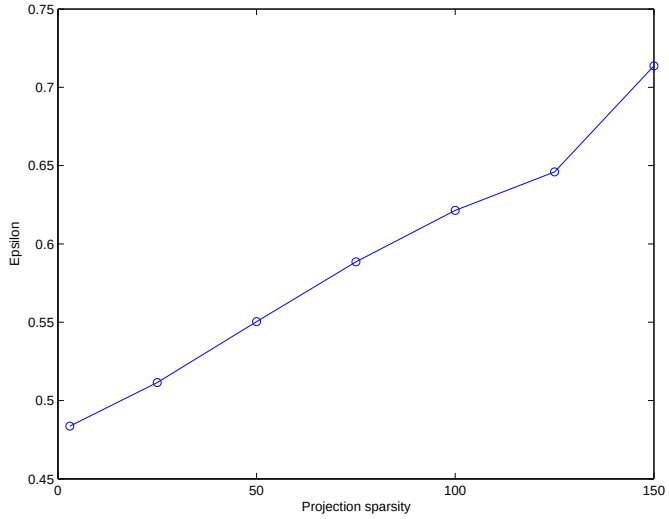


FIGURE 9.6: Varying projection sparsity when $d = 100, k = 5$

CHAPTER 10

Conclusion

10.1 Summary

Random projections are a powerful method of dimensionality reduction that provide us with both conceptual simplicity, and very strong error guarantees. The simplicity of projections allows them to be analyzed thoroughly, and this, combined with the error guarantees, makes them a very popular method of dimensionality reduction.

In this thesis, we have mainly focussed on a theoretical analysis of projections, and in particular, have looked to ask whether

(1) Projections can be applied to a data-streaming problem
(2) Theoretical results relating to dimension and error guarantees can be improved

To this end, we have shown that projections can be used to keep a sketch of a series of high-dimensional data-streams with all the known projection guarantees, and further, that this sketch improves upon an existing sketch used for the same purpose. In fact, we showed that this existing sketch is nothing more than a special case of a random projection. Motivated by the question of whether the sketches preserve dot-products, we derived a new high-probability bound for the dot-product's distortion under a random projection, and explained why this bound is tight. We also experimentally verified that the projection-based sketch is at least as accurate, and marginally quicker than the existing sketch by applying both sketches to an online clustering problem.

We then proved several theoretical results relating to projections and the lowest dimension to which one can reduce the original data. In particular, we showed that there are cases for which the existing bound is "meaningless", and derived a new bound that, for restricted error values, is much sharper than the existing bound.

We also looked at the case when the input data comes from a known distribution, and derived a new result on the standard projection guarantees relating to the error for this case. We analyzed what this says about projections and the structure of the input data.

Finally, we provided a preliminary model of input sparsity, and used this to make some basic statements about how projections behave when the input data has some known sparsity.

10.2 Future work

There are many potential areas in which work can be carried out in the future. A brief list is compiled below.

(1) Lower bound analysis for the reduced dimension k, possibly using Alon's method but under special cases, or perhaps a proof of a tight lower bound for specific input data

(2) An extended study of input distributions and their impact on the distortion of the projection, possibly tying in different ways of defining a distribution (using e.g. the moment-generating function of the distribution instead of the mean/variance)

(3) Using the results derived on input sparsity to derive an analytical solution (or at least to identify some distinct cases) for the distortion as a function of the "true" vector

(4) A detailed analysis of what happens when we combine sparse input data and sparse projection matrices, to justify a remarl of (Ailon and Chazelle, 2006), which claims "a sparse matrix will typically distort a sparse vector" - in particular, can we derive a correlation between the two, if there is one?

References

Milton Abramowitz and Irene A. Stegun. 1964. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, NY, USA, Ninth Dover printing, Tenth GPO printing edition.

Dimitris Achlioptas. 2001. Database-friendly random projections. In *PODS '01: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 274–281, New York, NY, USA. ACM Press.

Charu Aggarwal, Jiawei Han, Jianyong Wang, and Philip Yu. 2005. On high dimensional projected clustering of data streams. *Data Mining and Knowledge Discovery*, 10(3):251–273, May.

Nir Ailon and Bernard Chazelle. 2006. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. To appear in STOC '06.

Noga Alon, Yossi Matias, and Mario Szegedy. 1996. The space complexity of approximating the frequency moments. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29, New York, NY, USA. ACM Press.

Noga Alon. 2003. Problems and results in extremal combinatorics – I. *Discrete Mathematics*, 273(1):31–53.

Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. 2003. Stream: The stanford stream data manager. *IEEE Data Engineering Bulletin*, 26:19–26.

Richard Baraniuk, Mark Davenport, Ronald DeVore, and Michael Wakin. 2006. The Johnson-Lindenstrauss lemma meets Compressed Sensing. Preprint, June 12.

Richard Bellman. 1961. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, Princeton, NJ, USA.

Ella Bingham and Heikki Mannila. 2001. Random projection in dimensionality reduction: applications to image and text data. In *KDD '01: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 245–250, New York, NY, USA. ACM Press.

BoxMuller. 2006. Box-muller transform. `http://en.wikipedia.org/wiki/Box_Muller`.

Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Fred Reiss, and Mehul A. Shah. 2003. Telegraphcq: continuous dataflow processing. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 668–668, New York, NY, USA. ACM Press.

Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan. 2003. Comparing data streams using hamming norms (how to zero in). *IEEE Transactions on Knowledge and Data Engineering*, 15(3):529–540.

Sanjoy Dasgupta and Anupam Gupta. 1999. An elementary proof of the Johnson-Lindenstrauss lemma. Technical Report TR-99-006, International Computer Science Institute, Berkeley, CA, USA.

Lars Engebretsen, Piotr Indyk, and Ryan O'Donnell. 2002. Derandomized dimensionality reduction with applications. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 705–712, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Xiaoli Fern and Carla Brodley. 2003. Random projection for high dimensional data clustering: A cluster ensemble approach. In *The Twentieth International Conference on Machine Learning (ICML-2003)*, August.

Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. 2005. Mining data streams: a review. *SIGMOD Rec.*, 34(2):18–26.

Michel X. Goemans and David P. Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145.

Gene Golub and Charles Van Loan. 1983. *Matrix Computations*. John Hopkins University Press, Baltimore, Maryland, USA.

Leo Goodman. 1960. On the exact variance of products. *Journal of the American Statistical Association*, pages 708–713, December.

S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. 2000. Clustering data streams. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 359, Washington, DC, USA. IEEE Computer Society.

R. Hecht-Nielsen. 1994. Context vectors: general purpose approximate meaning representatins self-organized from raw data. *Computational Intelligence: Imitating Life*, pages 43–56.

Alexander Hinneburg, Charu C. Aggarwal, and Daniel A. Keim. 2000. What is the nearest neighbor in high dimensional spaces? In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 506–515, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Piotr Indyk. 2006. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323.

W.B. Johnson and J. Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in Modern Analysis and Probability*, pages 189–206, Providence, RI, USA. American Mathematical Society.

Ian T. Jolliffe. 1986. *Principal Component Analysis*. Springer-Verlag, New York, NY, USA.

Ping Li, Trevor J. Hastie, and Kenneth W. Church. 2006. Very sparse random projections. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 287–296, New York, NY, USA. ACM Press.

Jessica Lin and Dimitrios Gunopulos. 2003. Dimensionality reduction by random projection and latent semantic indexing. In *Proceedings of the Text Mining Workshop, at the 3rd SIAM International Conference on Data Mining*.

J. B. MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Symposium on Math, Statistics, and Probability*, volume 1, pages 281–297.

George Marsaglia and Wai Wan Tsang. 2000. The ziggurat method for generating random variables. *Journal of Statistical Software*, 5(8):1–7.

Michael Mitzenmacher and Eli Upfal. 2005. *Probability and computing*. Cambridge. Mitzenmacher.

Cleve Moler. 2001. Normal behavior. `http://www.mathworks.com/company/newsletters/news_notes/clevescorner/spring01_cleve.html`. Accessed September 28, 2006.

S. Muthukrishnan. 2005. *Data Streams: Algorithms And Applications*. Now Publishers.

Christos H. Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. 1998. Latent semantic indexing: a probabilistic analysis. In *PODS '98: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 159–168, New York, NY, USA. ACM Press.

J. Shawe-Taylor and N. Christianini. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, UK.

Santosh S. Vempala. 2004. *The Random Projection Method*, volume 65 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Providence, RI, USA.

A. C-C. Yao. 1977. Probabilistic computations: Toward a unified measure of complexity. In *Proc. of 18th IEEE FOCS*, pages 222–227.

APPENDIX A

Appendices

A.1 Miscellaneous proofs

We include the proofs of miscellaneous results here.

PROOF OF LEMMA 1. Firstly, consider the definition of the dot-product of two vectors:

$$\mathbf{x_i}.\mathbf{r} = ||\mathbf{x_i}||||\mathbf{r}|| \cos\theta_{x_i r}$$

In our case, $||\mathbf{x_i}||^2 = ||\mathbf{x_j}||^2 = ||\mathbf{r}||^2 = 1$. So,

$$\mathbf{x_i}.\mathbf{r} = \cos\theta_{x_i r}$$

$$\mathbf{x_j}.\mathbf{r} = \cos\theta_{x_j r}$$

So, the sign of the dot-products depends on the sign of the cosine of the angle between the vectors. Suppose we measure our angles in the interval $[-\pi, \pi]$. Then, it is certainly true that

$$\text{sgn}(\cos\theta) = \begin{cases} +1, & \theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \\ -1 & \theta < -\frac{\pi}{2}, \theta > \frac{\pi}{2} \end{cases}$$

This means that the dot-product $\mathbf{x_i}.\mathbf{r}$ say is positive when the vector $\mathbf{r}$ lies inside the region spanned by the line perpendicular to $\mathbf{x_i}$. To aid intuition, we look at the problem in the case of $\mathbb{R}^2$. The result easily generalizes to higher-dimensions.

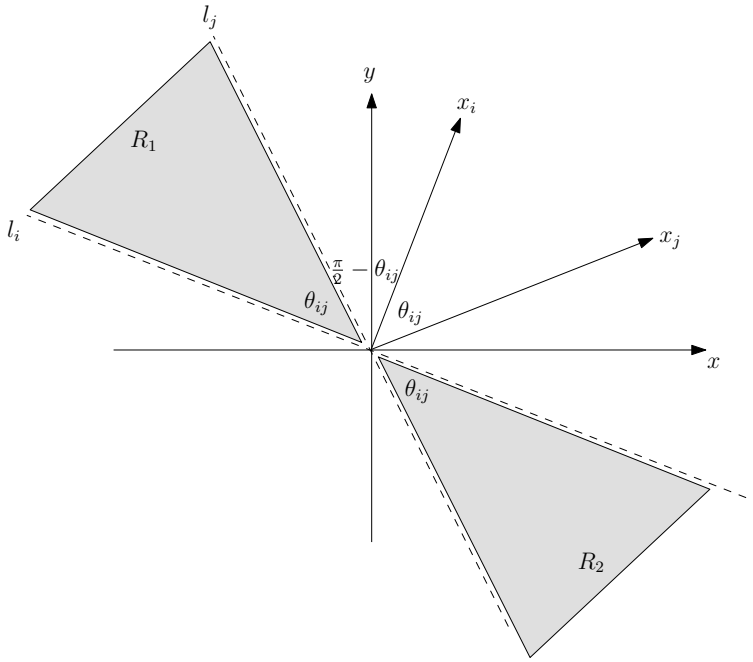
Here, l_i, l_j are lines drawn perpendicular to $\mathbf{x_i}, \mathbf{x_j}$ respectively. This means that the dot-products will have a different sign when the vector $\mathbf{r}$ lies inside the 'positive' region for one vector, but not for the other. Clearly, there are only two such regions, the ones marked R_1, R_2.

Using the fact that l_i, l_j are perpendicular to $\mathbf{x_i}, \mathbf{x_j}$, we get that the total angle marked out by the regions R_1, R_2 is $2\theta_{ij}$. Therefore, the probability of a different sign is

$$Pr(\text{sign changed}) = \frac{2\theta_{ij}}{2\pi} = \frac{\theta_{ij}}{\pi}$$

The expected number of edges in the cut is

$$\sum_{ij \in E} P(ij \text{ is cut}) = \sum_{ij \in E} \frac{\theta_{ij}}{\pi}$$



How bad is this approximation? We have that

$$\begin{aligned} c &= \frac{\sum \frac{\theta_{ij}}{\pi}}{\sum \frac{1-x_{ij}}{2}} \\ &= \frac{\sum \frac{\theta_{ij}}{\pi}}{\sum \frac{1-\cos\theta_{ij}}{2}} \\ &= \frac{2}{\pi} \frac{\sum \theta_{ij}}{\sum 1-\cos\theta_{ij}} \\ &\geq \min_{\theta} \frac{2\theta}{\pi(1-\cos\theta)} \\ &> 0.87856 \end{aligned}$$

Therefore, the solution we get is within 15% of the optimal solution, which is fairly accurate. □

PROOF OF THEOREM 7. Recall that[1]

$$\begin{aligned} (1-\epsilon)\text{median}_j(v_{ij}^2) &\leq \text{median}(X^2)||\mathbf{u_i}||^2 \\ &\leq (1+\epsilon)\text{median}_j(v_{ij}^2) \end{aligned}$$

[1]This theorem, and the proof, is due to Anh Pham, and is included for completeness' sake only

Let us call the estimate of the norm of $\mathbf{u}$, denoted by $e(||\mathbf{u}||^2)$, the following:

$$e(||\mathbf{u}||^2) := \frac{\text{median}_j(v_j^2)}{\text{median}(X^2)}$$

where $\mathbf{v}$ is the 'projection' of $\mathbf{u}$, which makes the above equation

$$(1-\epsilon)e(||\mathbf{u_i}||^2) \leq ||\mathbf{u_i}||^2 \leq (1+\epsilon)e(||\mathbf{u_i}||^2)$$

Now, to estimate dot-products, note that we can write the dot-product using norms:

$$\mathbf{x}.\mathbf{y} = \frac{||\mathbf{x}+\mathbf{y}||^2 - ||\mathbf{x}-\mathbf{y}||^2}{4}$$

This motivates an initial estimate for the dot-product,

$$\hat{e}(\mathbf{u_i}.\mathbf{u_j}) = \frac{e(||\mathbf{u_i}+\mathbf{u_j}||^2) - e(||\mathbf{u_i}-\mathbf{u_j}||^2)}{4}$$

So, we can bound the error on our estimate $e(\mathbf{u_i}.\mathbf{u_j})$:

$$\begin{aligned}
\hat{e}(\mathbf{u_i}.\mathbf{u_j}) &\leq \frac{1}{4}\left(\frac{||\mathbf{u_i}+\mathbf{u_j}||^2}{(1-\epsilon)} - \frac{||\mathbf{u_i}-\mathbf{u_j}||^2}{(1+\epsilon)}\right) \\
&= \frac{1}{4}\left(\frac{2\epsilon(||\mathbf{u_i}||^2+||\mathbf{u_j}||^2)}{(1-\epsilon^2)} + \frac{4\mathbf{u_i}.\mathbf{u_j}}{(1-\epsilon^2)}\right) \\
&= \frac{\epsilon}{(1-\epsilon^2)}\frac{||\mathbf{u_i}||^2+||\mathbf{u_j}||^2}{2} + \frac{\mathbf{u_i}.\mathbf{u_j}}{(1-\epsilon^2)}
\end{aligned}$$

A similar lower bound exists

$$\hat{e}(\mathbf{u_i}.\mathbf{u_j}) \geq -\frac{\epsilon}{(1-\epsilon^2)}\frac{||\mathbf{u_i}||^2+||\mathbf{u_j}||^2}{2} + \frac{\mathbf{u_i}.\mathbf{u_j}}{(1-\epsilon^2)}$$

Now define the estimate of the dot-product to be

$$e(\mathbf{u_i}.\mathbf{u_j}) := (1-\epsilon^2)\frac{\text{median}_l(v_{il}v_{jl})}{\text{median}(X^2)}$$

where $X \sim \mathcal{N}(0,1)$. This is clearly just $(1-\epsilon^2)\hat{e}$. We observe that we manage to get the same error bound as with projections:

$$|\mathbf{u_i}.\mathbf{u_j} - e(\mathbf{u_i}.\mathbf{u_j})| \leq \frac{\epsilon}{2}(||\mathbf{u_i}||^2+||\mathbf{u_j}||^2)$$

□

PROOF OF THEOREM 10. To show this, we follow Achlioptas' approach for projection, and we write the projection as $E = \frac{1}{\sqrt{k}} A.R$. Following the approach of §3.2, we write

$$\begin{aligned} k||\mathbf{v_1} - \mathbf{v_2}||^2 &= \sum_{i=1}^{k} \left(\sum_{j=1}^{d} w_j.r_{ji} \right)^2 \\ &= \sum_{i=1}^{k} \alpha^2 \end{aligned}$$

where, by independence w.r.t. i, we take

$$\begin{aligned} \alpha &:= \sum_j w_j r_{j1} \\ w_j &:= u_{1j} - u_{2j} \end{aligned}$$

Thus,

$$\begin{aligned} E(k||\mathbf{v_1} - \mathbf{v_2}||^2) &= \sum_{i=1}^{k} E(\alpha^2) \\ &= \sum_{i=1}^{k} Var(\alpha) + (E(\alpha))^2 \end{aligned}$$

so that by independence of the summand,

$$E(||\mathbf{v_1} - \mathbf{v_2}||^2) = Var(\alpha) + (E(\alpha))^2$$

With α as defined above,

$$\begin{aligned} E(\alpha) &= E\left(\sum_{j=1}^{d} w_j.r_{j1} \right) \\ &= \sum_{j=1}^{d} E\left(w_j.r_{j1} \right) \text{ by linearity} \\ &= \sum_{j=1}^{d} E(w_j) E(r_{j1}) \text{ by independence} \\ &= 0 \end{aligned}$$

Also,

$$Var(\alpha) = Var\left(\sum_{j=1}^{d} w_j.r_{j1} \right)$$

$$= \sum_{j=1}^{d} Var\left(w_j.r_{j1}\right) \text{ by independence}$$

Now, the exact formula for the variance of the product of two independent variables due to (Goodman, 1960) is:

$$Var(XY) = (E(X))^2 Var(Y) + (E(Y))^2 Var(X) + Var(X)Var(Y)$$

Take $X = w_j$ and $Y = r_{ji}$. Then,

$$E(Y) = 0, Var(Y) = 1$$

So,

$$\begin{aligned} Var(XY) &= (E(X))^2 + Var(X) \\ &= E(X^2) \end{aligned}$$

and as a result

$$Var(\alpha) = \sum_i E(w_i^2) = E(||\mathbf{w}||^2)$$

Using these results, we can see that

$$E(||\mathbf{v_1} - \mathbf{v_2}||^2) = E(||\mathbf{w}||^2)$$

That is, distances in the original and projected spaces will, on average, agree. □

PROOF OF THEOREM 11. We see that

$$\begin{aligned} Var(k||\mathbf{v_1} - \mathbf{v_2}||^2) &= \sum_{i=1}^{k} Var(\alpha_i^2) \text{ by independence of each } \alpha_i^2 \\ &= \sum E(\alpha_i^4) - (E(\alpha_i^2))^2 \end{aligned}$$

Again by independence, and by Equation 2.7,

$$Var(||\mathbf{v_1} - \mathbf{v_2}||^2) = \frac{E(\alpha_i^4) - (E(\alpha_i^2))^2}{k}$$

We already know the value of $E(\alpha_i^2)$, so we just have to calculate $E(\alpha_i^4)$.

Now, every α_i will have the same distribution α. We have

$$\begin{aligned}\alpha^4 &= (w_1 r_{1i} + w_2 r_{2i} + \ldots + w_d r_{di})^4 \\ &= \sum \binom{4}{k_1, \ldots, k_d} w_1^{k_1} r_{1i}^{k_1} \ldots w_d^{k_d} r_{di}^{k_d} \text{ where } \sum k_i = 4\end{aligned}$$

There are only five possible assignments, or permutations thereof, for the k_i's:

- $\{4, 0, 0, \ldots, 0\}$, with d possible permutations
- $\{3, 1, 0, \ldots, 0\}$, with $d(d-1)$ possible permutations
- $\{2, 2, 0, \ldots, 0\}$, with $d(d-1)/2$ possible permutations
- $\{1, 1, 2, 0, \ldots, 0\}$, with $d(d-1)(d-2)/2$ possible permutations
- $\{1, 1, 1, 1, 0 \ldots, 0\}$, with $d(d-1)(d-2)(d-3)/4!$ possible permutations

Note that the total number of solutions to the equation

$$k_1 + k_2 + \ldots k_d = 4, k_i \geq 0$$

is $\binom{4+d-1}{4}$, which can be shown to be the sum of the above expressions for the number of permutations.

We can denote α^4 as

$$\alpha^4 = \Sigma_1 + \Sigma_2 + \ldots + \Sigma_5$$

where $\sum_i$ corresponds to the terms falling under the ith permutation. We can evaluate each of these in turn. Firstly, recall from Lemma 3 that

$$E(r_i^n) = \begin{cases} (\sqrt{3})^{n-2} & n > 0 \text{ even} \\ 0 & n > 0 \text{ odd} \\ 1 & n = 0 \end{cases}$$

This clearly means that if *any* k_i is odd, the whole sum will be zero, because the $E(r_i^{k_i})$ will then be zero. This means that every k_i must be even, and so this leaves only Σ_1, Σ_3, and

$$E(\Sigma_2 + \Sigma_4 + \Sigma_5) = 0$$

Consider first $E(\Sigma_1)$. Here, all k_i's are zero except one, which equals 4. If $k_i = 0$, the terms just become one, so the summand just has a single term, raised to the fourth power. So, the

value must be

$$\begin{aligned} E(\Sigma_1) &= E\left(\sum_i \binom{4}{4,0,\ldots,0} w_i^4 r_i^4\right) \\ &= \sum E(w_i^4)E(r_i^4) \\ &= 3\sum E(w_i^4) \end{aligned}$$

Now consider $E(\Sigma_3)$. Here we have two terms that are raised to the second power, and the rest are all raised to zero (and must be one). Since the two terms are distinct, we can write the expectation as

$$\begin{aligned} E(\Sigma_3) &= E\left(\sum_{i<j} \binom{4}{2,2,0,\ldots,0} w_i^2 r_i^2 w_j^2 r_j^2\right) \\ &= \sum_{i<j} 6E(w_i^2)E(r_i^2)E(w_j^2)E(r_j^2) \\ &= 6\sum_{i<j} E(w_i^2)E(w_j^2) \end{aligned}$$

So,

$$E(\alpha^4) = 3\sum E(w_i^4) + 6\sum_{i<j} E(w_i^2)E(w_j^2)$$

Certainly

$$E(\alpha^2) = Var(\alpha) + (E(\alpha))^2 = Var(\alpha) = E(||\mathbf{w}||^2)$$

And therefore

$$\begin{aligned} Var(\alpha^2) &= 3\sum_i E(w_i^4) + 6\sum_{i<j} E(w_i^2)E(w_j^2) - \left(\sum_i E(w_i^2)\right)^2 \\ &= 3\left[\sum_i E(w_i^4) + 2\sum_{i<j} E(w_i^2)E(w_j^2)\right] - \left(\sum_i E(w_i^2)\right)^2 \\ &= 3\left[\sum_i Var(w_i^2) + E(w_i^2)^2 + 2\sum_{i<j} E(w_i^2)E(w_j^2)\right] - \left(\sum_i E(w_i^2)\right)^2 \\ &= 3\left[\sum_i Var(w_i^2) + \left(\sum_i E(w_i^2)\right)^2\right] - \left(\sum_i E(w_i^2)\right)^2 \\ &= 3\sum_i Var(w_i^2) + 2(E(||\mathbf{w}||^2))^2 \end{aligned}$$

So,

$$Var(||\mathbf{v_1} - \mathbf{v_2}||^2) = \frac{3\sum_i Var(w_i^2) + 2(E(||\mathbf{w}||^2))^2}{k}$$

What we have shown is that, essentially,

$$Var\left(\frac{||\mathbf{v_1} - \mathbf{v_2}||^2}{E(||\mathbf{u_1} - \mathbf{u_2}||^2)}\right) = \frac{\frac{3\sum_i Var(w_i^2)}{(E(||\mathbf{w}||^2))^2} + 2}{k}$$

We can rewrite this as

$$\begin{aligned} Var\left(\frac{||\mathbf{v_1} - \mathbf{v_2}||^2}{E(||\mathbf{w}||^2)}\right) &= \frac{3 \times \left(\frac{\sum E(w_i^4) + 2\sum_{i<j} E(w_i^2)E(w_j^2)}{\sum E(w_i^2)^2 + 2\sum_{i<j} E(w_i^2)E(w_j^2)} - 1\right) + 2}{k} \\ &= \frac{3 \times \frac{\sum E(w_i^4) + 2\sum_{i<j} E(w_i^2)E(w_j^2)}{\sum E(w_i^2)^2 + 2\sum_{i<j} E(w_i^2)E(w_j^2)} - 1}{k} \end{aligned}$$

Notice that when the input does not come from any distribution, $Var(w_i^2) = 0$, and so we get

$$Var\left(\frac{||\mathbf{v_1} - \mathbf{v_2}||^2}{E(||\mathbf{u_1} - \mathbf{u_2}||^2)}\right) = \frac{2}{k}$$

which is the standard result of Lemma 6. What is interesting is that this means that this is the *best case input*, because we cannot go any smaller than $\frac{2}{k}$! Essentially, any distribution that has $Var(w_i^2) > 0$ will have a greater variance in the answer than this case, even if only marginally. This is a remarkable result, as it suggests that input from a distribution is actually destroyed a little more than non-distribution input - projections will still preserve distances on average, but with a slightly higher variance. □

A.2 MATLAB code

```
1 function [ clusterAllocations, similarities, centroidSums ] = compareSketchesWithClustering(or
2
3     epsilon = 0.1;
4
5     projectionScaleFactor = 1/sqrt(k);
6     L2ScaleFactor = 0.49;
7
8     n = size(originalData, 1);
9     d = size(originalData, 2);
```

```
10
11      % Sketches
12      projectionSketch = zeros(n, k);
13      L2Sketch = zeros(n, k);
14
15      myData = originalData; % Current state of data
16
17      % Updates between start and dest
18      updates = makeUpdatesBetweenPoints(originalData, destinationData);
19
20      CLUSTERINGS_WANTED = 10;
21      period = floor(size(updates, 1) / CLUSTERINGS_WANTED);
22      numClusterings = floor(size(updates, 1) / period);
23
24      % Cluster allocations
25      clusterAllocations = cell(1, 4);
26
27      % Quality measures
28      similarities = cell(1, 3);
29
30      sameClustersWithSecondTry = zeros(1, numClusterings);
31      sameClustersWithProjections = zeros(1, numClusterings);
32      sameClustersWithL2 = zeros(1, numClusterings);
33
34      centroidSums = cell(1, 3);
35      centroidSumWithSecondTry = zeros(1, numClusterings);
36      centroidSumWithProjections = zeros(1, numClusterings);
37      centroidSumWithL2 = zeros(1, numClusterings);
38
39      disp(sprintf('# of updates = %d, clusterings = %d', size(updates, 1), numClusterings))
40      disp(sprintf('%d points in %d dimensions will be reduced to %d dimensions', n, d, k))
41
42      % Counters
43      t = 1;
44      clusterings = 1;
45
46      for update = updates'
47          %tic
```

```
48          i = update(1);
49          j = update(2);
50          updVal = update(3);
51
52          myData(i, j) = myData(i, j) + updVal;
53
54          % Update the sketches
55          rand('state', j)
56          R = sparse(projectionScaleFactor * achlioptasRandomMatrix(1, k));
57          projectionSketch(i, :) = projectionSketch(i, :) + updVal * R;
58
59          randn('state', j)
60          R = randn(1, k);
61          L2Sketch(i, :) = L2Sketch(i, :) + updVal * R;
62
63          % Do clustering
64          if mod(t, period) == 0
65              %disp(sprintf('Clustering #%d...', clusterings))
66
67              [ clusterIndicators, sameClusters, centroids ] = feval(clusterStep, myData, epsilo
68
69              sameClustersWithSecondTry(clusterings) = sameClusters(1);
70              sameClustersWithProjections(clusterings) = sameClusters(2);
71              sameClustersWithL2(clusterings) = sameClusters(3);
72
73              centroidSumWithSecondTry(clusterings) = centroids(1);
74              centroidSumWithProjections(clusterings) = centroids(2);
75              centroidSumWithL2(clusterings) = centroids(3);
76
77              clusterings = clusterings + 1;
78              %disp('Clustered!')
79          end
80
81          t = t + 1;
82          %toc
83      end
84
85      clusterAllocations{1} = convertClusterIndicatorToAllocations(clusterIndicators{1});
```

```
86      clusterAllocations{2} = convertClusterIndicatorToAllocations(clusterIndicators{2});
87      clusterAllocations{3} = convertClusterIndicatorToAllocations(clusterIndicators{3});
88      clusterAllocations{4} = convertClusterIndicatorToAllocations(clusterIndicators{4});
89
90      similarities{1} = sameClustersWithSecondTry;
91      similarities{2} = sameClustersWithProjections;
92      similarities{3} = sameClustersWithL2;
93
94      centroidSums{1} = centroidSumWithSecondTry;
95      centroidSums{2} = centroidSumWithProjections;
96      centroidSums{3} = centroidSumWithL2;
```

```
1  function [ clusterIndicators, sameClusters, centroids ] = kMeansClusterStep(myData, epsilon, n
2
3              % Find the cluster allocations
4              [ normalClusterIndicator, normalCentroidDistances ] = myKMeans(myData, numClusters
5              [ normalClusterIndicatorSecondTry, normalCentroidDistancesSecondTry ] = myKMeans(m
6
7              [ projectedClusterIndicator, projectedCentroidDistances ] = myKMeans(projectionSke
8              [ L2ClusterIndicator, L2CentroidDistances ] = myKMeans(L2Sketch, numClusters, @L2P
9
10             clusterIndicators = cell(1, 4);
11             clusterIndicators{1} = normalClusterIndicator;
12             clusterIndicators{2} = normalClusterIndicatorSecondTry;
13             clusterIndicators{3} = projectedClusterIndicator;
14             clusterIndicators{4} = L2ClusterIndicator;
15
16             % Check how many pairs are in the same cluster for the projected/normal
17             % allocation
18             sameClusters = zeros(1, 3);
19             sameClusters(1) = findClusterSimilarityUsingIndicators(normalClusterIndicator, nor
20             sameClusters(2) = findClusterSimilarityUsingIndicators(projectedClusterIndicator,
21             sameClusters(3) = findClusterSimilarityUsingIndicators(L2ClusterIndicator, normalC
22
23             normalCentroidDistances = findCentroidDistances(myData, normalClusterIndicator);
24             normalCentroidDistancesSecondTry = findCentroidDistances(myData, normalClusterIndi
25             projectedCentroidDistances = findCentroidDistances(myData, projectedClusterIndicat
```

```
26              L2CentroidDistances = findCentroidDistances(myData, L2ClusterIndicator);
27
28              centroids = zeros(1, 3);
29              centroids(1) = sum(normalCentroidDistances) / sum(normalCentroidDistancesSecondTry
30              centroids(2) = sum(normalCentroidDistances) / sum(projectedCentroidDistances);
31              centroids(3) = sum(normalCentroidDistances) / sum(L2CentroidDistances);
```

```
1  % Constructs the projection of a given nxd matrix (representing n points
2  % in R^d) into a subspace R^k, where k is chosen based on the error in
3  % terms of distance, and the probability of preserving distances within the
4  % error is 1-n^-beta
5  function projection = achlioptasRandomProjection(A, error, beta)
6
7      % Number of points
8      n = size(A, 1);
9
10     % Reduced dimension size
11     % Bound provided by Achlioptas
12     k = achlioptasReducedDimension(n, error, beta);
13
14     %disp('Reducing to dimension:')
15     %disp(k)
16
17     % Original dimension of the points
18     d = size(A, 2);
19
20     % Get the standard Achlioptas random matrix
21     R = achlioptasRandomMatrix(d, k);
22
23     projection = randomProjection(A, R);
```

```
1  function R = achlioptasRandomMatrix(d, k)
2
3      % Our random d x k matrix
4      % Entries are from 0, sqrt(3), -sqrt(3) with probability {2/3, 1/6,
```

```
5       % 1/6}
6       R_temp = rand(d, k);
7       R = zeros(d, k);
8
9       R (find (R_temp > 5/6)) = sqrt(3);
10      R (find (R_temp < 1/6)) = -sqrt(3);
```

```
1  % Returns the lowest reduced dimension that we can use to project n points
2  % with at most the given error for distortion with probability ≥ 1 -
3  % n^-beta
4  function dimension = achlioptasReducedDimension(n, error, beta)
5      dimension = ceil((4 + 2*beta)*log(n) ./ ((error.^2/2) - (error.^3/3)));
```

```
1  function R = liRandomMatrix(d, k, s)
2
3      % Our random d x k matrix
4      % Entries are from 0, sqrt(s), -sqrt(s) with probability {1 - 1/s, 1/2s,
5      % 1/2s}
6      R_temp = rand(d, k);
7      R = zeros(d, k);
8
9      R (find (R_temp > 1 - 1/(2*s))) = sqrt(s);
10     R (find (R_temp < 1/(2*s))) = -sqrt(s);
```

```
1  % Constructs the projection of the given n x d matrix into a subspace R^k
2  function projection = verySparseRandomProjection(A, error, beta)
3
4      % Reduced dimension size
5      % Bound provided by Achlioptas
6      k = ceil((4 + 2*beta)*log(n) / ((error^2 / 2) - (error^3 / 3)));
7
8      % Original dimension of the points
9      d = size(A, 2);
10
```

```
11      % Our random d x k matrix
12      % Entries are from 0, sqrt(s), -sqrt(s) with probability {1-1/s, 1/2s, 1/2s},
13      % where s = sqrt(d)
14      R_temp = rand(d, k);
15      R = zeros(d, k);
16
17      R (find (R_temp ≥ 1/sqrt(d))) = 0;
18      R (find (R_temp < 1/sqrt(d) & R_temp ≥ 1/(2*sqrt(d)))) = sqrt(sqrt(d));
19      R (find (R_temp < 1/(2*sqrt(d)))) = -sqrt(sqrt(d));
20
21      projection = randomProjection(A, R);
```