Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Draft Report 2018

Project Title:	**Dimension Reduction For Big Data**
Student:	**Devin Nanayakkara**
CID:	**00932538**
Course:	**EIE4**
Project Supervisor:	**Dr Cong Ling**
Second Marker:	**Prof. A. Manikas**

Abstract

Big Data is one of the fields that have grown in popularity amongst many companies at present. The expanding technological progress has made data grow in both size and complexity, that results in inherent statistical difficulty as well as computational difficulty for a standard machine. This project implements random projectors which embed high dimensional data onto a low dimensional subspace in the Euclidean space. These are called Fast Johnson-Lindenstrauss Transforms. It considers concepts of sparse matrices, error correcting codes, and BCH codes which are used to yield the low dimensional structure given a small distortion. The critical features analysed are the computation time and the probability of success for different embedded dimensions of each algorithm. K-Nearest Neighbours Search Algorithm, a machine learning technique is used to analyse the performance and feasibility of the random projectors in practice. The primary focus of this project is to explore algorithms for dimension reduction and compare their performance and characteristics.

Acknowledgements

To be completed.

Contents

Chapter 1

Introduction

1.1 Motivation

The means of processing and interpreting information has changed drastically in the recent past. According to Big Data Analytics, Data is defined as information which is structured efficiently for better usage or processing [1]. The journey of memory and storage has achieved many promising results from the Manchester Mark 1 (Williams-Kilburn) tube in 1947 to floppy disk in 1981 and more recently cloud storage solutions in 2009 [2]. However, the rapid development in memory and storage has lead information to grow in size as well as complexity. Thus giving rise to the Big Data era. Image and video processing is one of the primary applications of big data where a dimension represents each pixel. For instance, a 900-pixel stream would require 900 dimensions and an additional dimension to represent time [3]. Genomics is a developing area of study in the big data field. According to biotechnology research, the data storage demands for genetics research would run to $2 - 40$ exabytes (1 exabyte = 10^{18} bytes) by 2025 [4]. Analytics in video surveillance for security, healthcare monitoring systems, and analytics in finance are few examples of other major big data applications.

In big data, the information is very large and complex that running traditional data processing and learning algorithms on a standard machine, becomes inadequate. Difficulty in capturing, querying, and updating data, expensive costs for storage and transmission, and too complex for analysis are few drawbacks of data being large and complex [5]. The prevailing curse of dimensionality craves for a solution in order to survive with the daily expansion of data.

1.2 Aims and Objectives

This main aim of this project is to address the computational and statistical difficulties that are caused by big data. The idea is to embed the data in a high-dimensional space to a low-dimensional subspace through random projections. The advantage of dimension reduction is that traditional data processing and learning algorithms can be applied to the data without any drawbacks caused due to high dimensionality.

The project's focus is on developing random projection techniques for dimensional reduction and subsequently tests, evaluates, and compares these methods with respect to computation time, reduced dimension, and indubitably, the correctness. In this project, the data would be part of the Euclidean space where the projection would be from a high-dimensional Euclidean space to a random low-dimensional Euclidean subspace. After preparing the data in the reduced dimension, the data would be used in an instance-based learning algorithm which would evaluate the projection technique's correctness.

Using the idea of dimension reduction, this project would examine the lowest and highest reduced dimensions a method can achieve, the behaviour of probability error for reduced dimensions, and the performance between the reduced dimensional data and high-dimensional data. The report initially provides a section on technical background covering the essential information required for the random projection techniques. Then the random projection techniques are introduced, and the architecture of each method is covered. Formally the implementation of the systems is explained followed by the testing processes which covers all aspects in order to arrive at promising results. The next section will cover the results obtained from the testing processes and based on the performance comparisons the methods would be evaluated. Thus arriving on conclusions to provide results and supporting evidence to answer the questions:

1. What is a feasible architecture for a random projection technique?
2. Will the reduced dimensional data provide promising results on the learning algorithm in comparison to the high dimensional data?
3. If so, Can reduced dimensional data be a substitute for any high dimensional data?

Chapter 2

Background

2.1 Random Projection

Random projections are proven to be a useful tool for dimension reduction in the Euclidean space by various research institutes and researchers. It uses the pairwise distance between points to manipulate large datasets into small subspaces. In addition, such projections are proven to be computationally fast [6]. The basic idea in high-dimensional to low-dimensional space is shown in Figure 2.1.

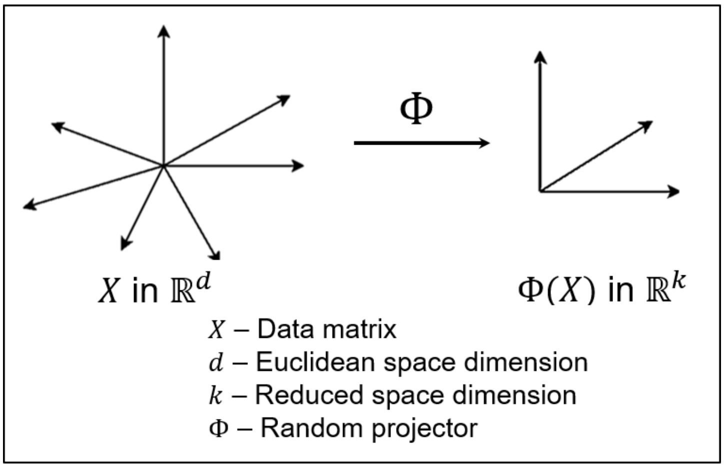

Φ
X in $\mathbb{R}^d$
$\Phi(X)$ in $\mathbb{R}^k$
X – Data matrix
d – Euclidean space dimension
k – Reduced space dimension
Φ – Random projector

Figure 2.1: Projecting the large data space to a lower dimensional data space

Random dimension reduction is used in many fields. In signal processing, compressed sensing reduces the number of samples through sparsity and hence reduces the cost and time of sensing. The main aim of compressed sensing is to reduce the number of samples. However, the change from the original sampling to reduced sampling should be very similar. Compressed sensing is widely used in the medical industry especially in Magnetic Resonance Imaging (MRI) [7]. In coding theory, error-correcting codes and parity-check codes are used to detect and correct errors after receiving a transmission. In neuroscience [8], it was found that the brain compresses large-scale complex information is random ways to handle the amount of information that enters the brain each second. A question arising at this point is that why is a degree of randomness necessary? This is because the information input is not always the same and it would not be the same. The parameters of a projection must be independent of the dataset and hence adding a certain level of randomness to the system. On some occasions data can arrive in a stream hence the projection or algorithm is unaware of the state or form of the next arrival. Hence for randomised dimensional reduction, universality laws were introduced [9]. The universality laws fundamentally build relationships between random linear mappings and preserving geometric structure after embedding onto a smaller dimensional space.

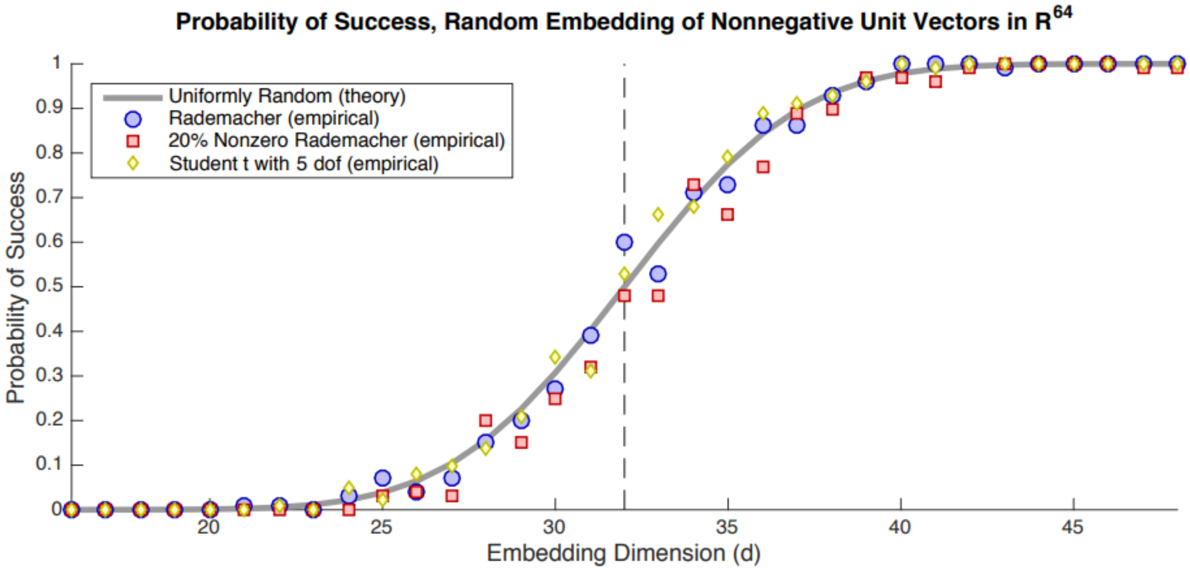


Figure 2.2: Universality of embedding dimension [9]

The above Figure 2.2 shows the probability of success when a set of non-negative unit vectors in $\mathbb{R}^{64}$ is used on 4 types of random linear transforms. The dotted line represents when the embedding dimension is 32. Similarly, this project would compare the performance of the random projection transforms against fixed data.

2.2 Euclidean Space

This project performs transformations in the Euclidean space. There are many reasons behind this choice but the main reason being that Euclidean spaces ability to generalise higher dimensions. The set points in the Euclidean space use properties such as distance and angle to build relationships between one another [13]. The Euclidean distance is a measure which this project uses on many occasions. The l_p-norm is used for distance calculation between points, and it is defined by the following Equation 2.1 for any $x \in \mathbb{R}^n$.

$$\|x_p\| = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} \tag{2.1}$$

The Hilbert space was another consideration for data space. However, though the Hilbert space is a generalisation of the Euclidean space in infinite dimensions, the point of origin of the Hilbert space is unique, zero vector. Whereas in Euclidean space, the point of origin is any reference point to the convention for the available data [15]. This would be useful when the dimension space is very large, and the reduced dimension subspace is far away or does not include the zero vector. In theory, Hilbert space would have to be considered in the future as data grows in complexity.

2.3 K-Nearest Neighbours Learning Algorithm

One of the most straightforward Machine Learning algorithms is the K-Nearest Neighbour Search [11]. When a query is run, the classification of an unclassified data point is needed to be deduced. The classification would be based on the K samples closest to the query by distance. K-Nearest Neighbour search can be done in 2 ways. One is the classification method where the query point is assigned a class same which is the majority class of the k nearest samples. The other method is by regression where the query point is assigned a real value which is the average of the k nearest samples.

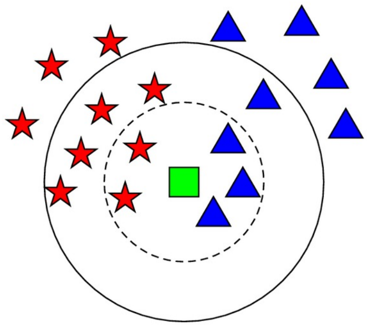


Figure 2.3: K-Nearest Neighbour Classification Search Example [10]

Figure 2.3 is a simple K-Nearest Neighbour Classification search query. In the instance where $K = 5$, the query point would be classified as a blue triangle (▲), because the blue triangles have the majority over the red stars. However, in the case where $K = 10$, the query point will be classified as a red star (★), because now the red stars have the majority over the blue triangles. In regression K-Nearest Neighbours, the query point would be assigned a value rather than a class, and the value is determined by getting the average of the k nearest neighbour's respective values.

Given a dataset with known target values, the K-Nearest Neighbours method can be used to test a model's accuracy and correctness [12]. Once the data is transformed into a reduced dimension, the results of applying the K-NN algorithm to the original dataset and reduced dimensional dataset would be compared. The K-Nearest Neighbours Search algorithm is used for the testing and analysis phase of the random projections to measure correctness. When the dimension is reduced, the new data may or may not have the same knowledge entropy as before. Hence this is an excellent experiment to examine the difference in data and indicate any information lost if applicable.

2.4 Curse of Dimensionality

Define a dataset $X_{n\times m}$, where n is the number of rows (parameters/dimensions) and m is the number of columns (observations/samples). In traditional data processing, there is a small number of dimensions and a large number of samples ($m > n$). However, in modern applications, there are a large number of dimensions but a smaller sample size ($m \approx n$ or $m < n$). The curse of dimensionality [19] arises as the dimensions become much larger than the sample size. This leads to computational difficulties where the computation power of a machine is insufficient to handle large datasets. Another aspect is the intrinsic statistical difficulty where some data are left in isolation, providing false structures and overfitting which describes the noise and not the relationship between the data. Handling the curse of dimensionality is the main issue that this project attempts to solve.

2.4.1 Intrinsic Statistical Difficulty

Isolated Data Points

Same as the K-Nearest Neighbours, where the unclassified data point is assigned the majority class (if classification) or the average value (if regression) with respect to the k nearest samples. Assume a K-NN for a high dimensional dataset. For any given test sample, there must exist at least one training sample within a unit distance from the test sample. The number of training samples required to satisfy this condition must be determined. Consider the following equations which attempt to cover the hypercube $[0,1]^n$ using unit balls [20].

$$V_n(r) = \frac{\pi^{n/2}}{\Gamma(n/2+1)} \overset{n\to\infty}{\sim} \left(\frac{2\pi e r^2}{n}\right)^{n/2} (n\pi)^{-1/2} \tag{2.2}$$

$$[0,1]^n \subset \bigcup_{i=1}^{m} B_n(\mathbf{x}^i, 1) \tag{2.3}$$

$$1 \leq m V_n(1) \tag{2.4}$$

$$m \geq \frac{\Gamma(n/2+1)}{\pi^{n/2}} \overset{n\to\infty}{\sim} \left(\frac{n}{2\pi e}\right)^{n/2} \sqrt{n\pi} \tag{2.5}$$

An n-dimensional sphere of radius $r > 0$ has volume $V_n(r)$ which is described by Equation 2.2. The $\Gamma(n/2+1)$ element is the Euler's gamma function which is nothing but $\Gamma(p) = (p-1)!$ and so the later describes when the number of dimensions n is very large. Equation 2.3 represents the covering of the $[0,1]^n$ hypercube using unit balls. This states that the hypercube is a subset of the union of m many balls. The volume of the union of each ball is less than the summation of the volume of each ball. Hence the minimum value of m can be determined using Equation 2.4. Following are examples of the how the value of m changes with n.

n	**10**	**15**	**20**	**30**	**50**	**100**	**150**
m	1	3	39	45 378	5.76×10^{12}	4.22×10^{39}	1.28×10^{72}

Table 2.1: Number of training samples required for any given test sample. Calculated using Equation 2.5 [20]

As shown in Table 2.1, as n increases, the number of training samples, m increases rapidly. A standard machine would not be able to handle data samples of size over 1×10^{40}. Regardless of the data being normalised, as the dimensions increase, the number of unit balls needed to cover the whole space increases worse than exponentially. Hence machine learning techniques such as K-Nearest Neighbours would not work for large dimensional structure of big data.

False Structure

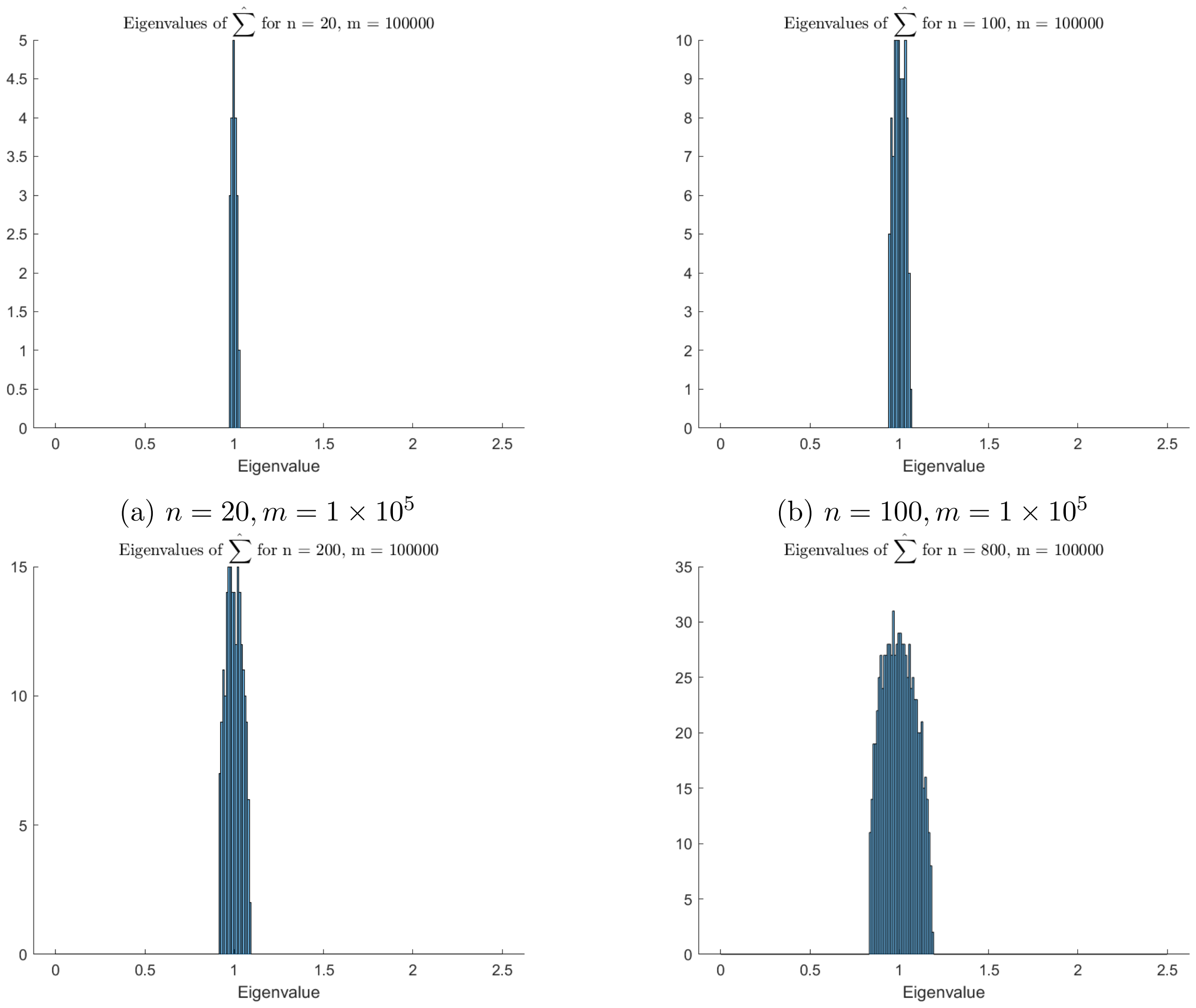
Consider the covariance matrix ($\sum$) in Equation 2.6 and the empirical covariance matrix ($\hat{\sum}$)in Equation 2.7 for a sample set $X = x^{(1)}, x^{(2)}, \ldots, x^{(m)} \in \mathbb{R}^n$ [20]. The law of large numbers states that "the average of the results obtained from a large number of trials should be close to the expected value, and will tend to become closer as more trials are performed"[22]. Hence when the dimension (n) is fixed, and the sample size tends to infinity, the empirical covariance matrix would tend to the identity matrix given in Equation 2.8. This implies that the eigenvalues of the empirical covariance matrix would be close to 1.

$$\sum_{x} := E[XX^T] \tag{2.6}$$

$$\hat{\sum} := \frac{1}{m}\sum_{i=1}^{m} x^i (x^i)^T = E[\tilde{X}\tilde{X}^T] \tag{2.7}$$

$$m \to \infty \implies \hat{\sum} \to I \tag{2.8}$$

When the number of dimensions (n) is very much smaller than m ($m >> n$), the eigenvalues of the empirical covariance matrix converges to 1 or are very much close to 1 as seen in Figure 2.4. Hence it can be deduced that the empirical covariance matrix tends to the identity matrix. However, in the case where the number of dimensions (n) is approximately or just less than the samples size (m), the eigenvalues tend to be more distributed along the axis.



(a) $n = 20, m = 1 \times 10^5$

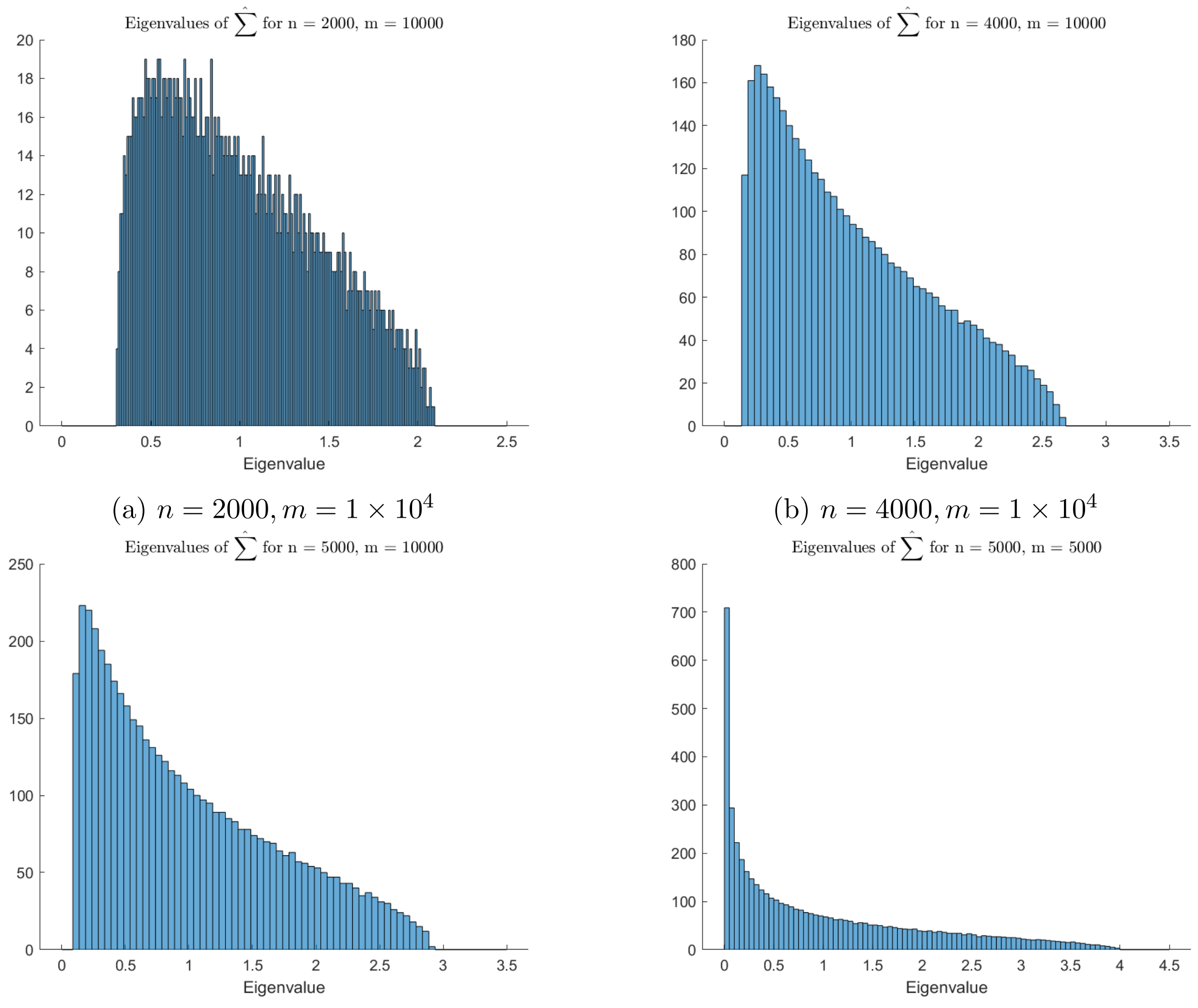
(b) $n = 100, m = 1 \times 10^5$

(c) $n = 200, m = 1 \times 10^5$

(d) $n = 800, m = 1 \times 10^5$

Figure 2.4: Eigenvalues of the empirical covariance matrix for low dimensions [20]

As seen in Figure 2.5, the eigenvalues of the empirical covariance matrix are more distributed and do not converge to 1. This implies that as the dimension of the data increases, the empirical covariance matrix becomes very much different from the identity matrix. This phenomenon is due to the asymptotic behaviour of singular values which given by the Marchenko–Pastur distribution [23].



(a) $n = 2000, m = 1 \times 10^4$

(b) $n = 4000, m = 1 \times 10^4$

(c) $n = 5000, m = 1 \times 10^4$

(d) $n = 5000, m = 0.5 \times 10^4$

Figure 2.5: Eigenvalues of the empirical covariance matrix for high dimensions [20]

Overfitting

This is one of the main problems that takes place in the machine learning field. When a hypothesis h_1 performs better (smaller error) than hypothesis h_2, but h_2 has a better performance than h_1 on unseen data (test data), h_1 is said to be overfitting [24]. Overfitting can occur in many occasions. In neural networks, when the learning algorithm is performed for an extended period of time, the algorithm starts to capture the noise in the training data. When the number of training samples is low, the existing examples may not represent every possible situation. Thus when new data arrives, the model would incorrectly classify the data. In the case of dimensionality, when the number of dimensions is very large, the model becomes very complex. Consider a regression problem with pairs of data (x_i, y_1), for $i = 1, \dots, m$, where x are the features set and y is the target. The algorithm must prepare a liner function, $\alpha \in \mathbb{R}^n$ such that $y_i \approx < x_i, \alpha >$ represented by the following equation where $\mathbf{y} = \{y_1, \dots, y_m\}$, $\mathbf{X} = \{x_1, \dots, x_m\}$ and $\mathbf{e} = \{e_1, \dots, e_m\}$ [20].

$$\mathbf{y} = \mathbf{X}\alpha + \mathbf{e} \tag{2.9}$$

In the case of a flat matrix ($m < n$), $\mathbf{y} = \mathbf{X}\alpha$ would have infinitely many solutions. $\mathbf{X}$ may be a full rank matrix, but even a small noise can affect the solution. To match the data perfectly for a m many distinct samples, there exists a polynomial of degree $m-1$ that would match the data perfectly [20]. This would yield the following equation.

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} 1 & x_1 & \dots & x_1^{m-1} \\ 1 & x_2 & \dots & x_2^{m-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \dots & x_m^{m-1} \end{pmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{m-1} \end{pmatrix}}_{\alpha} \tag{2.10}$$

Since the data is fitted perfectly, the noise carried by the samples would also be considered. This defeats the purpose of a machine learning algorithm which should generalise the data in order to predict new data accurately.

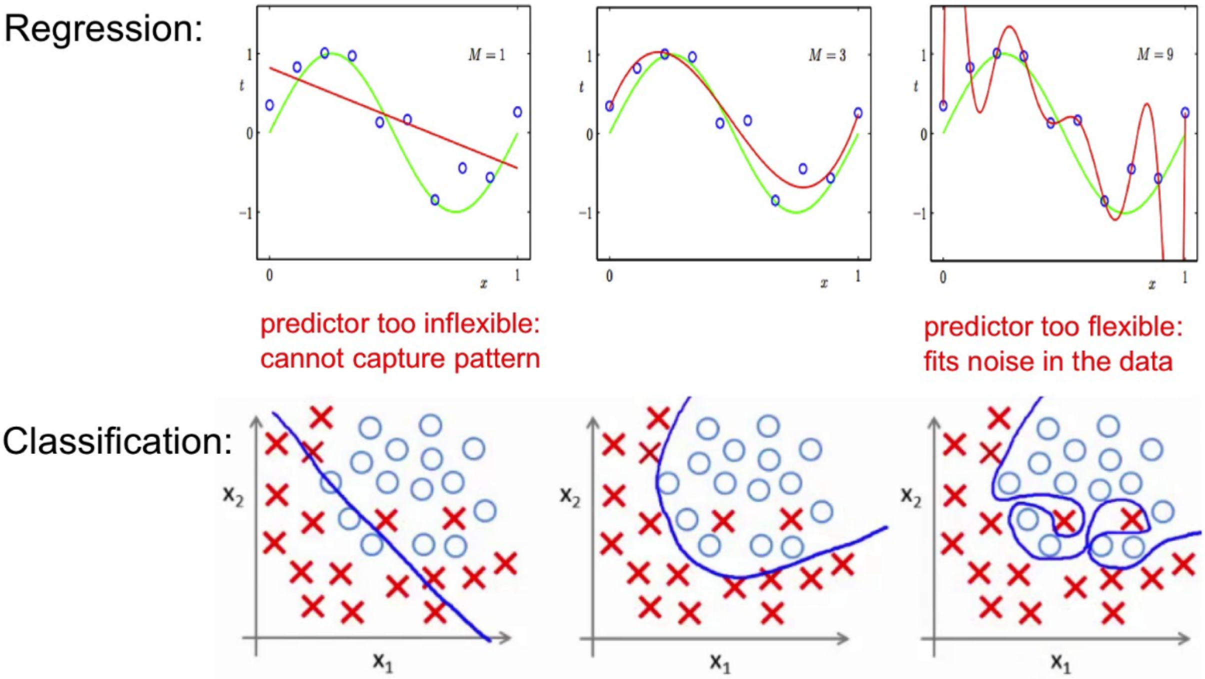


Figure 2.6: Examples of Underfitting and Overfitting [25]

Figure 2.6 indicates the behaviour of a model when it is exposed to overfitting. The hypothesis attempts to perfectly classify the training data and hence loses generality in the model. This is confirmed in Figure 2.7, where the error in training data decreases but due to the loss in generality, the error on unseen data increases rapidly. Thus leading to incorrect prediction and the model is impractical.

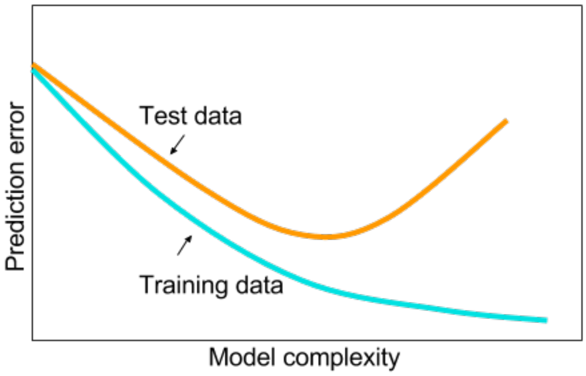


Figure 2.7: Overfitting affecting the prediction error [26]

2.4.2 Computation Difficulty

Big data can be categorised into 4 aspects such as volume, velocity, variety, and veracity [27]. This project will only focus on the volume aspect in terms of sample size, dimension size, and the computation time of the projection. As the dimensionality of datasets increase, the complexity of the model being created through a learning algorithm would also increase. In the case where a neural networks model is to be created in order to classify 900-pixel images into facial emotions, the time taken to train and reach a well-generalised model will increase. This is mainly due to the number of dimensions being high which adds complexity to the hidden layers of the neural networks model. Thus making the algorithm change the structure of features at every iteration. This project looks closely at the change in computation time of the projection as dimensionality increases which is covered in the later stages.

2.5 Johnson-Lindenstrauss Transform

Random projections for dimension reduction were revolutionised by the theorem introduced by William Johnson and Joram Lindenstrauss in the 1980s. They introduced the Johnson-Lindenstrauss (JL) transform which is given by the following equation. For all $(x, y) \in X$,

$$(1-\epsilon)\|x-y\|_2 \leq \|\Phi(x)-\Phi(y)\|_2 \leq (1+\epsilon)\|x-y\|_2 \tag{2.11}$$

What is implied by Equation 2.11 is that projecting any points in the dataset X from a high-dimensional space to a low-dimensional space must, up to a distortion of $(1 \pm \epsilon)$, preserve pairwise distances [16]. Based on pairwise distances, they define the random projector matrix $\Phi_{k\times d}$ (embedding dimension, k and Euclidean dimension, d) with the rows being random unit vectors orthogonal to each other. However, this original JL projection has a computational time of order $O(kn)$, where n is the size of the dataset X and the embedding dimension, $k \approx \epsilon^{-2}\log n$ [17, 3]. Since then numerous research took place to reduce the value of k while improving on the computation time and complexity.

After the introduction to the initial Johnson-Lindenstrauss transform, the fast JL methods showed significant improvement in performance through means for sparse matrices, error-correcting code matrices, restricted isometry property conversions etc. The common characteristic of all these methods in the randomness. In any instance, there exists a certain degree of randomness in the projection, random diagonal matrices or even the elements of the projector as independent and identically distributed Gaussian entries [18]. In addition, most cases use a Fourier transform to speed up the computation similar to simple Fast Fourier Transforms in sampling signals. Combining these two would densify the input data to make sure that no information is isolated. Thus resulting in lower distortions. This area will be further looked at when designing and preparing the random projection in Section 3.

2.6 Error Correcting Codes

Since the information age, coding theory has played a significant role in the digital revolution. As the size of an electronic device decreased, the data communication between these machines has drastically increased [33]. For example, a Radio Common Carrier used by the military in the 1960s for just communication has now developed into a smartphone which is the size of a hand but has a significantly broader range in terms of functionality. Due to the increase in data demand, it required error detecting and error correcting mechanisms on the machine. Hence Error-correcting codes became popular in the communication world.

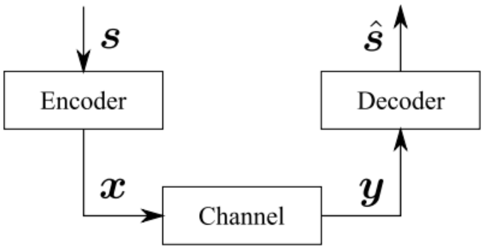


Figure 2.8: Communication between sender and receiver through a channel [32]

In any device, it is essential that the input signal (S) and the output signal ($\hat{S}$) in Figure 2.8 are the same. The error-correcting code would handle the errors produced by the channel before sending out the information. The concepts of BCH codes in error-correcting codes are used in this fast JL method. The Bose-Chaudhuri-Hocquenghem (BCH) belong to the cyclic codes class. A matrix is cyclic if, for each vector x in the matrix X, the vector x' obtained by shifting the elements of the vector x cyclically to the right by one unit is also a vector of the matrix X [33].

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \qquad \begin{aligned} a &= \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \\ a' &= \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \end{aligned}$$

The above matrix A is a cyclic code because all its vectors (rows) are a right cyclical shift of one vector. BCH Codes are the cyclic codes with generator polynomials that "make the minimum distance guaranteed by the BCH bound, large" [33].

Chapter 3

Analysis and Design

3.1 Fast Johnson-Lindenstrauss Transform 1

3.1.1 Previous Work

After introducing the JL projection, many amended and refined the original JL. However, the computation and complexity did not significantly improve. Through a clever observation by Prof. Dimitris Achlioptas [28], sparsity was factored into the random projection matrix. Using the properties of l_2 norm and distortion vectors between points, he defined the elements of the random projection matrix as follows:

$$\Phi_{i,j} = \begin{cases} \frac{\sqrt{3}}{d} & \text{with probability } \frac{1}{6} \\ 0 & \text{with probability } \frac{2}{3} \\ -\frac{\sqrt{3}}{d} & \text{with probability } \frac{1}{6} \end{cases} \tag{3.1}$$

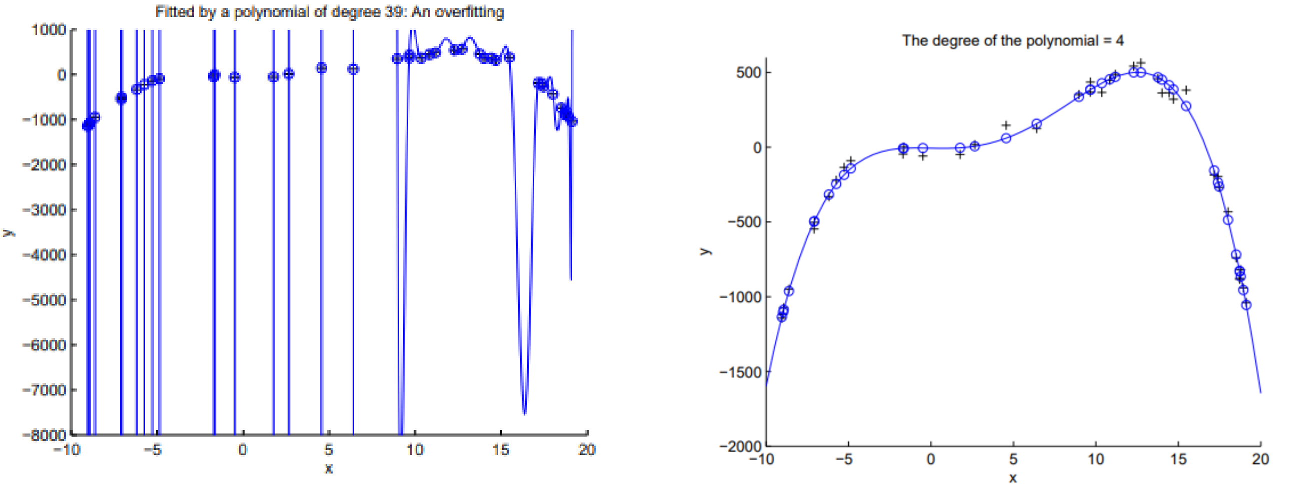
A matrix is defined to be sparse when most of its elements are 0. In this case $\frac{2}{3}$ of the elements of Φ is 0, making Φ a sparse matrix. The sparsity of this matrix helps to reduce the computation time significantly because most of its elements are 0 hence these entries can be ignored. However, there must be a restriction to limit the level of sparsity any matrix can have. This is due to instances when specific data points (vectors) are very sparse themselves.

$$\Phi = \begin{bmatrix} c_1 & 0 & c_6 & 0 & 0 & 0 \\ 0 & c_4 & 0 & 0 & c_9 & 0 \\ c_2 & 0 & 0 & c_8 & 0 & c_{11} \\ 0 & 0 & c_7 & 0 & 0 & 0 \\ c_3 & 0 & 0 & 0 & 0 & c_{12} \\ 0 & c_5 & 0 & 0 & c_{10} & 0 \end{bmatrix} \qquad x = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

In the example above, Φ is a sparse random projection matrix with $\frac{2}{3}$ of its elements being 0 and x is a sparse vector. By observation, only the element c_8 is able to carry the information of x. Therefore in some instances, information carried in very sparse vectors such as x can be ignored or isolated. Hence there was a necessity to densify the projection matrix to avoid loss of information due to too few non-zero coordinates.

3.1.2 Sparse Approximation

In Chapter 2, the curse of dimensionality was introduced and explained how overfitting could occur in big data applications. In order to match the data perfectly, a polynomial of degree $m - 1$ which exists, must be found. This loses the generality, and hence noise interferes with the data model. The role of sparse approximation is that it sets certain polynomials to zero. In the same instance as earlier, it sets a few elements of α in Equation 2.10 to zero. Leaving only a few non-zero elements and making α is a sparse vector. This allows the data model to be generalized because the hypothesis function is not complex.



(a) α has 39 non-zero elements

(b) α is sparse and has only 4 non-zero elements

Figure 3.1: Prefect fit vs Generalized classification example [20]

3.1.3 Underlying Method

Since the need to densify the matrix, Prof. Nir Ailon and Prof. Bernard Chazelle introduced the fast JL transform which included Fourier transform properties in addition to sparsity [3]. The random projection matrix was defined as:

$$\Phi_{k\times d} = S_{k\times d}.H_{d\times d}.D_{d\times d} \tag{3.2}$$

$H_{d\times d}$ represents an extended form of discrete Fourier transform over the field $\mathbb{F}_{n^d}$ called Walsh-Hadamard matrix (where n is the size of the dataset). The elements of the matrix are defined as:

$$H_{i,j} = d^{-\frac{1}{2}}(-1)^{<i-1.j-1>} \tag{3.3}$$

where $< a, b >$ is the dot product between a and b.

This matrix has a runtime of $O(d \log d)$ [3]. The underlying concept is based on the Uncertainty Principle for signal processing. It takes advantage of the fact that "a signal cannot be localised in both signal's time domain and its spectrum's frequency domain" and the ability to obtain precise values (measurements) [29, 30]. The Walsh-Hadamard matrix is essentially preconditioning the input with a fast Fourier transform. This step densifies the isometry of any sparse vectors in the input matrix [3]. However, in order to avoid loss of information, there must exist a certain degree of randomness in the Fourier matrix so that dense vectors would not be sparse. Hence a random matrix needs to be included.

$D_{d\times d}$ is a random diagonal matrix whose elements are ± 1 with probability $\frac{1}{2}$.

$$D = \begin{pmatrix} \pm 1 & 0 & \cdots & 0 \\ 0 & \pm 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \pm 1 \end{pmatrix}$$

This matrix adds a certain degree of randomness to the projection. Hence avoids sparsification of dense vectors. Note that the H and D are orthogonal square matrices and hence the projection from a high-dimensional space to a low-dimensional space results in no distortion [3]. Now that densification of any sparse vectors preconditions the input, it is ready to be computed with the sparse matrix.

$S_{k\times d}$ is a sparse matrix with a sparsity of $|S|$ non-zero entries (usually $\approx k^3$). The elements of the matrix are an independent mixture, $S_{i,j} \sim \mathcal{N}(0, \frac{1}{q})$ with some probability q and $S_{i,j} = 0$ with probability $1 - q$. Only $|S|$ elements are non-zero, and hence only these elements are required for computation of with this matrix. So only an $O(|S|)$ factor is added to the runtime [3]. The value of q is determined by the following equation where n is the sample size and $p \in \{1, 2\}$.

$$q = \min\left\{\Theta\left(\frac{\epsilon^{p-2}\log^p n}{d}\right), 1\right\} \tag{3.4}$$

3.1.4 Theory on Performance

When an input vector $x \in \mathbb{R}^d$ is multiplied by the random matrix D, $D(x)$ requires $O(d)$ steps. Then to compute the Hadamard matrix multiplication, $H(Dx)$, requires $O(dlogd)$ which is the fast Fourier transform. The final multiplication with the sparse matrix, $S(HDx)$, requires $|S|$ steps where $|S|$ is the number of non-zero entries of S [3].

This method has a computation time of $O(d \log d + |S|)$, where $d \log d$ is the runtime for the Walsh-Hadamard Fourier transform matrix and $|S|$ being the l_0 pseudo-norm of the sparse matrix (number of non-zero entries). When the embedding dimension (k) is very small, $k \leq d^{\frac{1}{3}}$, the runtime becomes approximately $O(d \log d)$ [3]. The Theorem for the fast JL transform is given below, for proofs refer [3].

Theorem 3.1 *Given a fixed set of X of n points in $\mathbb{R}^d$, $\epsilon < 1$, and $p \in \{1, 2\}$, draw a matrix Φ from FJLT (Fast Johnson-Lindenstrauss Transform). With probability at least $\frac{2}{3}$, the following 2 events occur:*

1. *For any $x \in X$,*

   $(1-\epsilon)\alpha_p\|x\|_2 \leq \|\Phi x\|_p \leq (1+\epsilon)\alpha_p\|x\|_2$
   where $\alpha_1 = k\sqrt{2\pi^{-1}}$ and $\alpha_2 = k$.

2. *The mapping $\Phi : \mathbb{R}^d \to \mathbb{R}^k$ requires,*

   $O(d \log d + \min\{d\epsilon^{-2} \log n, \epsilon^{p-4} \log^{p+1} n\})$
   operations.

This project will consider the Euclidean space Johnson-Lindenstrauss properties (JLP) which is when $p = 2$ rather than the study of the Manhattan space JLP when $p = 1$. As sated in the theorem, computation of Φx will be $O(d \log d + \min\{d\epsilon^{-2} \log n, \epsilon^{p-4} \log^{p+1} n\})$. This is derived from the previously stated computation time of $O(d \log d + |S|)$ where that the latter is of $O(\epsilon^{-2} \log n)$. Hence the worst case runtime is of $O(d \log d + qd\epsilon^{-2} \log n)$ where the qd terms arrive from the instant that d elements are non-zero (which has probability q) in the sparse matrix.

3.2 Fast Johnson-Lindenstrauss Transform 2

3.2.1 Previous Work

The fast JL method 1 made an improvement on the runtime performance and also the embedding dimension limit compared to Dimitris Archlioptas's approach through sparse approximation. This gave a runtime of $O(d \log d)$ for values of $k = O(d^{\frac{1}{3}})$ [3]. The difficulty was that once the upper-bound of k is exceeded beyond the limit, the method fails. More specifically, the modelling of the random projection matrix Φ becomes invalid because the distortion increases for small ϵ values as well as the computation time increases. Hence the Johnson-Lindenstrauss properties no longer hold. Further research by Prof. Nir Ailon and Dr Edo Liberty, increased the upper-bound of k for a better runtime of $O(d \log k)$ [31]. The research used the same approach of using a randomised extended Fourier transforms but uses concepts from coding theory rather than sparsity. This section covers the fast JL method technique using BCH codes and error correcting codes from coding theory.

3.2.2 Underlying Method

This method consists of two theorems [31].

Theorem 3.2 *For any code matrix A of size $k \times d$ for $k < d$, the mapping $x \mapsto Ax$ can be computed in time $O(d \log k)$.*

Theorem 3.3 *Let $\delta > 0$ be some arbitrarily small constant. For any d, k satisfying $k < d^{\frac{1}{2}-\delta}$, there exists an algorithm constructing a random matrix A of size $k \times d$ satisfying Johnson-Lindenstrauss properties $(0 \leq \epsilon \leq 1/2)$, such that the time to compute $x \mapsto Ax$ for any $x \in \mathbb{R}^d$ is $O(d \log k)$. The constriction uses $O(d)$ random bits and applies to both the Euclidean $(p = 2)$ and the Manhattan $(p = 1)$ cases.*

However the construction of the projection matrix A is not simple in terms of theory but it has fast computation.

3.2.3 Theorem Explanation

Use of error correcting codes

The initial step is to use the functionalities of error correcting codes to define certain Lemmas. Within a set of random variables, if there are m independent random variables, that set of random variables is said to be m-wise independent. m-wise independent variables become useful in cases of complex random algorithms since it is easier to generate than entirely independent random variables (full rank matrices) and it is useful for any k value [34]. Hence the following lemma is introduced [31].

Lemma 3.1 *There exists a 4-wise independent code matrix of size $k \times f_{BCH}(k)$, where $f_{BCH}(k) = \Theta(k^2)$.*

Use of BCH codes

The proof assumes that $\delta > 0$ and it is an arbitrarily small constant. Let B be a matrix containing unit norm vector columns whose dimension is $k \times d$ and a random diagonal matrix, D of dimension $d \times d$ with elements ±1 with probability 1/2, similar to FJLT method 1.

Lemma 3.2 *Assume B is a $k \times d$ 4-wise independent code matrix. [31]*

The above Lemma 3.2 along with BCH code Lemma 3.1, the code matrix B is constructed. Select an appropriate f_{BCH} (denoted by β for simplicity) that is divisible by d. The values of d and β must be a power of 2 so that in binary terms, d is at most β with padding of zeros to the end. Define a matrix B_k with dimensions $k \times \beta$ derived from Lemma 3.2. Formally, define B which is d/β copies of the matrix B_k side by side. The resulting code matrix B is still 4-wise independent because the B_k matrices are laid horizontally. Though the matrix B

is 4-wise independent, it is no longer a code matrix [31]. The structure of the matrix B is as follows:

$$B_{k\times d} = [B_k \quad B_k \quad B_k \dots B_k] \tag{3.5}$$

Since 4-wise independence is considered, $\|x\|$ must be controlled for $k < d^{1/2-\delta}$. Similar to FJLT method 1, a randomised orthogonal transformation matrix, Φ is introduced. This has a running time of $O(d \log d)$ [31]. The primary product that undergoes the construction of Φ is HD', where H is the Walsh-Hadamard matrix, and D' is the random ± 1 diagonal matrix. D' denotes that the random diagonal matrix is to be computed some r number of times, which is denoted but Equation 3.6 and yields Equation 3.7.

$$D' = D^{(1)}, D^{(2)}, D^{(3)}, \dots, D^{(r)} \tag{3.6}$$

$$\Phi_d^{(r)} = HD^{(r)}HD^{(r-1)} \dots HD^{(1)} \tag{3.7}$$

As these techniques are combined, the randomised 4-wise independent matrix and the randomised orthogonal matrix, the projection matrix A is finally constructed by $A = BD\Phi^{(r)}$, and it obeys Euclidean Johnson-Lindenstrauss Properties for $k < d^{1/2-\delta}$ and requires a runtime of $O(d \log d)$. However, due to the nature and the structure of these elements/matrices and how they are designed, it is possible to achieve a runtime of $O(d \log k)$ [31].

Since the code matrix B_k was copied d/β to construct B, the same idea can be applied to Φ as well. The $d \times d$ matrices in Φ are decomposed as, d/β copies of $\beta \times \beta$ Walsh-Hadamard matrices, denoted by H_β and d/β copies of $\beta \times \beta$ random diagonal matrices. Hence the running time would drop to $O(d \log k)$ since the Walsh-Hadamard matrix can be applied independently on each block.

$$BD = \underbrace{(B_k \quad B_k \quad B_k \dots B_k)}_{d/\beta \text{ copies of } k\times\beta \text{ blocks}} \underbrace{\begin{pmatrix} D_\beta & 0 & \cdots & 0 \\ 0 & D_\beta & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D_\beta \end{pmatrix}}_{d/\beta \text{ copies of } \beta\times\beta \text{ blocks}} \tag{3.8}$$

$$HD' = \underbrace{\begin{pmatrix} H_\beta & 0 & \cdots & 0 \\ 0 & H_\beta & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & H_\beta \end{pmatrix}}_{d/\beta \text{ copies of } \beta\times\beta \text{ blocks}} \underbrace{\begin{pmatrix} D_\beta^{(1)} & 0 & \cdots & 0 \\ 0 & D_\beta^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D_\beta^{(r)} \end{pmatrix}}_{d/\beta \text{ copies of } \beta\times\beta \text{ blocks}} \tag{3.9}$$

From Equation 3.8 and Equation 3.9, it is clear that computation can be done independently for separate blocks. Hence the running time gets reduced to $O(d \log k)$.

3.2.4 Theoretical Performance

This method has improved the runtime performance by eliminating the $\log d$ factor to give a new runtime of $O(d \log k)$. In addition, the upper-bound of the embedding dimension is now $d^{\frac{1}{2}-\delta}$ compared to $d^{\frac{1}{3}}$ in method 1. However, still, this method would produce distortion above the JL limit if k exceeds $d^{\frac{1}{2}-\delta}$. This method uses random variables in Rademacher distribution for the proof of bounding the embedding dimension [31].

Chapter 4

Implementation

4.1 FJLT Method 1

4.1.1 The Projection Matrix

The FJLT method 1 is created using the underlying Equation 3.2 in Chapter 3. This involves the multiplication of the sparse matrix, Hadamard matrix and the random diagonal matrix in order to obtain the projection matrix.

The elements of the sparse matrix are non-zero with probability q and are zero with probability $1-q$. A non-zero element is a random number from the normal distribution with mean 0 and standard deviation $1/q$. The parameter q is dependent on the number of samples (n) and the original dimension of the data (d), it is defined as $q = \min\left\{\Theta(\frac{\log^2 n}{d}), 1\right\}$. Hence this matrix was implemented using the MATLAB function `normrnd(mu,sigma)` [41] for the non-zero elements and the function `datasample()` [42] with weights $[q, 1-q]$ for the selection of non-zero and zero elements.

The Walsh-Hadamard matrix has the following simple properties.

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad H_{2^{2n}} = \begin{pmatrix} H_{2^n} & H_{2^n} \\ H_{2^n} & -H_{2^n} \end{pmatrix}$$

In order to normalise the matrix, the matrix is multiplied by $1/\sqrt{d}$. The MATLAB function `hadamard(d)` [43] was used to create this matrix. To prepare the random diagonal matrix, the MATLAB function `datasample()` with equal weights of $[0.5, 0.5]$ for ±1 was used for the selection of the diagonal elements. Finally, these 3 matrices are multiplied to obtain the projection matrix of the Fast Johnson-Lindenstrauss Transform Method 1.

4.1.2 Verification

The approach for verifying the validity of this projection method is to make sure that the results follow the statements of Theorem 3.1 stated in Chapter 3. The main statement that should be tested at this stage is that with 2/3 probability, given an error $\epsilon < 1$, the projection preserves the l_2-norm (distance) in the Euclidean space. This is the first statement of the FJLT Method 1 theorem which is given by the following equation.

$$(1-\epsilon)k\|x\|_2 \leq \|\Phi x\|_2 \leq (1+\epsilon)k\|x\|_2 \tag{4.1}$$

Once the projection matrix (Φ) is prepared, each data point (x) is multiplied by Φ to obtain the reduced dimensional data point (Φx). Once all data points are transformed into the reduced plane, the probability that the inequality in Equation 4.1 is held, is calculated. Another measurement carried out at this stage is the computation time of Φx for any data point x. This measurement is used for the comparison of the 2 FJLT methods with regards to running time. In order to increase precision, this process is repeated for 20 iterations and the measurements of probability and computation time are obtained as an average.

The other important aspect that must be verified is the inequality $k \leq d^{1/3}$, where the reduced dimension, k is upper bounded by $d^{1/3}$. The reduced dimension used for testing ranges from values small as 2 to values larger than 30. This approach is used so that there can be a significant change in the probability and computation time as k goes beyond $d^{1/3}$.

4.1.3 Parameter Selection

The datasets used at this stage are randomly selected values between 0 and 1. To avoid points which are outliers or clustered, the data is normalised with respect to features. Normalisation can be approached in many ways. In this instance, the following method is used, for a features attribute a,

$$a_{new} = \frac{a_{old} - a_{min}}{a_{max} - a_{min}} \tag{4.2}$$

The above equation ensures that the new values in the features attribute is distributed between 0 and 1. This becomes an advantage when datasets contain features sets with values of different ranges. For example, a tyre characteristics dataset may have a features attribute for thread depth in millimetres, mileage in kilometres, and durability in years. Hence all these attributes would be normalised to a set range by using Equation 4.2.

In order to observe trends in validation stage, different values are used for parameters such as reduced dimension, original dimension, and sample size.

These parameters have values within a broad range because the trends become visible as the dimensionality is altered. In addition, it helps to identify the optimal parameter set when performing comparisons with FJLT method 2. It is a requirement that the original dimension is a power of 2 because the construction of the Hadamard matrix requires the dimensions to be a power of 2 square matrix. In the case where the dataset has a dimensionality that is not a power 2, extra zero feature sets can be padded to the dataset to reach 2^b dimensions which

Parameter		**Values**
Sample Size	n	{50, 100, 500, 1000, 5000, 10000}
Original Dimension	d	{32, 64, 128, 256, 512, 1024}
Reduced Dimension	k	{2, 3, 4, 8, 10, 16, 24, 32, ...}

Table 4.1: Parameter values

is higher than the original dimension. The padding zeros to the end of the dataset will not affect the data in any way because the addition of a zero feature implies that the respective dimension has value 0 for all data points.

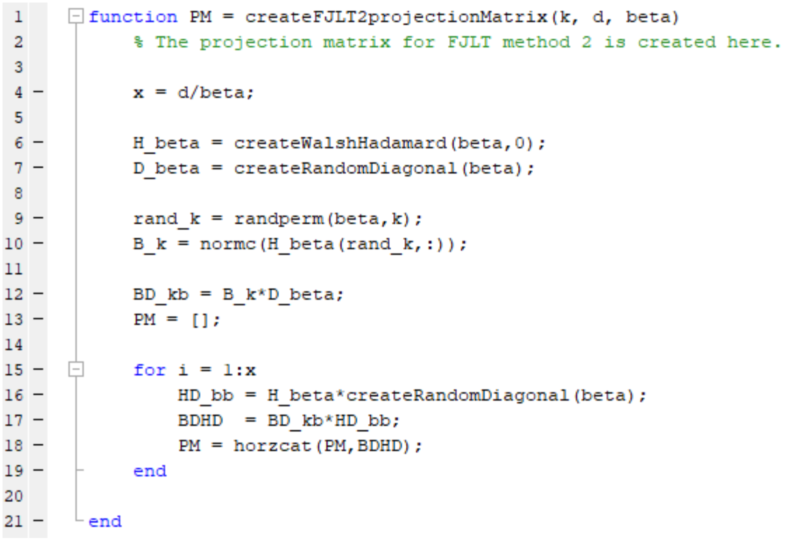
The data on the average probability of Equation 4.1 being held against the error, across different reduced dimensions (k) is recorded. In addition the error at which the respective dimension reaches the 2/3 probability of holding the inequality in Equation 4.1 is also recorded.

4.2 FJLT Method 2

4.2.1 Projection Matrix

Understanding the approach in preparing the projection matrix for FJLT method 2 is more complex compared to FJLT method 1. However, it is more simpler to compute since the large matrices are broken down into smaller matrices and computed them separately. As mentioned in the 3.2.3 Theorem Explanation section in Chapter 3, an appropriate β value is selected initially which is of $\Theta(k^2)$. Then a $\beta \times \beta$ Walsh-Hadamard matrix (H_β) is obtained using the MATLAB function `hadamard(β)` similar to the previous method. The repeating $k \times \beta$, B_k matrix is created by selecting k random rows of H_β and normalising the matrix by having unit norm columns which are feasible since $\beta = \Theta(k^2)$. Then the B_k matrix is multiplied by a $\beta \times \beta$ diagonal matrix. Obtaining the resulting $k \times \beta$, BD matrix is the first step in creating the projection matrix.

When the theory is observed carefully, it is clear that only the latter part, the randomised orthogonal transformation matrix Φ, given by Equation 3.7 and Equation 3.9 in Chapter 3. Since B_k is copied d/β times side by side, the computation of multiplying Hadamard matrix, H_β and a random diagonal matrix, D_β must be repeated d/β times, each time with a newly created D_β matrix. In each iteration, the resulting HD matrix is multiplied with the previously obtained BD matrix and then concatenated horizontally to the end of the projection matrix. This method of computation by breaking down the elements is shown in Figure 4.1.
This concludes the approach undertaken to prepare the projection matrix for the Fast Johnson-Lindenstrauss Transform method 2.



```
function PM = createFJLT2projectionMatrix(k, d, beta)
    % The projection matrix for FJLT method 2 is created here.

    x = d/beta;

    H_beta = createWalshHadamard(beta,0);
    D_beta = createRandomDiagonal(beta);

    rand_k = randperm(beta,k);
    B_k = normc(H_beta(rand_k,:));

    BD_kb = B_k*D_beta;
    PM = [];

    for i = 1:x
        HD_bb = H_beta*createRandomDiagonal(beta);
        BDHD  = BD_kb*HD_bb;
        PM = horzcat(PM,BDHD);
    end

end
```

Figure 4.1: Creating the FJLT method 2 projection matrix

4.2.2 Verification

The verification approach of this method's projection matrix is based on the Johnson-Lindenstrauss property described in Equation 2.11 of Chapter 2. For a given error, $\epsilon < 0.5$, the projection must preserve the l_2-norm (distance) in the Euclidean space. The inequality used for this verification is as follows:

$$(1-\epsilon)\|x\|_2 \leq \|\Phi x\|_2 \leq (1+\epsilon)\|x\|_2 \tag{4.3}$$

Similar to the previous method, once the projection matrix (Φ) is prepared, each data point (x) is multiplied by Φ to obtain the reduced dimensional data point (Φx). Then once all the data points are transformed into a reduced dimensional space, the probability that the inequality in Equation 4.3 being held is calculated. In addition, the time to compute the Φx multiplication for any data point x is calculated. This is the running time measurement of the FJLT method 2 and it is used for comparison with the FJLT method 1. Similarly, to ensure precision, the process is repeated for over 30 iterations the resulting probability and computation time measurements are obtained in averages.

As mentioned in Theorem 3.3 of Chapter 3, the projection matrix is valid for the reduced dimensions, $k \leq d^{1/2-\delta}$, for an arbitrary small $\delta > 0$. Hence a wide range of values for reduced dimension, k was used for identifying critical differences as dimensionality is altered and goes beyond $d^{1/2}$.

4.2.3 Parameter Selection

The datasets used for this stage are similar to the datasets used in FJLT method 1, that is, randomly selected values between 0 and 1. However, the normalisation is approached using the unit norm columns method where the resulting dataset would have unit norm data points.

The range of values selected for parameters such as original dimension, reduced dimension, and the sample size is the same as the values mentioned in Table 4.1. In addition to these inputs, there is another parameter which must be selected which is β. It is defined to be $\beta = \Theta(k^2)$.

$\boldsymbol{k}$	k^2	Ideal β
2	4	4
3	9	8
4	16	16
5	25	16
6	32	32
7	49	32
8	64	64
9	81	64
10	100	128
12	144	128
15	225	256
18	324	256
20	400	512
22	484	512
25	625	512
28	784	512
30	900	1024
32	1024	1024

Table 4.2: Ideal β values for some reduced dimension, k values

However, it is important to note that the β value is upper bounded by the original dimension value d. For an instance where $d = 32$, if the reduced dimension values are selected as 3, 4, 6, 8, 10, and 15, their β values would be 8, 16, 32, 32, 32, and 32 respectively.

To visualise the results of this method, the average probability of Equation 4.3 being held is against the error is plotted for many reduced dimension values. In addition, the error at which the best probability is produced for each reduced dimension is recorded.

4.3 FJLT Method 1 vs FJLT Method 2

Following the two theorems stated in Chapter 3, it is clear that for FLJT method 2, there exists a matrix for which the JL properties are obeyed. Whereas in FJLT method 1, the JL properties are obeyed with probability 2/3. In addition, the first method has a runtime of $O(d \log d)$ compared to a runtime of $O(d \log k)$ to compute Φx.

The parameter values used for this testing stage is same as the values mentioned in Table 4.1, where these values are applied to the 2 methods. The same dataset is used when testing a specific set of parameters in order to observe the difference between the 2 methods with respect to the algorithm and not the data. The probability of success is calculated with respect to obeying JL properties for each reduced dimension k. In addition, the runtime of computing Φx is also recorded. To maintain precision, the 2 methods are repeated for 10 iterations, and the measurements are obtained as an average.

This would give a clear idea of the relationship between error and the reduced dimension of both methods and would give the ability to compare their computation times. It is important to note that the selected range of reduced dimension values must include the points around $d^{1/3}$ and $d^{1/2}$ because these are the upper bounds of the FJLT method 1 and 2 respectively.

Hence the change in the probability of success and computation must be undoubtedly visible to deduce any correlation with the theorems.

4.4 K-Nearest Neighbors Search Algorithm

Once the 2 FJLT methods are validated and tested, the projection matrices are used in a machine learning problem where it must classify specific points using the k-nearest neighbour (k-NN) algorithm. The dataset used for this experiment is a sample data set from the MATLAB Statistics and Machine Learning Toolbox [44].

4.4.1 Data

The data set is formally known as Iris flower dataset (`fisheriris.mat` in MATLAB). Where the dataset has a set of features and are classified into three related species [45]. Where the attributes include characteristics of the flower, and the target includes one of *setosa*, *versicolor*, or *virginica*. This dataset was used for its simplicity to distinguish the differences between the FJLT projected datasets and the original dataset. Thus giving the ability to observe structural changes and other characteristics.

4.4.2 Scenario Implementation

Initially, the data matrix is normalised using the Equation 4.2 for each feature set. This results in each attribute getting standardised to values between 0 and 1. Then the resulting normalised dataset is used to create the reduced dimensional projected datasets for certain values of k. Formally, for each normalised dataset, a set of data points are chosen at random for testing hence each class of this set's data points is predicted. The MATLAB function `knnsearch(SampleData,TestData,‘k’,10)` provides the indices of the 10 nearest neighbours for each data point. Hence with this information, each test data point is classified to its respective majority class.

In the next stage of this experiment, 10-fold cross-validation is applied on each of the obtained projected datasets as well as the original. This cross-validation method splits the data into 10 folds and uses one fold as the test set in each iteration. For every dataset, the classification error rate and the confusion matrix is recorded. The confusion matrix is recorded because it helps in justifying the structure of the projected datasets by observing how each data point is classified correctly and incorrectly.

4.4.3 Data Representation

For stage 1 results of this experiment, both the actual and predicted values are recorded for each dataset. The size of the random test is gradually increased, and the change in classification is recorded. The `fisheriris` data is represented by the Figure 4.2 and Figure 4.3. For the next stage, the classification error is plotted against the reduced dimension, and in addition, the classification error measurement and the confusion matrices are recorded.

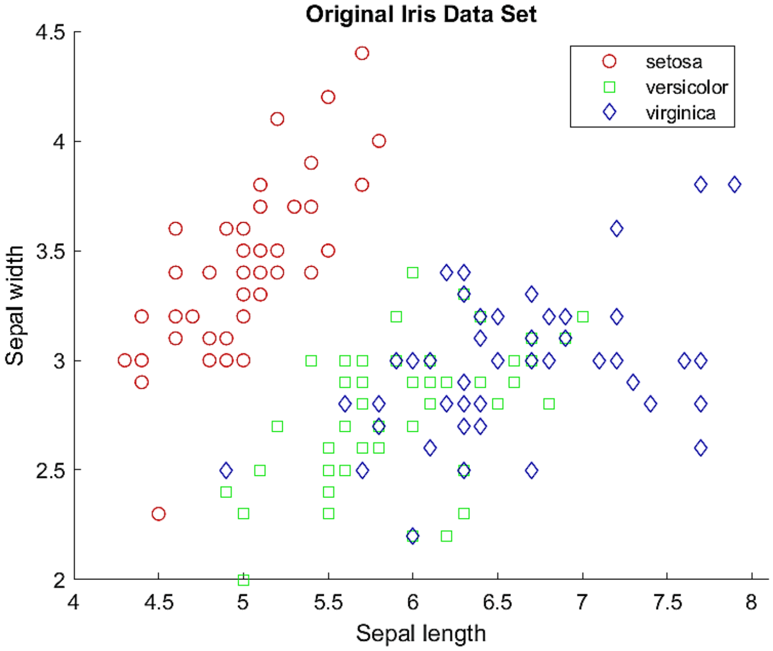


Figure 4.2: Sepal features of the original `fisheriris` datset

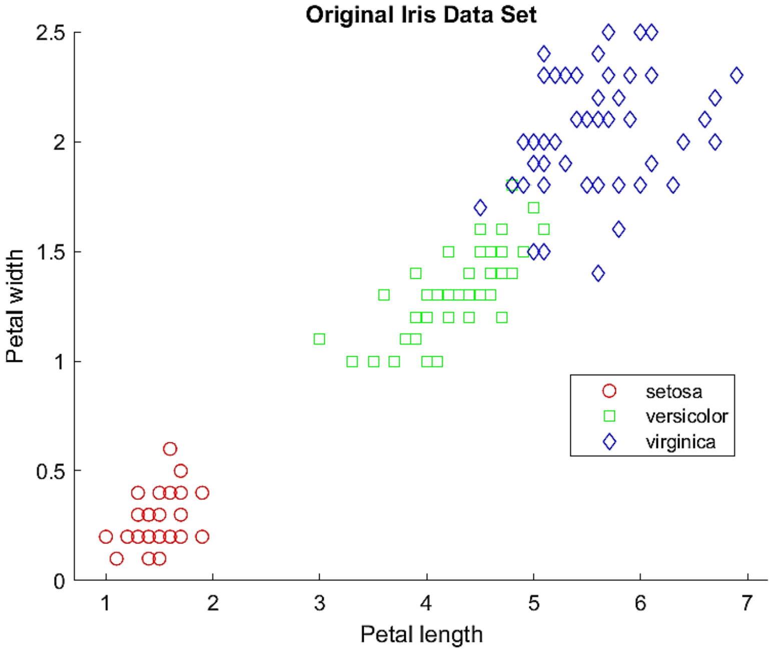


Figure 4.3: Petal features of the original `fisheriris` datset

Chapter 5

Results

This section of the report contains the results obtained from the implementation and the tests mentioned previously in Chapter 4.

5.1 FJLT Method 1

5.1.1 Verification Tests

As previously mentioned for this test, the results obtained were on the behaviour of the probability of success changes with error, the error at which the probability of success reaches 2/3 and also the computation time taken to perform the Φx multiplication. This process was repeated for 20 iterations, and the average was taken.

Initially, the verification test was applied to a simple case where the number of dimensions, $d = 32$ and the sample size, $n = 50$. Figure 5.1 and Figure 5.2 were obtained as the results for this set of parameters. For $d = 32$, the ideal reduced dimension is restricted by $k \leq 3(\approx d^{1/3})$. Hence when $k = 2$ or $k = 3$, the results must show better performance than other reduced dimensions. This can be clearly seen in Figure 5.1, where the red ($k = 3$) line has the best performance in terms of reaching high success probability for low error value, and in Figure 5.2, where $k = 3$ achieves 2/3 probability with the lowest error. The blue ($k = 2$) line also shows decent performance, but it reaches a constant probability of success which is lower than other reduced dimensions, even though it reaches 2/3 probability with the second least error value.

An interesting observation at this stage is that the yellow ($k = 4$) line reaches 2/3 probability with an error not much greater than $k = 3$ and also has good computation time. However, recall that this FJLT method is only valid or instead works well for values $k \leq d^{1/3}$ and for this scenario $k = 4 > d^{1/3}(\approx 3)$. Does this data point represent an anomaly? This will be resolved in the later stages when results are obtained as the value of n is increased.

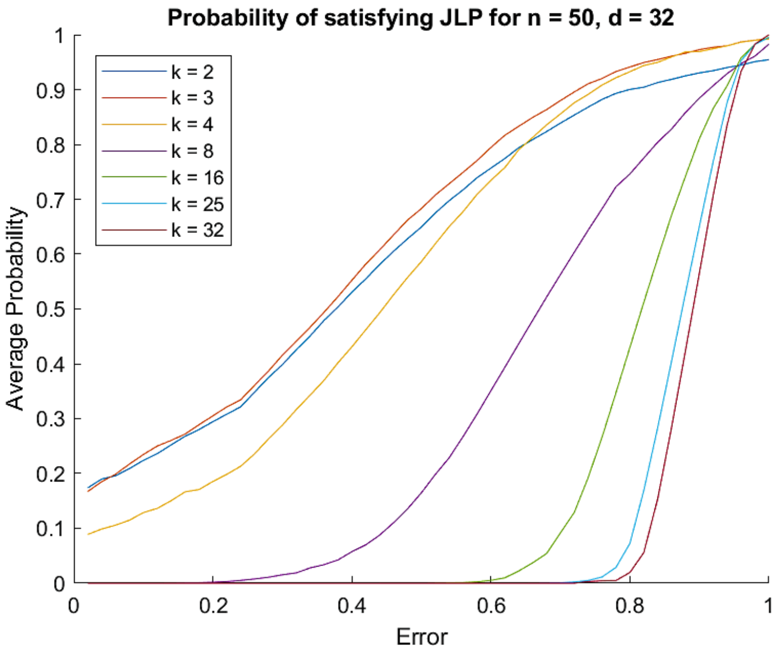


Figure 5.1: Probability of success graph for $n = 50$ and $d = 32$

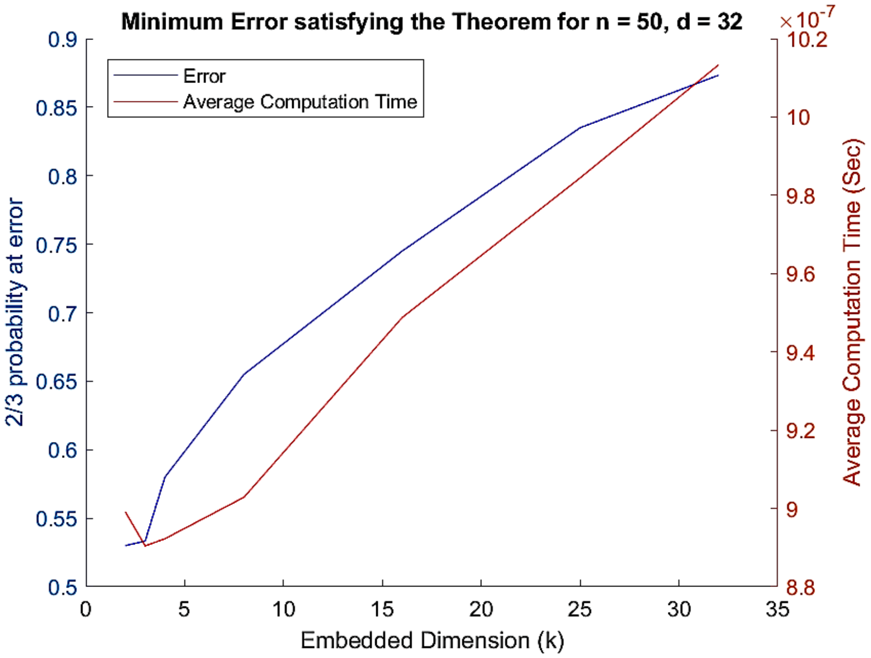


Figure 5.2: Computation time and error required to reach 2/3 probability

For the other reduced dimensions such as when $k \in \{8, 16, 25, 32\}$, the error required to have a decent success probability increases. This is mainly due to the fact that when the k increases while n is kept constant, the sparsity of the sparse matrix will increase because of the additional dimensions and as a result the l_2-norm bounds would not be held for smaller errors. In addition, the computation time increases rapidly as the reduced dimension increase. Recall that this FJLT method has a running time of $O(d \log d + \min\{d\epsilon^{-2} \log n, \epsilon^{-2} \log^3 n\})$ for the euclidean space. For the case of high reduced dimensions, as error increases, the computation time will also increase since both the terms in the latter minimum part has a ϵ^{-2} term. Since the error, $\epsilon < 1$, the ϵ^{-2} term will increase with the error. Thus increasing the overall computation time.

Since the observations and trends for lower sample space seen in Figure 5.1 and Figure 5.2 are justified, the process was then run for a larger sample size of $n = 5000$ to replicate a big data scenario with $d = 32$.

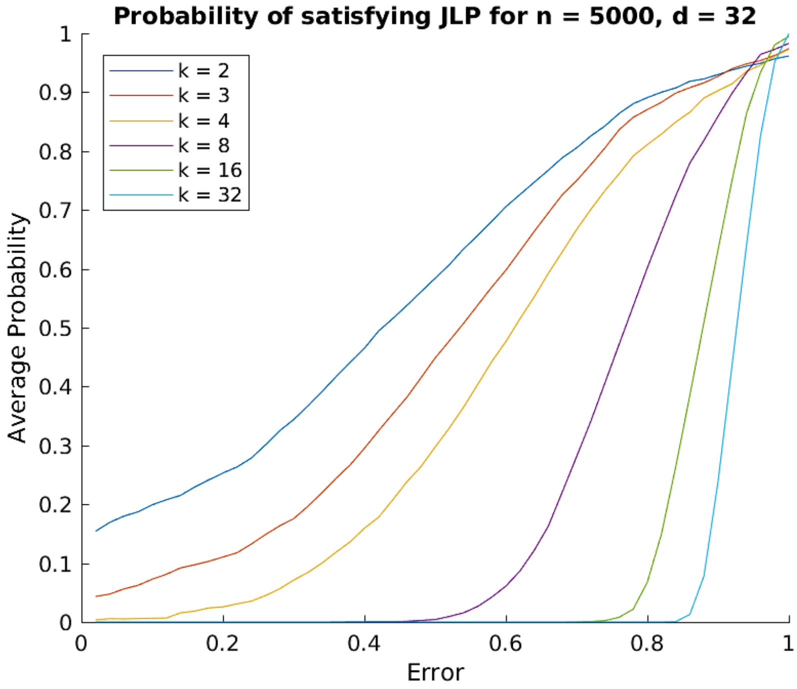


Figure 5.3: Probability of success graph for $n = 5000$ and $d = 32$

In this case, for $n = 5000$ and $d = 32$, the ideal reduced dimension is $k = 2$ (blue line) followed by $k = 3$ (red line). In the previous test, $k = 4$ had decent results in terms of error to reach 2/3 success probability and computation time. However in this scenario, from Figure 5.3 and Figure 5.4 it is clear that the error taken to reach 2/3 success probability is significantly greater than that of the optimal k (which is 2). This can be due to the fact that as the sample size increases, the change in the l_2-norms gets larger and larger. In support, this FJLT method is meant to provide the reduced dimension solution which guarantees low distortion error [3]. Hence having $k = 4 (> d^{1/3})$, will not provide the solution with the lowest distortion.

The other larger reduced dimension values have the same behaviour as to the previous test which is that it requires higher error as well as higher computation time. Hence it can be concluded that for $d = 32$ this FJLT method is valid since it meets the theoretical expectation. The next stage is to carry out the same tests for larger original dimensions such as 64, 128, 256, etc.

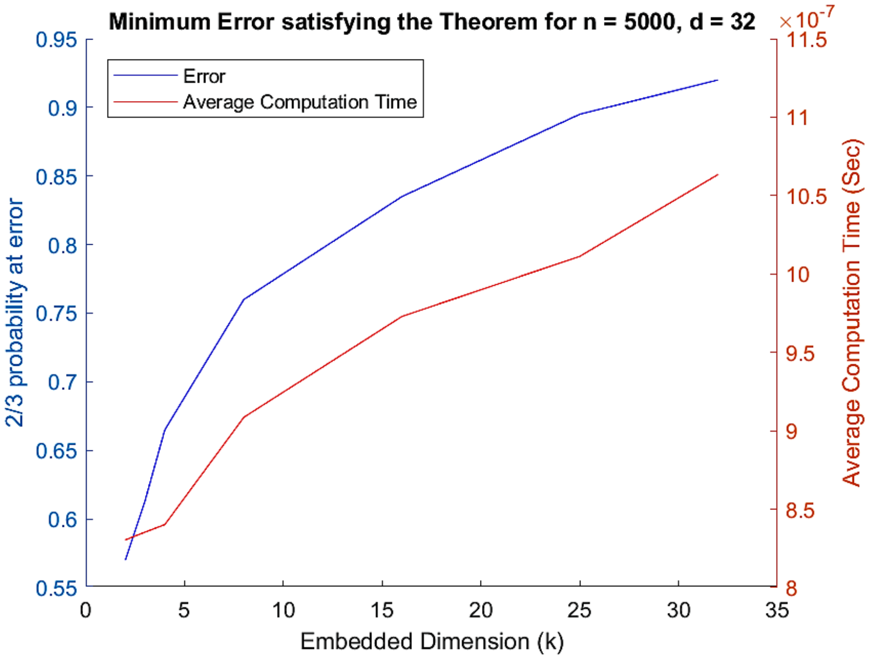
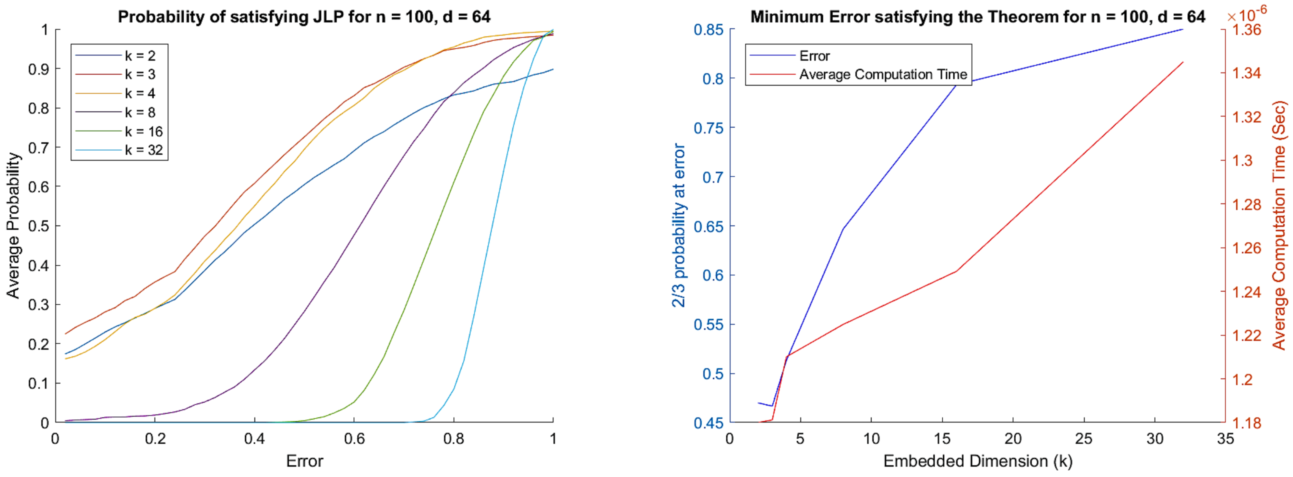


Figure 5.4: Computation time and error required to reach 2/3 probability for $n = 5000$

5.1.2 Parameter Selection

The next step was to make sure that the theoretical expectation is followed for larger dimensions, d. Figure 5.5 shows the results for $d = 64$. In this case, $d^{1/3} = 4$ and hence as seen from the probability of success plot, the best performance is given when $k = 3$ (red line) followed by $k = 4$. The reduced dimension values above 4, has higher error and computation time.

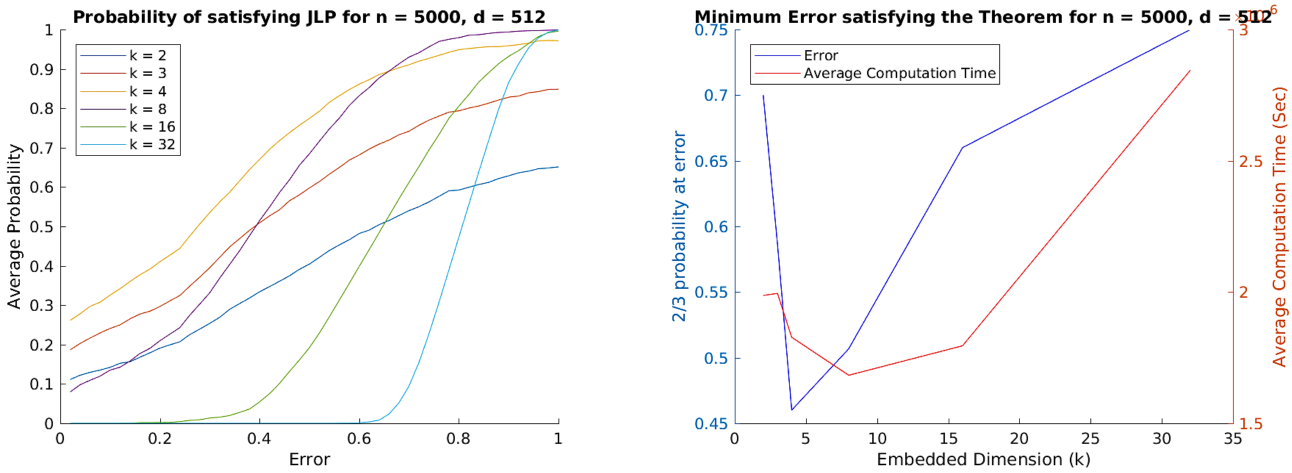


(a) Probability of success graph

(b) Computation time graph

Figure 5.5: Results of the test when $n = 100$ and $d = 64$

Another essential test to carry out is analysing the behaviour of the success probability and computation time when there is a large original dimension space, d and a large sample size. This test can be viewed as applying the projection matrix on a big data scenario. The following figures were the results obtained when $d = 512$ and $n = 5000$.



(a) Probability of success graph

(b) Computation time graph

Figure 5.6: Results of the test when $n = 5000$ and $d = 512$

When $d = 512$, ideally $k \leq 8 (= d^{1/3})$. By observing Figure 5.6a, it is clear that the best solution is provided when $k = 4$ (yellow line) followed by $k = 8$ (purple line). However when $k = 2$ (blue line) or $k = 3$ (red line), the success probability becomes constant at 0.8 and 0.6 respectively. In addition, the computation time for these two k values is relatively high. Hence the optimal solution is obtained when $4 \leq k \leq 8$. The following table shows a summary of results obtained when tests were run with different values for sample size, original dimension, and reduced dimension. The green shaded cells represent the optimal solution in the respective test, and the grey (NaN) cells indicate that 2/3 success probability was not reached.

			k					
d	$d^{1/3}$	n	**2**	**3**	**4**	**8**	**16**	**32**
64	4	1000	0.540	0.540	0.640	0.760	0.880	0.940
		5000	0.460	0.540	0.680	0.820	0.880	0.940
		10000	0.580	0.600	0.720	0.800	0.880	0.940
128	5	1000	0.520	0.420	0.380	0.680	0.800	0.900
		5000	0.500	0.500	0.600	0.740	0.840	0.920
		10000	0.520	0.560	0.620	0.760	0.880	0.920
256	6	1000	0.680	0.520	0.400	0.580	0.720	0.860
		5000	0.600	0.380	0.400	0.640	0.780	0.880
		10000	0.540	0.380	0.440	0.640	0.820	0.900
512	8	1000	NaN	0.740	0.480	0.380	0.620	0.780
		10000	0.780	0.480	0.360	0.520	0.720	0.840
1024	10	10000	NaN	0.840	0.500	0.340	0.600	0.720

Table 5.1: Error at which 2/3 probability is reached for different tests.

As observed from the results of numerous tests, it is clear that an optimal solution can be obtained from the FJLT method 1 which obeys Johnson-Lindenstrauss properties as well as other theoretical expectations. Please refer to the Appendix, to view the summary of results from more tests in addition to results in Table 5.1.

5.2 FJLT Method 2

5.2.1 Verification

For this method, the results obtained were on the behaviour of the success probability against error, the error at which the highest success probability was achieved as well as the runtime time to compute the projected data points. This process was repeated for 50 iterations before computing the average.

Initially, the parameters were set to, sample size, $n = 100$ and original dimension, $d = 128$. The results obtained are shown in Figure 5.7 and Figure 5.8. Unlike the previous FJLT method, this method must provide an optimal solution which approaches a success probability of 1 with the least error. This means that the best projection will always obey the Johnson-Lindenstrauss properties. Hence the error at which the highest success probability is recorded in contrast to the 2/3 probability in FJLT method 1. It is important to note that the Johnson-Lindenstrauss properties for this method are more strict because the error is upper bounded by 0.5, that is $\epsilon < 0.5$, whereas in the previous method, $\epsilon < 1$. Therefore high success probabilities can be achieved with very little error.

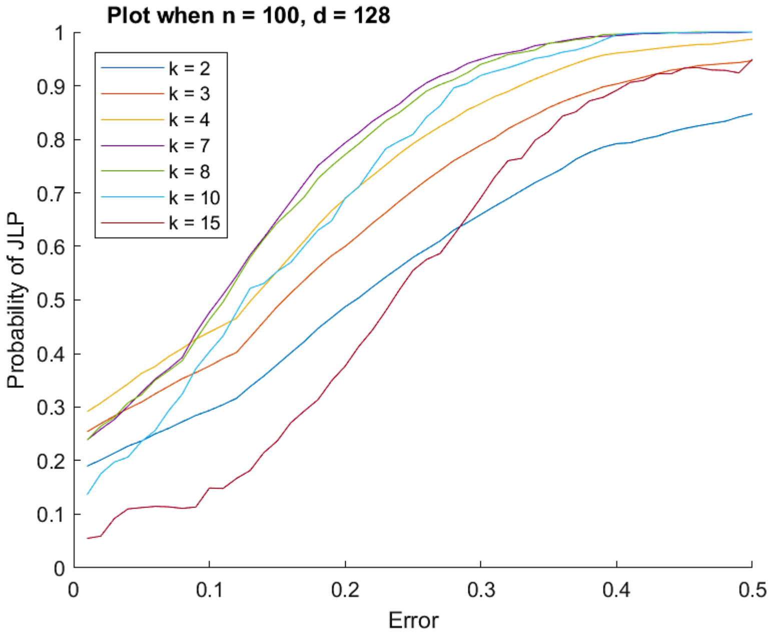


Figure 5.7: Probability of success plot for $n = 100$

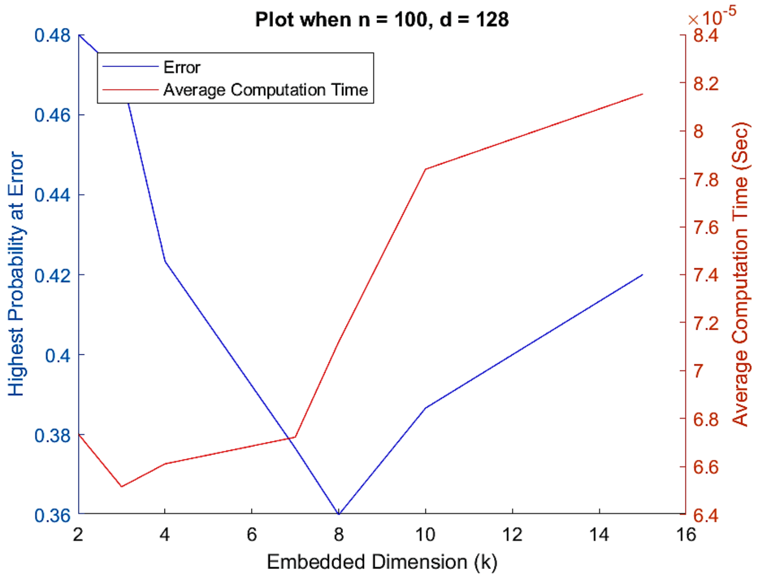


Figure 5.8: Achieving the highest probability and the computation time plot for $n = 100$

This method requires the reduced dimension, $k \leq d^{\frac{1}{2}-\delta}$ for an arbitrary small δ. In this test case, $k \leq 11(\approx d^{1/2})$. By observation, the best performance is gained when $k = 7$ (purple line) which also has the least computation time. From Figure 5.8, it is clear that the computation time increases rapidly for larger k values even though they have potential to achieve high probabilities from low errors. However, the error at which the first occurrence of the highest probability (most of the time, 1) is only recorded. When the averaged probability across 50 iterations is observed (in Figure 5.7), for higher reduced dimensions, the probability stabilises at 1 after the optimal $k = 7$ has stabilised at probability 1. Hence the results from this test obey JL properties as well as follows the theoretical statements of the FJLT method 2.

The test was then run on larger sample size datasets to verify whether the results follow the JL properties and theoretical statements. Figure 5.9 and Figure 5.10 were obtained when the test was run for $n = 5000$ and $d = 256$. Here the optimal performance is obtained when the reduced dimension is $k = 8$ (red line) or $k = 9$ (yellow line) which also produces the result with low computation time. Therefore, this test also provides valid results in terms of obeying JL properties and following theoretical statements. It is essential to note that for large reduced dimensions, the success probability tend to deviate a lot from the mean. This behaviour will be addressed in the later stages.

The next step is to observe the results for a broader range of reduced dimensions and sample sizes.

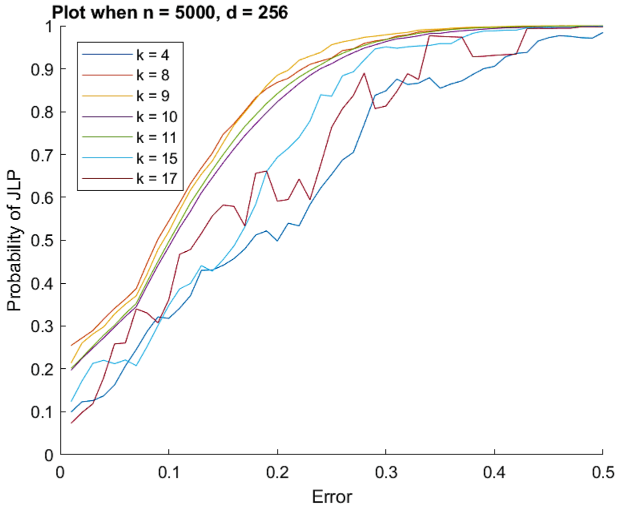


Figure 5.9: Probability of success when $n = 5000$ and $d = 256$

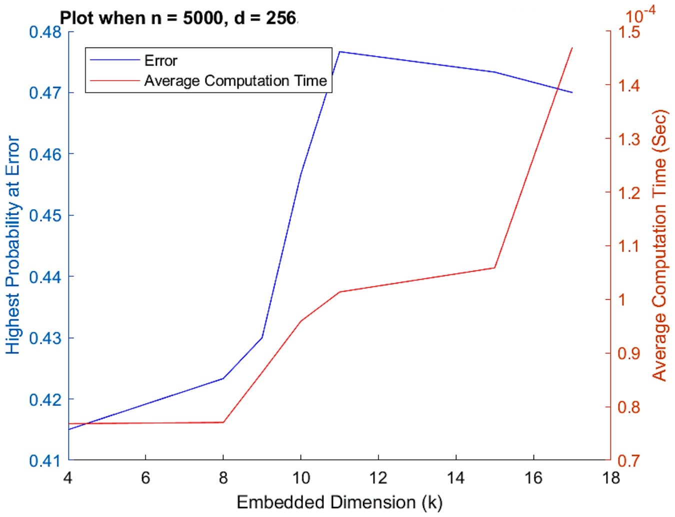
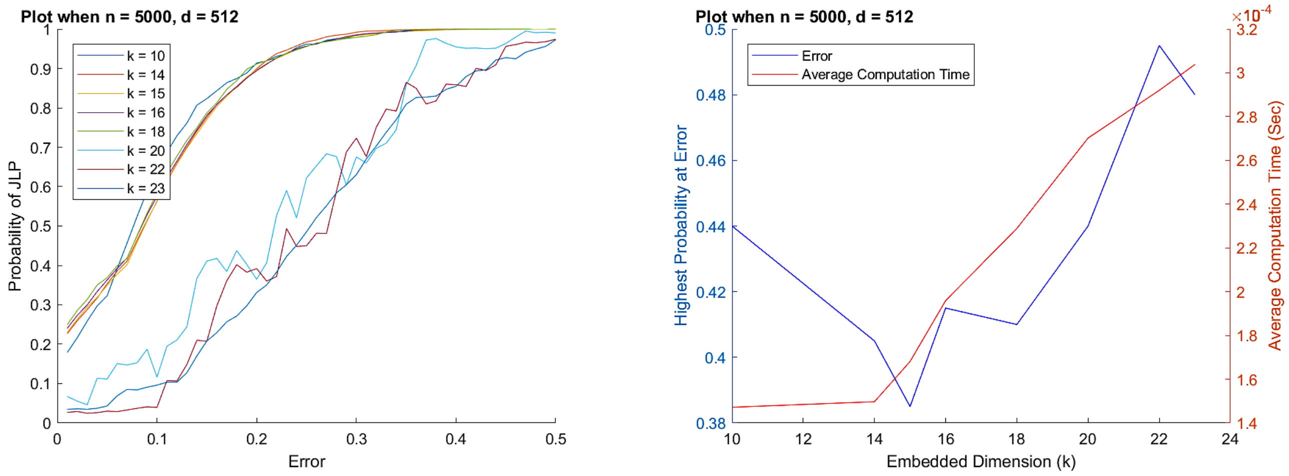


Figure 5.10: Computation time when $n = 5000$ and $d = 256$

5.2.2 Parameter Optimization

The following results were obtained when the original dimension was increased to $k = 512$ for $n = 5000$. By observing Figure 5.6a, it is clear that the FJLT method is not feasible for large reduced dimensions because the trends for $k \in \{20, 22, 23\}$ is not stable and does not maintain a good success probability. Hence an advantage of this FJLT method is that it drives the reduced dimension to smaller values.

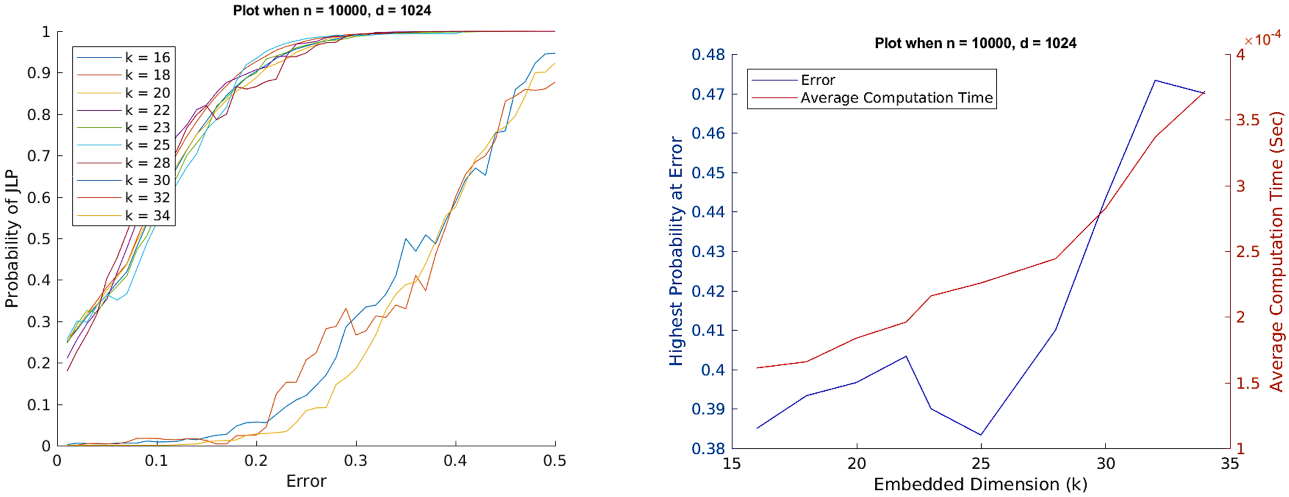
In this case, $k \leq 22(\approx d^{1/2})$, and the optimal performance can be gained when $k \in \{14, 15, 16\}$. Another test that was carried out was to observe the behaviour for very large parameters such as $n = 10000$ and $d = 1024$. This dataset can be used to represent a big data features dataset. The results obtained are represented in the Figure 5.12. Please refer to the Appendix to view a large image of Figure 5.11a and Figure 5.12a.



(a) Probability of success graph

(b) Computation time graph

Figure 5.11: Results of the test when $n = 5000$ and $d = 512$



(a) Probability of success graph

(b) Computation time graph

Figure 5.12: Results of the test when $n = 10000$ and $d = 1024$

The plot shows that the ideal reduced dimension which guarantees low distortion can be any of k between 18 and 25. The best solution is obtained when $k = 25$ because it stabilises at success probability 1 with the least distortion. When looking at the performance of high reduced dimensions such as $k \in \{30, 32, 34\}$ in Figure 5.11a, they tend to have high computation times as well as high distortion. Much variation is seen for large reduced dimensions especially in Figure 5.12a and Figure 5.11a. The FJLT projection does not hold these reduced dimensions because they are above the upper bound of allowed reduced dimensions. Therefore much deviation from the mean is seen when compared to much lower reduced dimensions. For $d = 1024$, the reduced dimension must be $k \leq d^{\frac{1}{2}-\delta}$, to obtain $k = 25$, the δ value must be approximately 0.036. Hence for large datasets, the FJLT method 2 projection obeys JL properties and follows the claims from theoretical analysis.

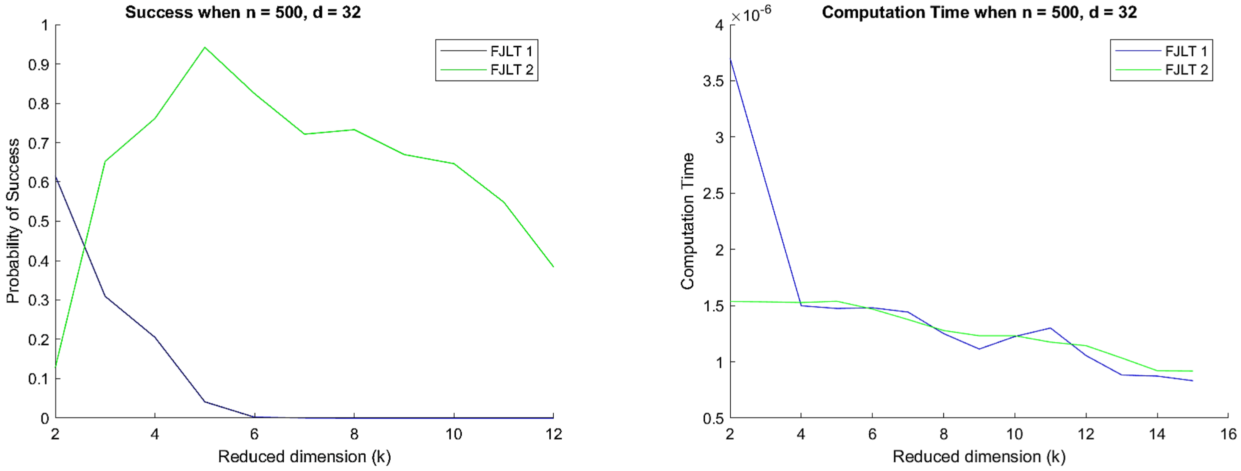
The following table provides a summary of some of the tests that were carried out and recorded the most suitable reduced dimension for each test and the error at which the success probability stabilises at 1.

d	$d^{1/2}$	n	k	β	$error$
32	5.7	50	5	16	0.380
64	8	50	6	32	0.350
128	11.3	500	3	8	0.310
		5000	3	8	0.330
256	16	50	8	64	0.340
		500	8	64	0.330
		10000	8	64	0.360
512	22.6	1000	18	256	0.330
		10000	18	256	0.300

Table 5.2: Best performing reduced dimension solution for different tests

5.3 Comparison of the Two Projection Methods

This section contains the results of the tests that were carried out to distinguish the two FJLT methods. The results were obtained by selecting a dataset which was applied to both FJLT projections. The primary focus for these tests is the behaviour of the probability of success against the reduced dimension and their respective computation times. The process was repeated for 10 iterations, and an average was taken. Initially, small sample size cases were tested.

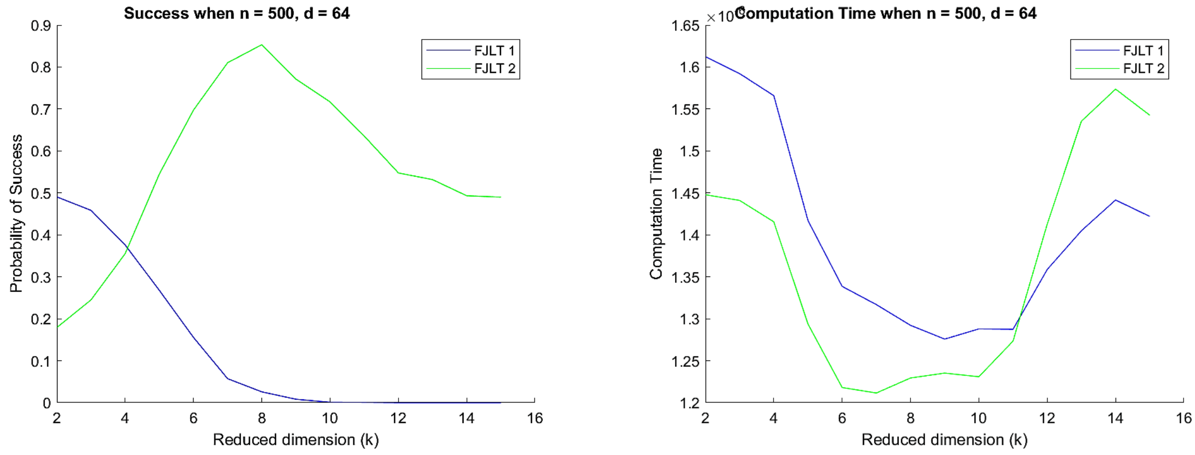


(a) Probability of success graph

(b) Computation time graph

Figure 5.13: Results of the test when $n = 500$ and $d = 32$

In Figure 5.13a, the best solution from FJLT method 1 is given when $k = 2$ with a success probability of 0.6. However, FJLT method 2 provides a better solution when $k = 5$ with a success probability of over 0.9. In addition, FJLT 2 provides a solution with 2.5 times better computation time than FJLT 1 with respect to their optimal reduced dimensions as seen in Figure 5.13b. Hence for small sample sizes, FJLT method 2 guarantees better performance than FJLT 1. Comparison tests were next carried out for a higher dimension of $d = 64$.

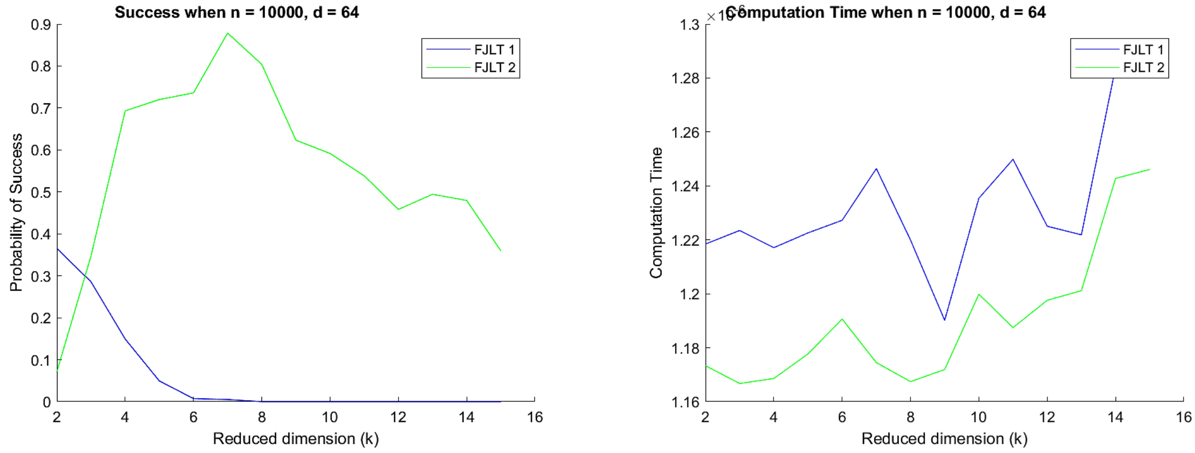


(a) Probability of success graph

(b) Computation time graph

Figure 5.14: Results of the test when $n = 500$ and $d = 64$

Similar to the previous test, FJLT 2 provides the best success probability of 0.85 for $k = 8$ compared to 0.5 success probability at $k = 2$ for FJLT 1. FJLT 2 method has 1.5 times better computation time than FJLT 1. It is interesting to notice that, beyond a specific reduced dimension, the computation time of FJLT method 2 becomes more than that of FJLT method 1. However, the results from FJLT 1 after $k = 10$ is considered to be invalid since the JL properties are not obeyed, and hence the FJLT 1 theorems are not followed. The next test scenario was to examine the behaviour for very large datasets such as $n = 10000$ in order to claim that the FJLT method 2 guarantees low distortion and better runtime over FJLT 1.



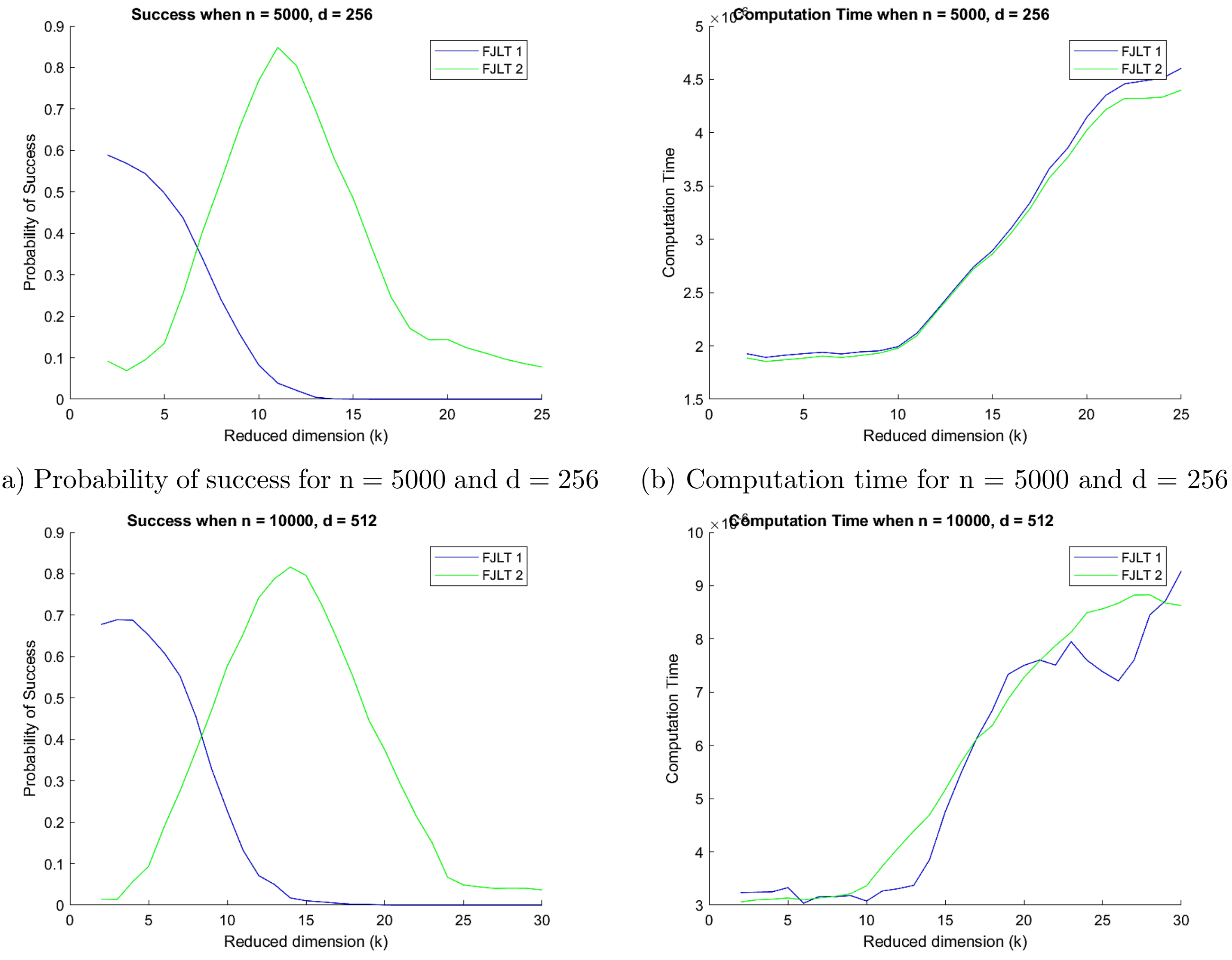
(a) Probability of success graph

(b) Computation time graph

Figure 5.15: Results of the test when $n = 500$ and $d = 64$

The observations that can be made for this test is no different to the previous tests. The FJLT method 2 provides a lower computation time with a 0.9 success probability. The FJLT method 2 guarantees a better solution than FJLT 1 mainly because of the distortion and the runtime. Recall that the distortion for FJLT 2 is defined as $\epsilon < 0.5$, whereas the distortion for FJLT 1 is $\epsilon < 1$. Therefore the FJLT method 2 is bound to give results with better accuracy. The runtime for computing Φx for FJLT method 1 is of $O(d \log d)$, whereas for FJLT method 2, it is defined as $O(d \log k)$. Hence FJLT 2, trims the computation time from $O(d \log d)$ to $O(d \log k)$ because the multiplication of the matrices is done in smaller sizes separately. Therefore it can be claimed that FJLT method 2 guarantees better performance.

The following graphs were obtained when the projection was simulated for datasets which replicate big data scenarios. It was observed that the difference between the computation times of the 2 methods becomes small. This can be due to the high complexity of the dataset which drives the difference in computation time between reduced dimensions to be large. However, the results of these tests support the previously stated claims.



(a) Probability of success for n = 5000 and d = 256

(b) Computation time for n = 5000 and d = 256

(c) Probability of success for n = 10^4 and d = 512

(d) Computation time for n = 10^4 and d = 512

Figure 5.16: Results of the tests with large datasets

5.4 Machine Learning Application

5.4.1 K-NN for Random Samples

In this section, the two FJLT projection methods were applied on a 32-dimensional dataset used for classification. The machine learning technique used for this was the k-Nearest Neighbours search algorithm. The first two columns of each dataset were used for the graphical representation of the data. Figure 5.17 shows the original normalised dataset which will be used for analysis with FJLT projections.

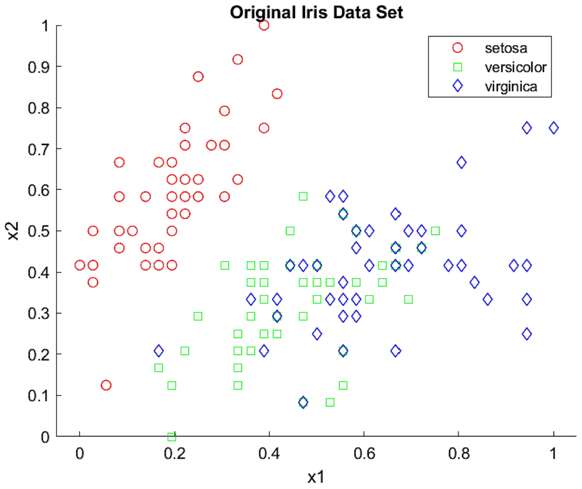


Figure 5.17: Original dataset normalised

In this test, 5 samples were selected at random, and their respective class was predicted depending on the 10 nearest neighbouring samples. To visualise how the structure of data changes when the projection is performed, the following graphs were obtained. The plot of query points on the original dataset is represented in Figure 5.18, and the respectively projected datasets are represented in the graphs of Figure 5.19.

It can be deduced that the isometric properties of the dataset are likely to change as seen in the below figures. This is because the FJLT methods are distance preserving given a small distortion whereas isometry is strict distance preserving. However, it is interesting to examine the correctness of these transformed datasets. Hence the actual and predicted results in terms of classification were recorded, and the test was repeated for 10 iterations to obtain the average accuracy.

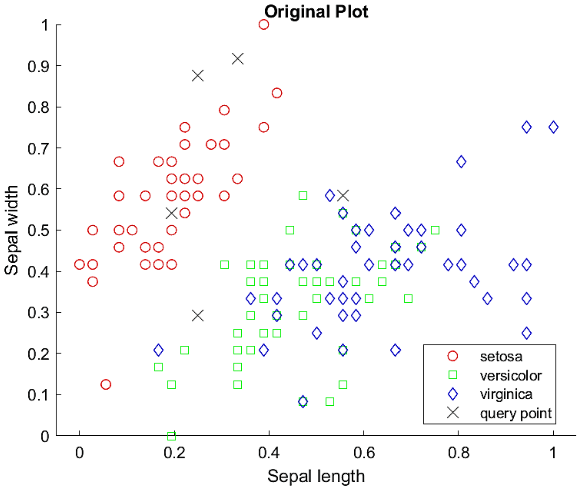
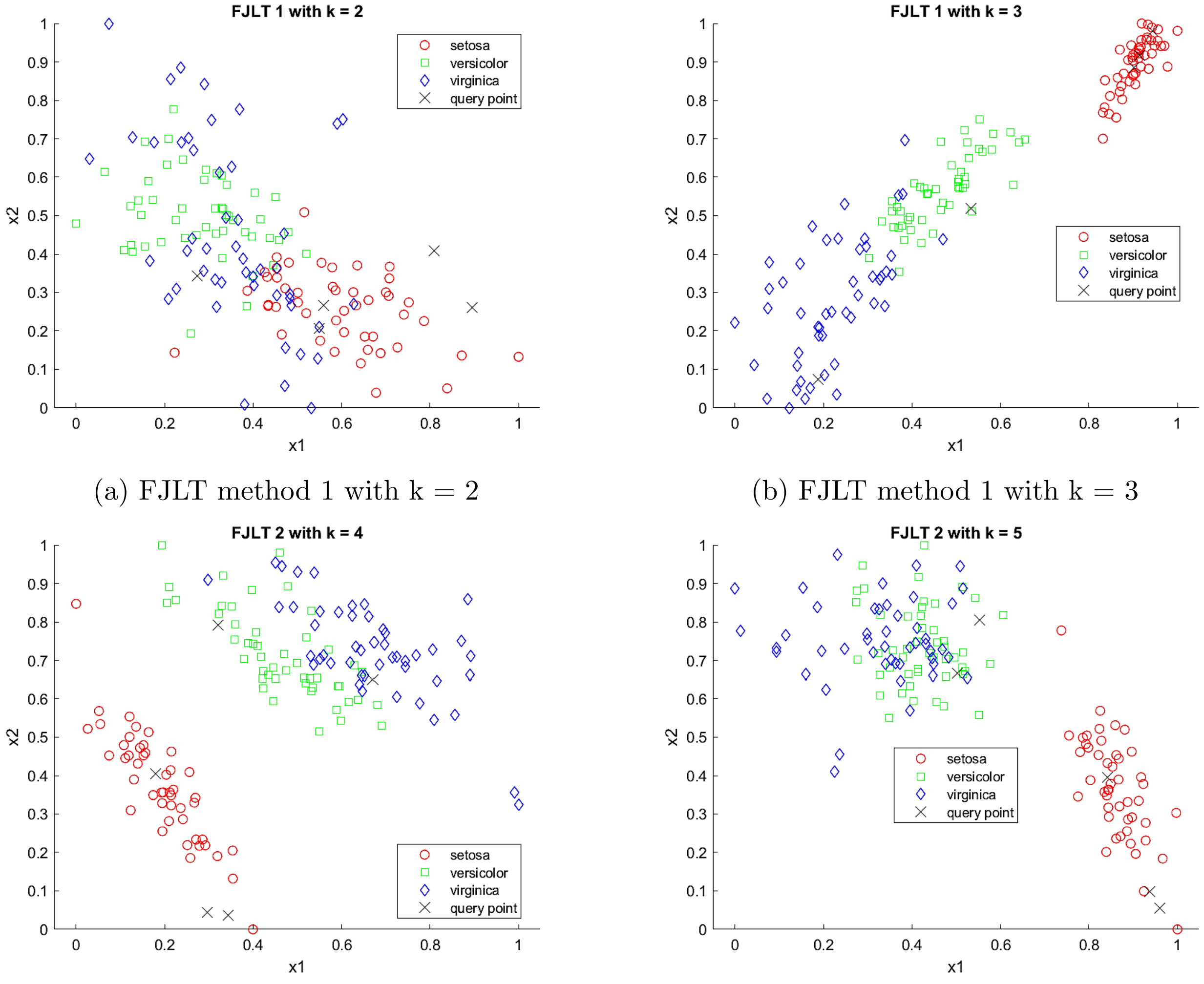


Figure 5.18: Original plot with query points



(a) FJLT method 1 with k = 2

(b) FJLT method 1 with k = 3

(c) FJLT method 2 with k = 4

(d) FJLT method 2 with k = 5

Figure 5.19: Reduced dimensional results

Type	*d*	*k*	1: *versicolor*, 2: *setosa*, 3:*virginica*		**Accuracy Over 10 Tests**
			Actual (for this test)	**Predicted (for this test)**	
Original	32		[2, 1, 2, 2, 3]	[2, 1, 2, 2, 3]	100%
FJLT 1	32	2	[2, 1, 2, 2, 3]	[2, 3, 2, 2, 2]	80%
FJLT 1	32	3	[2, 1, 2, 2, 3]	[2, 1, 2, 2, 3]	96%
FJLT 2	32	4	[2, 1, 2, 2, 3]	[2, 1, 2, 2, 3]	96%
FJLT 2	32	5	[2, 1, 2, 2, 3]	[2, 1, 2, 2, 3]	100%

Table 5.3: Table of the results obtained from Figure 5.18 and Figure 5.19

By observing Table 5.3, it can be claimed that even though the isometric properties has changed, the FJLT projection methods guarantee correctness and accuracy of the data. Hence it is important to note that the projection does not lose the generality underlying in the dataset. Formally, more tests were carried out with larger reduced dimensions and larger test sample sizes. The following table provides a summary of the critical data that were obtained from these tests.

Type	**Test Size**	**Accuracy (as per dataset) %**									
		Original	**k**								
			2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
FJLT 1	5	80.0	78.0	84.0	88.0	86.0	90.0	92.0	90.0	84.0	92.0
FJLT 2	5	80.0	86.0	86.0	84.0	88.0	86.0	90.0	90.0	86.0	86.0
FJLT 1	10	90.0	89.0	88.0	92.0	90.0	91.0	93.0	94.0	92.0	91.0
FJLT 2	10	90.0	92.0	87.0	85.0	87.0	94.0	92.0	93.0	94.0	92.0
FJLT 1	15	100.0	94.0	94.0	100.0	99.3	100.0	100.0	100.0	100.0	100.0
FJLT 2	15	100.0	90.0	95.3	98.0	97.3	100.0	100.0	99.3	99.3	100.0

Table 5.4: Summary of correctness tests

By observing Table 5.4, it is clear that the correctness of the dataset is maintained even though the isometry changes when the projection is computed. As the reduced dimension increases, the accuracy of the results obtained has relatively increased compared to the lower dimensions. This is mainly due to the fact that as the dimensions increase, more information is carried through and hence that extra bit of information drives the accuracy to higher values.

5.4.2 Cross-Validation Tests

For this test, the dataset was divided into 10 folds where 8 folds were used for training, 1 fold for validation, and 1 fold used for testing. This process was iterated 10 times where a different fold was selected for testing in each iteration. Please refer to the appendix to view the breakdown of the dataset. This 10-fold cross-validation was applied to both projection methods, and at each reduced dimension, the classification error was recorded. In addition, confusion matrices for some reduced dimensions were also recorded. The confusion matrix was computed in order to visualise each projection method's performance, where it indicates the distribution of the data points into different classes.

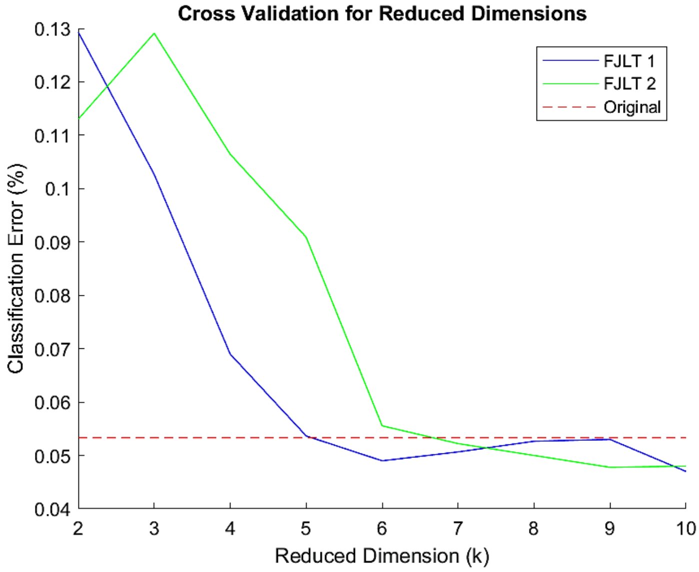


Figure 5.20: 10-fold cross validation for reduced dimensions

As seen in Figure 5.20 the classification rate decreases with the reduced dimension value. The red dotted line represents the classification error obtained when the 10-fold cross-validation was applied to the original dataset.

Type	**k**	**Classification Error**
Original		0.060
FJLT 1	2	0.200
FJLT 1	3	0.130
FJLT 2	4	0.053
FJLT 2	5	0.033

Table 5.5: Classification errors

The above table shows the classification error of the respective datasets. For this test, it was observed that the data projected by FJLT method 2 produced great results compared to the original dataset. However, as seen in Figure 5.20, on average the projected data has similar performance to the original dataset for higher reduced dimensions. The confusion matrices of some reduced dimensions obtained through testing are given below.

		Actual Class		
		setosa	**versicolor**	**virginica**
Predicted Class	**setosa**	50	0	0
	versicolor	0	46	5
	virginica	0	4	45
	undefined	0	0	0

Table 5.6: Confusion matrix for original dataset

		Actual Class		
		setosa	**versicolor**	**virginica**
Predicted Class	**setosa**	49	0	0
	versicolor	1	43	22
	virginica	0	7	28
	undefined	0	0	0

Table 5.7: Confusion matrix for FJLT 1 projection with k = 2

		Actual Class		
		setosa	**versicolor**	**virginica**
Predicted Class	**setosa**	50	4	0
	versicolor	0	41	11
	virginica	0	5	39
	undefined	0	0	0

Table 5.8: Confusion matrix for FJLT 1 projection with k = 3

		Actual Class		
		setosa	**versicolor**	**virginica**
Predicted Class	**setosa**	50	0	0
	versicolor	0	48	6
	virginica	0	2	44
	undefined	0	0	0

Table 5.9: Confusion matrix for FJLT 2 projection with k = 4

		Actual Class		
		setosa	**versicolor**	**virginica**
Predicted Class	**setosa**	50	0	0
	versicolor	0	47	2
	virginica	0	3	48
	undefined	0	0	0

Table 5.10: Confusion matrix for FJLT 2 projection with k = 5

*The undefined attribute is present in order to represent any inconclusive results.

From these confusion matrices, it can be said that the least confused class is *setosa*, where most of the time these data points are correctly classified. Subsequently, the *virginica* class is the most confused class because some of it data points are wrongly classified.

Implementing and testing the FJLT methods in machine learning applications thus verifies the correctness of the data projection and transformation.

Chapter 6

Evaluation

6.1 Distortion Error

As discussed in Chapter 5, the distortion error becomes very important in projections. It defines how the isometry of the dataset can change and to what extent. This allows the dataset to maintain its generality by not losing much information, as seen in Section 5.4. One of the goals of the two FJLT projection methods was to guarantee a low distortion solution for low reduced dimensions. At each stage of testing, the most appropriate reduced dimension was discussed, which produced the dataset which satisfies the JL properties with the least error. It was observed that when data becomes more complex, the distortion error tends to be more. However with FJLT method 2, always guarantees a reduced dimension which satisfies JL properties with a distortion error, $\epsilon < 0.5$ which is less than the Johnson-Lindenstrauss theorem with restriction of $\epsilon < 1$. Through testing, the FJLT method 2 produced the reduced dimension with the least distortion compared to FJLT method 1.

6.2 Reduced Dimension

One of the objectives of this project was to evaluate the lower bounds of the projection methods. In each theorem of the FJLT projection methods, the restriction on the reduced dimension was stated. Formally in Chapter 5, those restrictions were tested, and it was clear that the reduced dimension, $k \leq d^{1/3}$ for the FJLT method 1 and $k \leq d^{\frac{1}{2}-\delta}$ for the FJLT method 2. Going beyond these limits the distortion error becomes high, and the JL properties were not met, and hence low distortion for the reduced dimension was not guaranteed. The FJLT method 1 is useful for instances where the dataset projected must be of very low dimension, for a larger reduced dimension, the FJLT method 2 becomes useful. Thus the performance based on the reduced dimension will depend on the projection requirements on how low the reduced dimension must be.

6.3 Computation Time

The computation time is one of the main focus points in in the random projection area. The previous JL random projection had a runtime of $O(kn)$. The fast Johnson-Lindenstrauss transform methods have an immediate improvement on the computation time. The FJLT method 1 is able to maintain a computation time of $O(d \log d)$ through sparse matrices. The FJLT method 2 can maintain a computation time of $O(d \log k)$ through error correcting codes and breaking down large matrices. The difference was clearly identified throughout the testing phase analysis in Chapter 5, where the FJLT 2 optimal reduced dimension has a computation time substantially better than that of FJLT method 1. As seen in Section 5.3, for very larger datasets with a sample size over 10,000 and dimension size over 1000, the difference in the computation time becomes small. However as the reduced dimension increases, the computation also rapidly increases. Hence the difference in the two methods becomes hard to distinguish. Overall, the FJLT method 2 is bound to have a better computation time than FJLT 1.

6.4 Practical Applications

Applying the FJLT methods on a machine learning application proved that the projection maintains the generality and correctness of the data. The results of the cross-validation tests and k-nearest neighbours search algorithm showed great outcomes with over 90% classification rate, and for some occasions, it produced a better classification rate than the original dataset. These tests proved the use of FJLT methods could be used to project very large datasets onto a reduced dimension space that would eliminate the intrinsic statistical difficulty and computation difficulty due to the curse of dimensionality.

The tests on large and complex datasets (which has over 10,000 samples and over 1,000 dimensions) and machine learning applications were carried out in order to demonstrate the FJLT projection methods on big data applications. The FJLT methods can be viewed as a data compressing technique which is used in many applications. For instance, in image processing, few pixels are chosen at random to be excluded from the dataset for faster computation and efficient storing. Similarly, another area where FJLT methods can be used is in compressed sensing.

6.5 Summary

The main advantage of FJLT projections is the elimination of the curse of dimensionality. Since the dimensionality is reduced, the number of unit balls required to cover the whole space is reduced, thus decreasing the number of training samples required. This eliminates the data isolation aspects. In addition, the dimension reduction avoids false structure because the number of samples is now significantly higher than the dimension size. This drives the empirical covariance matrix of the sample set to the identity matrix, thus stabilising the structure of the data. A highlight improvement that dimension reduction provides is avoiding overfitting due to the high data complexity. The complexity of the data is significantly reduced, and hence the generality is increased. In addition to eliminating the intrinsic statistical difficulty, it also addresses the computational difficulties. A standard machine can use the reduced dimensional data for computation and analysis, and it is efficient in terms of storing and transferring.

	Technique	**Reduced Dimension Restriction**	**Computation Time**
FJLT 1	Sparse reduction	$k \leq d^{1/3}$	$O(d \log d)$
FJLT 2	Dual BCH codes	$k < d^{1/2}$	$O(d \log k)$

Table 6.1: Fast JL projection methods

The above table summarises the main characteristics of the two fast Johnson-Lindenstrauss methods implemented in this project. Overall the two methods provide exceptional results for any dataset type. However, when the properties such as computation time and distortion error are considered, FJLT method 2 provides a better random projector.

Chapter 7

Future Work

7.1 Other FJLT Methods

Johnson-Lindenstrauss Transforms and Restricted Isometry Property

By using sparsity and error-correcting codes, it allowed performing dimension reduction transforms. However, the embedding dimension k was restricted to values below $d^{\frac{1}{2}}$. To perform transforms beyond $d^{\frac{1}{2}}$, the restricted isometry property from compressed sensing must be introduced. The restricted isometry property preserves the l_2-norm of a matrix. The following theorem defines the restricted isometry property [35].

Theorem 7.1 *A matrix A in $\mathbb{R}_{m\times n}$ is said to satisfy the Restricted Isometry Property (RIP) with parameters (s, δ), if for all $\mathcal{T} \subset [n]$ such that $|\mathcal{T}| \leq s$ and for all $q \in \mathbb{R}^{|\mathcal{T}|}$, it holds that,*

$$(1-\delta)\|q\|_2^2 \leq \|A_{\mathcal{T}}q\|_2^2 \leq (1+\delta)\|q\|_2^2 \tag{7.1}$$

Theorem 7.2 *The Restricted Isometry Constant δ_s is defined as the smallest constant δ for which the s-RIP holds,*

$$\delta_s = \inf\{\delta : (1-\delta)\|q\|_2^2 \leq \|A_{\mathcal{T}}q\|_2^2 \leq (1+\delta)\|q\|_2^2 \quad \forall|\mathcal{T}| \leq s, \forall q \in \mathbb{R}^{|\mathcal{T}|}\} \tag{7.2}$$

By inspection, s represents a certain level of sparsity in the matrix A. This means that certain s rows are chosen from A. With the Equation 7.1, it tries to preserve the l_2-norm, which is the length. Hence the term isometry because it preserves the lengths. The smallest δ that can be used is known as the restricted isometry constant which is represented by Equation 7.2. The meaning of this is that when a vector q gets multiplied by a matrix A, the length would change. However, if s-RIP submatrix of A is used, the change in the length is significantly small because it is bounded by $(1 \pm \delta)$ [35]. If the restricted isometry property is represented in a fast RIP matrix, this would be useful to perform fast recovery of sparse signals.

Using this approach, the RIP construction method was introduced. It was initially used for sparse signal recovery and later was used for dimensional reduction [36]. The method was to have a Fourier transform matrix (Walsh-Hadamard matrix) and then chose k rows of that matrix at random. This resulted in forming a matrix which a $k \times d$ RIP matrix with $\left(s, \delta = \left((s \log^4 d)/k\right)^{\frac{1}{2}}\right)$ [36]. To make the matrix suitable for random dimension reduction projections, the $d \times d$ random (± 1) diagonal matrix was added to the end. Similar to fast JL method 1, this has the Fourier matrix and the random diagonal matrix. In addition, it has the same runtime performance of $O(d \log d)$ [37]. However, the embedding dimension was now in terms of the number of samples n, giving $k \approx \epsilon^{-4} \log^4 n$ [37]. The following indication represents the construction of a projection matrix for the fast JL based on the restricted isometry property.

$$\underbrace{k \begin{matrix}\rightarrow\\ \rightarrow\\ \rightarrow\\ \rightarrow\end{matrix} \begin{pmatrix} & & \\ & H & \\ & & \\ & & \end{pmatrix}}_{\text{RIP}(k\times d)} \underbrace{\begin{pmatrix}\pm1 & 0 & \cdots & 0\\ 0 & \pm1 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & \pm1\end{pmatrix}}_{\text{Random Diagonal Matrix}(d\times d)}$$

Sparse Dimension Reduction

More recently, Prof. Jelani Nelson contributed towards the performance bounds of random dimension reduction projectors [38, 39, 40]. Most of the concepts were based on sparsity. One of the methods he defined was that for a projection matrix of $k \times d$, each column of the matrix has a limit for the number of non-zero elements it can have. This provides the same upper-bound as the sparse JL transform in Method 1. However using this theory on a RIP matrix, resulted in forming the optimal RIP matrix. Here the approach is for having sparse columns rather than rows [39]. The new approach produced excellent results in improving the bounds of a JL transform, where the lower-bound is now non-restricted. The sparse JL transform was then tested and was able to obtain favourable results in areas such as linear algebra, compressed sensing and other statistical analysis applications [40].

7.2 Future Work

Further simulations could be carried out in areas such as datasets that represent a large number of pixels of images. For instance, in an emotion detection algorithm, it would be interesting to observe the classification results of the reduced dimensional images and the original images. This would be beneficial because much research is being carried out in areas such as smart recognition and security-related applications.

The current fast JL transforms have a restricted upper bound of $d^{1/2}$ on the reduced dimension. Hence further research is to increase this upper bound in addition to improving the computation time beyond $O(d \log k)$. To obtain these results, the restrictions and limitations on the theory must be eliminated or trimmed. The work on JL transforms with techniques such as restricted isometry, and sparse dimension reduction has shown potential to exceed the reduced dimension and would be the next step in JL projection.

Chapter 8

Conclusions

The primary focus of this project was to implement two random projection methods that would project the high dimensional data onto a low dimensional subspace in order to yield low complexity solutions. The Johnson-Lindenstrauss transform is the fundamental method for dimension reduction and this project successfully presented the implementation of two methods of fast Johnson-Lindenstrauss transform.

FJLT method 1 used techniques such as sparsity and Fourier transform to produce a random projector with a runtime of $O(d \log d)$ for reduced dimension values $k \leq d^{1/3}$. However, the FJLT method 2 produced a random projector with runtime $O(d \log k)$ for reduced dimension values $k < d^{1/2}$. This method used techniques from error correcting codes and BCH codes. In Chapter 5, the significant difference in performance of the two methods was identified and discussed. It was concluded that for very low reduced dimensions, the FJLT 1 would be more beneficial in some instances. On average, FJLT method 2 was proved to be the best random projector that guarantees low distortion. However, the upper bound on the reduced dimension is restricted by $d^{1/2}$. Any reduced dimensions above this would result in false structure and hence high distortion which would lead to data being incorrect.

Preparation of the FJLT method 2 projector was a highlight of the implementation stage. Though the theorems seem simple, the theory behind the implementation was complicated, but creating the projection matrix was efficient. The result of the break down of large matrices showed significantly on the computation time comparisons in Chapter 5. Thus making FJLT 2 the better performing random projector over FJLT 1. The random projectors were focused on the Euclidean space, but they also have potential feasibility in the Manhattan space as well [3, 31].

Applying the random projectors on machine learning application proved the feasibility of the methods in practice. The tests showed that even though the isometry of the data is changed, the generality is preserved, which leads to accurate results. However, a limitation of these methods in practice is that it is not recommended for very sensitive data where even the slightest data loss cannot be tolerated in the system.

To conclude, dimension reduction can be achieved through fast Johnson-Lindenstrauss transforms with techniques including sparsity and coding theory. These methods have excellent computation time, and it is also feasible in big data applications in practice.

Bibliography

[1] Rajaraman, V. (2016). Big data analytics. Resonance, [online] 21(8), pp.695-716. Available at: `https://link-springer-com.iclibezp1.cc.ic.ac.uk/article/10.1007/s12045-016-0376-7` [Accessed 19 Jun. 2018].

[2] Computerhistory.org. (2018). Memory & Storage | Timeline of Computer History | Computer History Museum. [online] Available at: `http://www.computerhistory.org/timeline/memory-storage/` [Accessed 31 May 2018].

[3] Ailon, N. and Chazelle, B. (2010). *Faster dimension reduction.* Communications of the ACM, [online] 53(2), pp.97-104. Available at: `https://www.cs.princeton.edu/~chazelle/pubs/fasterdim-ac10.pdf` [Accessed 19 Jan. 2018].

[4] Stephens, Z., Lee, S., Faghri, F., Campbell, R., Zhai, C., Efron, M., Iyer, R., Schatz, M., Sinha, S. and Robinson, G. (2015). *Big Data: Astronomical or Genomical?.* PLOS Biology, [online] 13(7), p.e1002195. Available at: `http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1002195` [Accessed 22 Jan. 2018].

[5] En.wikipedia.org. (2018). *Big Data.* [online] Available at: `https://en.wikipedia.org/wiki/Big_data` [Accessed 15 Jan. 2018].

[6] Bingham, E. and Mannila, H. (2001). *Random projection in dimensionality reduction: Applications to image and text data.* Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01. [online] Available at: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.8124&rep=rep1&type=pdf` [Accessed 22 Jan. 2018].

[7] Dai, W. (2017). *Compressed Sensing from Topics in Large Dimensional Data Processing* [online] Available at: `http://www.ee.imperial.ac.uk/wei.dai/Teaching/EE4.66.2017/00EE4.66_part_1.pdf` [Accessed 31 May 2018].

[8] Gao, P. and Ganguli, S. (2015). *On simplicity and complexity in the brave new world of large-scale neuroscience.* Current Opinion in Neurobiology, [online] 32, pp.148-155. Available at: `https://www.sciencedirect.com/science/article/pii/S0959438815000768` [Accessed 27 Jan. 2018].

[9] Oymak, S. and Tropp, J. (2015). *Universality laws for randomized dimension reduction, with applications.* 2010 Mathematics Subject Classification, [Online] 3. Available at: `https://arxiv.org/pdf/1511.09433.pdf` [Accessed 27 Jan. 2018].

[10] Wu, J., Cui, Z., Sheng, V., Shi, Y. and Zhao, P. (2014). *Mixed Pattern Matching-Based Traffic Abnormal Behavior Recognition.* [Image] Available at: `https://www.researchgate.net/publication/260612049_Mixed_Pattern_Matching-Based_Traffic_Abnormal_Behavior_Recognition` [Accessed 25 Jan. 2018].

[11] En.wikipedia.org. (2018). *K-nearest neighbors algorithm.* [online] Available at: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm [Accessed 25 Jan. 2018].

[12] Beyer, K., Goldstein, J., Ramakrishnan, R. and Shaft, U. (1999). *When Is "Nearest Neighbor" Meaningful?.* Database Theory — ICDT'99, [online] 1540, pp.217-235. Available at: https://link.springer.com/content/pdf/10.1007\%2F3-540-49257-7_15.pdf [Accessed 25 Jan. 2018].

[13] En.wikipedia.org. (2018). Euclidean space. [online] Available at: https://en.wikipedia.org/wiki/Euclidean_space [Accessed 31 May 2018].

[14] En.wikipedia.org. (2018). *Norm (mathematics).* [online] Available at: https://en.wikipedia.org/wiki/Norm_(mathematics) [Accessed 26 Jan. 2018].

[15] En.wikipedia.org. (2018). Origin (mathematics). [online] Available at: https://en.wikipedia.org/wiki/Origin_(mathematics) [Accessed 31 May 2018].

[16] Dasgupta, S. and Gupta, A. (2002). *An elementary proof of a theorem of Johnson and Lindenstrauss.* Random Structures and Algorithms, [online] 22(1), pp.60-65. Available at: http://onlinelibrary.wiley.com/doi/10.1002/rsa.10073/epdf [Accessed 24 Jan. 2018].

[17] Johnson, W. and Lindenstrauss, J. (1984). *Extensions of Lipschitz mappings into a Hilbert space.* Conference on Modern Analysis and Probability, [online] pp.189-206. Available at: https://www.researchgate.net/publication/235008656_Extensions_of_Lipschitz_maps_into_a_Hilbert_space [Accessed 19 Jan. 2018].

[18] Bourgain, J., Dirksen, S. and Nelson, J. (2015). *Toward a unified theory of sparse dimensionality reduction in Euclidean space.* Geometric and Functional Analysis, [online] 25(4), pp.1009-1088. Available at: https://arxiv.org/pdf/1311.2542.pdf [Accessed 19 Jan. 2018].

[19] En.wikipedia.org. (2018). *Curse of dimensionality.* [online] Available at: https://en.wikipedia.org/wiki/Curse_of_dimensionality [Accessed 22 Jan. 2018].

[20] Dai, W. (2017). *Curse of Dimensionality from Topics in Large Dimensional Data Processing* [online] Available at: http://www.ee.imperial.ac.uk/wei.dai/Teaching/EE4.66.2017/00EE4.66_part_1.pdf [Accessed 31 May 2018].

[21] En.wikipedia.org. (2018). Volume of an n-ball. [online] Available at: https://en.wikipedia.org/wiki/Volume_of_an_n-ball [Accessed 1 Jun. 2018].

[22] En.wikipedia.org. (2018). Law of large numbers. [online] Available at: https://en.wikipedia.org/wiki/Law_of_large_numbers [Accessed 1 Jun. 2018].

[23] En.wikipedia.org. (2018). Marchenko–Pastur distribution. [online] Available at: https://en.wikipedia.org/wiki/Marchenko\%E2\%80\%93Pastur_distribution [Accessed 2 Jun. 2018].

[24] Petridis, S. (2018). Course 395: Machine Learning - Lectures. [ebook] pp.32-34. Available at: https://ibug.doc.ic.ac.uk/media/uploads/documents/ml-lecture3-2018.pdf [Accessed 2 Jun. 2018].

[25] Lavrenko, V. (2015). IAML8.3 Examples of overfitting and underfitting. [video] Available at: `https://www.youtube.com/watch?v=dBLZg-RqoLg` [Accessed 2 Jun. 2018].

[26] Gluon.mxnet.io. (2018). Overfitting and regularization — The Straight Dope 0.1 documentation. [online] Available at: `https://gluon.mxnet.io/chapter02_supervised-learning/regularization-scratch.html` [Accessed 2 Jun. 2018].

[27] Kune, R., Konugurthi, P., Agarwal, A., Chillarige, R. and Buyya, R. (2015). The anatomy of big data computing. Software: Practice and Experience, [online] 46(1), pp.79-105. Available at: `https://arxiv.org/ftp/arxiv/papers/1509/1509.01331.pdf` [Accessed 2 Jun. 2018].

[28] Achlioptas, D. (2001). *Database-friendly random projections.* Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '01. [Online] Available at: `https://dl.acm.org/citation.cfm?doid=375551.375608` [Accessed 23 Jan. 2018].

[29] Tsitsvero, M., Barbarossa, S. and Di Lorenzo, P. (2016). *Signals on Graphs: Uncertainty Principle and Sampling.* IEEE Transactions on Signal Processing, [Online] 64(18), pp.4845-4860. Available at: `https://arxiv.org/pdf/1507.08822.pdf` [Accessed 23 Jan. 2018].

[30] En.wikipedia.org. (2018). *Uncertainty principle.* [Online] Available at: `https://en.wikipedia.org/wiki/Uncertainty_principle` [Accessed 23 Jan. 2018].

[31] Ailon, N. and Liberty, E. (2008). *Fast Dimension Reduction Using Rademacher Series onDual BCH Codes.* Discrete & Computational Geometry, [Online] 42(4), pp.615-630. Available at: `https://www.math.ias.edu/csdm/files/06-07/nailon_fast_dimension_reduction_using_rademacher.pdf` [Accessed 20 Nov. 2017].

[32] Dai, W. (2017). *Error Correcting Codes: Fundamentals* [online] Available at: `http://www.ee.imperial.ac.uk/wei.dai/Teaching/EE4.07.2017/00EE4.07_Part_2.pdf` [Accessed 31 May 2018].

[33] Peterson, W. and Weldon, E. (1972). *Errorcorrecting codes.* 2.ed. Cambridge,Mass.: MIT press, pp.206-308.

[34] Aspnes, J. (2003). KwiseIndependence. [online] Cs.yale.edu. Available at: `http://www.cs.yale.edu/homes/aspnes/pinewiki/KwiseIndependence.html` [Accessed 4 Jun. 2018].

[35] Dai, W. (2017). *Greedy Algorithms from Topics in Large Dimensional Data Processing* [online] Available at: `http://www.ee.imperial.ac.uk/wei.dai/Teaching/EE4.66.2017/00EE4.66_part_2.pdf` [Accessed 31 May 2018].

[36] Rudelson, M. and Vershynin, R. (2006). *Sparse reconstruction by convex relaxation: Fourier and Gaussian measurements.* 2006 40th Annual Conference on Information Sciences and Systems. [online] Available at: `http://ieeexplore.ieee.org.iclibezp1.cc.ic.ac.uk/document/4067804/?reload=true` [Accessed 28 Jan. 2018].

[37] Ailon, N. and Liberty, E. (2013). *An Almost Optimal Unrestricted Fast Johnson-Lindenstrauss Transform.* ACM Transactions on Algorithms, [online] 9(3), pp.1-12. Available at: `https://www.researchgate.net/publication/45920633_An_Almost_Optimal_Unrestricted_Fast_Johnson-Lindenstrauss_Transform` [Accessed 28 Jan. 2018].

[38] Kane, D. and Nelson, J. (2014). *Sparser Johnson-Lindenstrauss Transforms.* Journal of the ACM, [online] 61(1), pp.1-23. Available at: `https://arxiv.org/pdf/1012.1577.pdf` [Accessed 19 Jan. 2018].

[39] Nelson, J. and NguyÅn, H. (2013). *Sparsity lower bounds for dimensionality reducing maps.* Proceedings of the 45th annual ACM symposium on Symposium on theory of computing - STOC '13. [online] Available at: `https://arxiv.org/pdf/1211.0995.pdf` [Accessed 19 Jan. 2018].

[40] Bourgain, J., Dirksen, S. and Nelson, J. (2015). *Toward a unified theory of sparse dimensionality reduction in Euclidean space.* Geometric and Functional Analysis, [online] 25(4), pp.1009-1088. Available at: `https://arxiv.org/pdf/1311.2542.pdf` [Accessed 19 Jan. 2018].

[41] Uk.mathworks.com. (2018). Normal random numbers - MATLAB normrnd- MathWorks United Kingdom. [online] Available at: `https://uk.mathworks.com/help/stats/normrnd.html` [Accessed 12 Jun. 2018].

[42] Uk.mathworks.com. (2018). Randomly sample from data, with or without replacement - MATLAB datasample- MathWorks United Kingdom. [online] Available at: `https://uk.mathworks.com/help/stats/datasample.html` [Accessed 12 Jun. 2018].

[43] Uk.mathworks.com. (2018). Hadamard matrix - MATLAB hadamard- MathWorks United Kingdom. [online] Available at: `https://uk.mathworks.com/help/matlab/ref/hadamard.html` [Accessed 12 Jun. 2018].

[44] Uk.mathworks.com. (2018). Sample Data Sets- MATLAB & Simulink- MathWorks United Kingdom. [online] Available at: `https://uk.mathworks.com/help/stats/sample-data-sets.html` [Accessed 12 Jun. 2018].

[45] Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. [online] Available at: `http://archive.ics.uci.edu/ml` [Accessed 12 Jun. 2018]

[46] Pantic, M. (2018). N-fold Cross validation. [online] Ibug.doc.ic.ac.uk. Available at: `https://ibug.doc.ic.ac.uk/media/uploads/documents/ml-lecture2-2018_part2.pdf` [Accessed 16 Jun. 2018].

Appendices

.1 FJLT Optimal Results

		FJLT 1		**FJLT 2**		
n	**d**	**k**	**min e**	**k**	**beta**	**min e**
50	32	3	0.40	5	16	0.40
	64	4	0.44	6	32	0.38
	128	4	0.40	7	32	0.35
	256	8	0.32	9	64	0.25
	512	16	0.30	16	256	0.25
100	32	2	0.50	5	16	0.38
	64	3	0.40	7	32	0.37
	128	4	0.40	6	32	0.34
	256	8	0.32	10	128	0.34
	512	8	0.36	15	256	0.28
500	32	2	0.52	5	16	0.42
	64	2	0.50	7	32	0.40
	128	3	0.44	7	32	0.37
	256	4	0.40	9	64	0.35
	512	8	0.30	14	256	0.32
1000	32	2	4.80	5	16	0.46
	64	2	0.54	7	32	0.41
	128	4	0.38	7	32	0.38
	256	4	0.40	9	64	0.35
	512	8	0.38	18	256	0.33
5000	32	2	0.58	5	16	0.45
	64	2	0.46	7	32	0.42
	128	3	0.50	7	32	0.39
	256	3	0.38	9	64	0.35
	512	4	0.36	14	256	0.33
10000	32	2	0.50	5	16	0.45
	64	2	0.60	7	32	0.41
	128	2	0.52	7	32	0.39
	256	3	0.38	9	64	0.34
	512	4	0.36	18	256	0.32
	1024	8	0.34	25	512	0.30

Table 1: Optimal reduced dimension and error for different parameters

n: Data sample size
d: Original dimension size
k: Reduced dimension size
$\min e$: Minimum error at which the projection satisfies JL properties

.2 FJLT 2 High Dimensional Figures

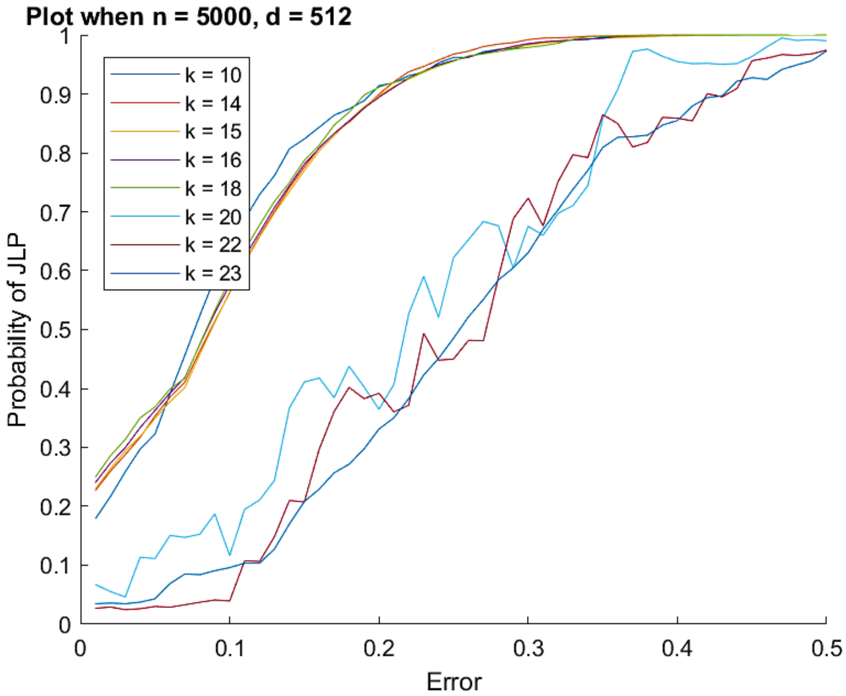


Figure 1: Probability of success when sample size = 5000 and d = 512

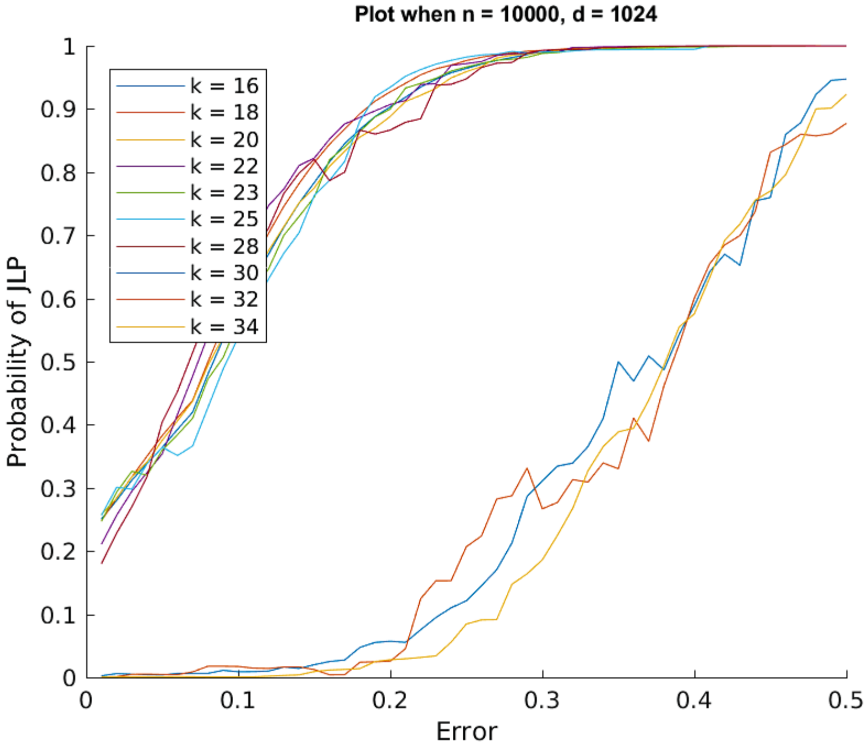


Figure 2: Probability of success when sample size = 10000 and d = 1024

.3 K-Fold Cross Validation Process

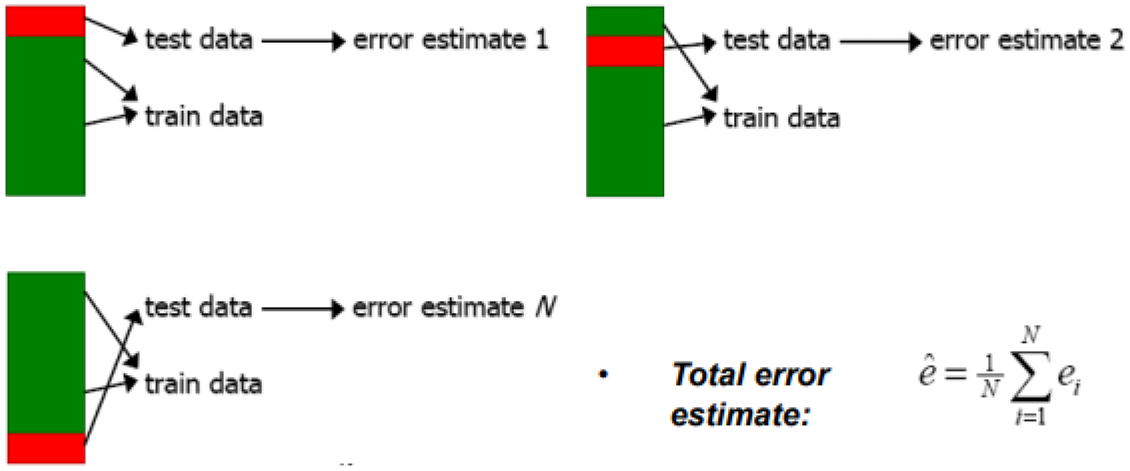


Figure 3: Separation of the data into N folds[46]

.4 Source Codes