Dimensionality Reduction for the Purposes of Automatic Pattern Classification

Stephen Strong, BEng (Hons)

School of Microelectronic Engineering
Faculty of Engineering and Information Technology
Griffith University, Brisbane, Australia

July 25, 2013

Dissertation submitted in fulfillment of the requirements of the degree of Masters of Philosophy
Abstract

Pattern classification is a common technique used in a variety of applications. From simple tasks, such as password acceptance, to more complex tasks, such as identification by biometrics, speech recognition, and text recognition. As a result, a large number of pattern classification algorithms have emerged, allowing computers to perform these tasks. However, these techniques become less effective when excessive data on a given object is provided in comparison to the number of samples required to train. As a result, much research has been placed in finding efficient methods of reducing the dimensionality of the data while maintaining maximum classification accuracy.

Dimensionality reduction aims to maximize the spread between samples of different classes, and minimize the spread between samples of the same class. A variety of methods aiming to do this have been reported in the literature. The most common methods of dimensionality reduction are Linear Discriminant Analysis and its variants. These typically focus on the spread of all the data, without regard to how spread out sections of the data already are. Few methods disregard the spread of data that is already spread out, but these are not so commonly accepted. While the classification accuracy is often better using these techniques, the computational time is often a large obstacle.

This thesis will investigate several methods of dimensionality reduction, and then discuss algorithms to improve upon the existing algorithms. These algorithms utilize techniques that can be implemented on any hardware, making them suitable for any form of hardware.
I am grateful to Professor Kuldip Paliwal for his support during my time with him. Additionally, I would like to extend my gratitude to the members of the Signal Processing Laboratory for their helpful suggestions and for creating an enjoyable work environment.

It is my pleasure to have worked with Professor Vladimir Estivill-Castro, and I thank him for his great assistance in writing this thesis. His help has improved the quality of this thesis notably.

My thanks is offered to my family and friends for their moral support during my time here. I would not have made it this far without them.

Finally, I express my appreciation to Griffith University for providing excellent facilities, laboratory space and equipment.
Statement of Originality

The work in this thesis has not been submitted for a degree or diploma in any other university. To the best of my knowledge, this thesis contains no information published by another author except where referenced appropriately.

Stephen Strong
Contents

Abstract v

Acknowledgements vii

Statement of Originality ix

1 Introduction 1
  1.1 Problem Overview ........................................................................ 1
  1.2 Organisation of the Thesis .............................................................. 2
  1.3 Publications Resulting from Research ........................................... 3

2 Literature Review 5
  2.1 Pattern Classification ................................................................. 5
    2.1.1 Introduction ........................................................................... 5
    2.1.2 Nearest Neighbour ............................................................... 6
    2.1.3 Nearest Centroid ................................................................. 7
    2.1.4 Support Vector Machines ...................................................... 8
    2.1.5 Soft Margin Support Vector Machines .................................... 11
    2.1.6 Kernels .............................................................................. 12
    2.1.7 Kernel SVM ......................................................................... 14
  2.2 Dimensionality Reduction ............................................................ 15
    2.2.1 Introduction ......................................................................... 15
    2.2.2 Principal Component Analysis .............................................. 16
4.4 Experimental Results . . . . . . . . . . . . . . . . . . . . . . . 52
4.5 Application to Regularized LDA . . . . . . . . . . . . . . . . 54
4.6 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57

5 Fast Implementation of Null Linear Discriminant Analysis using Simple Matrix Operations 59
5.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
5.2 Necessary and Sufficient Conditions for NLDA . . . . . . . . 61
5.3 Derivation of the New Algorithm . . . . . . . . . . . . . . . . 64
5.4 Experimental Results . . . . . . . . . . . . . . . . . . . . . . . 66
5.5 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68

6 Conclusion 69
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Accuracy of a variety of methods under normal conditions</td>
<td>43</td>
</tr>
<tr>
<td>3.2</td>
<td>Accuracy of a variety of methods with half the number of classes</td>
<td>43</td>
</tr>
<tr>
<td>3.3</td>
<td>Accuracy of a variety of methods with 2 training samples per class</td>
<td>44</td>
</tr>
<tr>
<td>3.4</td>
<td>Accuracy of a variety of methods with an increased number of training samples per class</td>
<td>45</td>
</tr>
<tr>
<td>3.5</td>
<td>Accuracy of a variety of methods with noisy training data</td>
<td>45</td>
</tr>
<tr>
<td>3.6</td>
<td>Accuracy of a variety of methods with noisy testing data</td>
<td>46</td>
</tr>
<tr>
<td>4.1</td>
<td>MMC LDA Algorithm 1</td>
<td>49</td>
</tr>
<tr>
<td>4.2</td>
<td>MMC LDA Algorithm 1 with PCA trick</td>
<td>49</td>
</tr>
<tr>
<td>4.3</td>
<td>MMC LDA Algorithm 2</td>
<td>52</td>
</tr>
<tr>
<td>4.4</td>
<td>Accuracy of MMC LDA with varying algorithms</td>
<td>53</td>
</tr>
<tr>
<td>4.5</td>
<td>Computational time of MMC LDA with varying algorithms</td>
<td>53</td>
</tr>
<tr>
<td>4.6</td>
<td>RLDA Algorithm 1</td>
<td>55</td>
</tr>
<tr>
<td>4.7</td>
<td>RLDA Algorithm 2</td>
<td>56</td>
</tr>
<tr>
<td>4.8</td>
<td>Accuracy of RLDA with varying algorithms</td>
<td>56</td>
</tr>
<tr>
<td>4.9</td>
<td>Computational time of RLDA with varying algorithms</td>
<td>57</td>
</tr>
<tr>
<td>5.1</td>
<td>NLDA Algorithm 1</td>
<td>65</td>
</tr>
<tr>
<td>5.2</td>
<td>Error rate of NLDA with varying algorithms</td>
<td>66</td>
</tr>
<tr>
<td>5.3</td>
<td>Computational time of NLDA with varying algorithms</td>
<td>67</td>
</tr>
</tbody>
</table>
5.4 Accuracy of QR-NLDA with respect to the necessary and sufficient conditions presented in Chapter 5. . . . . . . . . . . . 67

5.5 Accuracy of NLDA Algorithm 1 with respect to the necessary and sufficient conditions presented in Chapter 5. . . . . . . 67
Dimensionality Reduction
for the Purposes of
Automatic Pattern Classification
Chapter 1

Introduction

1.1 Problem Overview

Recognition of objects in our environment is a common task for people. By merely reading this, the reader recognises each letter in each word, then combines those letters into words and recognises their meaning, and then combines that to generate a greater meaning by combining them into sentences and paragraphs. Similarly, people can identify other people by merely looking at them and classifying them. In each case, people consistently perform these tasks with close to 100% accuracy. This capability of humans is taken for granted. However, current technology is unable to compete with this capability. Even young children can identify letters and words, yet a computer will still get errors when attempting to do the same. The simplest reason for this is that humans do not know how this process is done beyond instinct. As a result, researchers have attempted to show ways for which computers can classify various patterns. This area is called automatic pattern classification. Common recognition tasks include face recognition, speaker recognition, speech recognition, spam recognition and computer virus detection.

Current technology has not attained a satisfactory level of accuracy for some of these tasks to be applied in certain areas. For example, if face recognition were to be used for security alone, and the program was inaccurate to the point where it allowed 1 in 20 people to enter when it
should deny them access, and reject 1 in 20 people when it should allow them access, it would cause a security problem. Therefore, the current accuracy of these algorithms is insufficient.

In pattern classification, we are given a set of sample data referred to as the training samples, and the class labels of the training samples. Additionally, we are given some samples belonging to unknown class labels, referred to as unseen samples. The aim of pattern classification is to determine the class labels of the unseen samples. To do this, a decision rule is developed based on the training samples, and the decision rule is applied to each test sample. The algorithm used to make the decision rule should be robust to outliers and should not be overly sensitive to errors in the data. It has been shown that when the number of training samples is too few in comparison to the number of features in each sample, the decision rules tend to be highly specific to the training data, and do not perform well on test data. To overcome this, feature extraction algorithms can be used to reduce the number of features in each sample and therefore make the number of training samples larger in comparison to the number of features. The two methods of feature extraction are feature selection and dimensionality reduction.

Feature selection is often only useful when there is some significance in determining which features are the most useful. Generally speaking, dimensionality reduction algorithms are more accurate than feature selection algorithms, although they also have a larger computation time. Therefore, the focus of this thesis will be on dimensionality reduction techniques. Furthermore, emphasis will be placed on linear dimensionality reduction techniques, where the smaller set of features are produced by linear combinations of the original features.

1.2 Organisation of the Thesis

Chapter 2 will contain an overview of the current literature. We will include an overview of several pattern classification techniques as an overview of the area, then proceed to give an overview of unsupervised linear dimensionality reduction techniques. Next, we will discuss a spectrum of supervised linear
Chapter 1

dimensionality reduction techniques. Finally, we will give a brief overview of some databases that can be used to test these dimensionality reduction techniques.

Chapter 3 will examine the accuracy of the methods for Small Sample Size algorithms in the current literature, under varying conditions.

Chapter 4 will introduce a method of performing supervised linear dimensionality reduction techniques faster. This method will be applied to two of the techniques introduced in Chapter 2, and a comparison will be shown between the original methods and the proposed methods.

Chapter 5 will examine the criteria used for developing algorithms to perform Nullspace Linear Discriminant Analysis (NLDA) efficiently. Several papers have been produced which have claimed to have shown theoretically that their algorithms produce the same results as NLDA, however, these papers use invalid techniques to do so. We will show that the criteria being used to test equivalence is flawed and introduce a set of necessary and sufficient criteria for testing equivalence.

Chapter 6 will extend the method introduced in Chapter 3 to NLDA, and use the criteria introduced in Chapter 4 to show that the new method is equivalent in accuracy while also being more computationally efficient.

1.3 Publications Resulting from Research

The work in this thesis has produced the following papers that are currently under review.


Chapter 2

Literature Review

2.1 Pattern Classification

2.1.1 Introduction

Pattern classification is a popular area of research, due to the variety of situations in which it arises. It is an essential part to face recognition, speech recognition, protein classification, and many other tasks. Pattern classification can be divided into supervised learning, unsupervised training, and semi-supervised training [6]. Supervised learning takes a set of samples with known classes and attempts to find a function $f(\cdot)$ such that $f(x)$ is the class of $x$ [6]. Unsupervised learning takes a collection of samples without knowing the classes and tries to associate classes to each sample. Semi-supervised training takes a collection of mostly unlabelled data with some labelled samples, and attempts to find the function $f(\cdot)$ such that $f(x)$ is the class of $x$.

In each pattern classification problem, we assume we are given a set of $D$-dimensional vectors, $\{x_i|1 \leq i \leq N\}$, such that each vector belongs to one of $C$ classes, $\{C_j|1 \leq j \leq C\}$. If $x_i$ belongs to class $C_j$, we state $x_i \in C_j$. Given a new sample $\hat{x}$, we wish to find the class that $\hat{x}$ belongs to. The vectors with known classes are referred to as training samples, while the vector with an unknown class label is referred to as the unseen sample [6]. To determine the accuracy of an algorithm, databases are used.
Each database contains a set of training samples, a set of test samples, and the expected label of each test sample. The classification algorithm is run using the test samples as unseen samples and the resulting labels are compared to the expected labels.

### 2.1.2 Nearest Neighbour

In the nearest neighbour system, the test sample is compared to all training samples, and the class of the nearest training sample is considered the class of the test sample.

This method is sensitive to outliers. If a test sample is close to an outlier, the method will detect the outlier instead of a sample of the appropriate class. To compensate for this, the \( k \)-nearest neighbour algorithm is used in its place, where the class of the test sample is the most common class of the \( k \) nearest training samples [6].

Training a nearest neighbour classifier is simple. Just make the data easily readable. To classify a sample, the distance to each training sample will need to be found. Doing this requires \( O(D) \) time for each sample, and therefore, calculating all the distances requires \( O(ND) \) time. Finding the smallest distance takes \( O(N) \) time. Therefore, the overall classification is performed in \( O(ND) \) time.

It is possible to make the testing process more efficient at the cost of training time. The original features can be separated into subgroups, using a data structure named R-Trees [13]. These have no guarantee of improving the worst case time, but improve the average case significantly. Furthermore, the Delaunay Triangulation can be used to remove unnecessary training samples from the data [23]. This directly reduces the number of training samples being used, however, once again, there is no guarantee that this will reduce the worst time, since there may not be any unnecessary training samples. The average case time is reduced notably.

The main disadvantage of this method is that testing a sample is computationally expensive in comparison to other methods for testing a sample, even with the more efficient methods such as R-Trees and the Delaunay triangulation [6]. However, it is one of the most accurate
classification methods [6].

Also, the nearest neighbour method is highly affected by the scale of the data [6]. If the data is modified such that one feature is multiplied by some constant for all samples, the nearest neighbour method may classify it differently. The easiest method to overcome this is to standardize the size of the data, generally by dividing by the standard deviation. Often, however, this is not considered an issue, and the data is left untransformed.

2.1.3 Nearest Centroid

Since the biggest problem with the nearest neighbour method is the computational time required, we could try reducing the data to one point that best resembles the data. This would mean that, instead of having to search through all the data for each test sample, we would only need to search through a number of samples equal to the number of classes [6]. The sample used is the mean of the class.

This method ignores the structure of the data [6]. It is possible for the structure of the data to make the means a less useful representation of the data. For example, if a class consists of two clusters, the class mean will not lie close to any of the other samples, but instead, between the two clusters. The nearest centroid method works best for Gaussian classes with covariance matrices being a scalar multiple of the identity matrices.

Training a nearest centroid classifier involves calculating the mean of each class, which can be done in $O(ND)$ time. Testing a sample is similar to testing the nearest neighbour method, but there are only $C$ samples to compare to. As a result, testing a sample takes $O(CD)$ time. Methods such as R-Trees and the Delaunay Triangulation are no longer necessary for the nearest centroid algorithm.

This method is competitive with other common methods in terms of speed, however, the method makes several assumptions about the data, such as the mean being a valid representative of the data, and that the covariance matrices are multiples of the identity matrix. The Nearest Centroid classifier will perform poorly on data formed by rings where each class mean is roughly equal, for example.
2.1.4 Support Vector Machines

Support Vector Machines (SVMs) is a classifier that only works with two classes [6]. When working with SVMs, we introduce a set of values, \(\{y_i|1 \leq i \leq N\}\), such that

\[
y_i = \begin{cases} 
-1 & \text{if } x_i \in C_1, \text{ and} \\
1 & \text{if } x_i \in C_2.
\end{cases}
\]

SVM aims to maximize the distance between the boundaries of each class [6]. This can be expressed mathematically by the equation

\[
\text{maximize } J \text{ such that} \\
w^T x_i + b \geq J \text{ if } y_i = 1, \\
w^T x_i + b \leq -J \text{ if } y_i = -1, \text{ and} \\
w^T w = 1.
\]

Unfortunately, algorithms to solve this equation are inefficient and therefore impractical for use [6, 26]. To overcome this, we set \(J\) to 1, and minimize \(w^T w\). This is multiplied by \(\frac{1}{2}\) to reduce complexity at later stages of the derivation.

\[
\text{Minimize } \frac{1}{2} w^T w \text{ such that} \\
w^T x_i + b \geq 1 \text{ if } y_i = 1, \text{ and} \\
w^T x_i + b \leq -1 \text{ if } y_i = -1.
\]

Finally, since \(y_i\) can only be 1 or -1, we can simply this to

\[
\text{minimize } \frac{1}{2} w^T w \text{ such that} \\
1 - y_i(w^T x_i + b) \geq 0.
\]

Using the Karush-Kuhn-Tucker (KKT) conditions [6], this has a new
formulation.

Maximize $L$ with respect to $w, b$ and minimize with respect to $\alpha_i$ such that

$$L = \frac{1}{2}w^T w + \sum_{i=1}^{N} (\alpha_i - \alpha_i y_i w^T x_i - \alpha_i y_i b),$$

and

$$\alpha_i \geq 0.$$

Taking the partial derivatives with respect to $w$ and $b$, the following equations are derived.

$$\frac{\delta L}{\delta w} = w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0,$$

and

$$\frac{\delta L}{\delta b} = - \sum_{i=1}^{N} \alpha_i y_i = 0.$$

Equating the partial derivatives to 0 and simplifying gives us the following equations.

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i,$$

and

$$\sum_{i=1}^{N} \alpha_i y_i = 0.$$

Substituting back into $L$:

$$L = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i y_i x_i^T x_j y_j \alpha_j + \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i y_i x_i^T x_j y_j \alpha_j$$

$$= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i y_i x_i^T x_j y_j \alpha_j.$$
Therefore, training a SVM is performed by minimizing \( L \) where [6]

\[
L = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i y_i x_i^T x_j y_j \alpha_j,
\]

\( \alpha_i \geq 0 \forall i, \) and

\[
\sum_{i=1}^{N} \alpha_i y_i = 0
\]

and finding [6]

\[
\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i, \text{ and } \quad b = 1 - \max_{y_i=1} \mathbf{w}^T \mathbf{x}_i.
\]

Note that the values for \( \alpha_i \) have to be larger than or equal to 0. In practice, the only case where \( \alpha_i \neq 0 \) is when \( 1 - y_i (\mathbf{w}^T \mathbf{x}_i + b) = 0 \). The \( \mathbf{x}_i \) that cause this to happen are called “support vectors” [6]. The optimal methods to train a SVM are the Interior Point method and Sequential Minimization Optimization. The details of these algorithms are beyond the scope of this thesis.

To predict the class of the test sample, \( \mathbf{x} \) using SVM, the following is used [6]

\[
\hat{y}_i = \begin{cases} 
-1 & \text{if } \mathbf{w}^T \mathbf{x} + b < 0, \text{ and} \\
+1 & \text{otherwise.}
\end{cases}
\]

The time to classify a single sample is \( O(D) \), comparable to the nearest centroid method (considering that SVM can only work with two classes).

The drawbacks of this method are that it cannot cope with data that cannot be linearly sepeated [6]. In addition to that, outliers can influence the response significantly. It also cannot cope with more than two classes. This basic form of SVM is purely theoretic due to the limitation on linear seperability between the classes. Generally speaking, when SVM is used, a soft margin is assumed.
2.1.5 Soft Margin Support Vector Machines

To overcome the requirement of linear separability, a soft margin is used [6]. This replaces the strict requirement of being on the correct side of the margin with a penalization for not being on the correct side.

The soft margin method allows for some error in the margin, but penalizes it [6]. As a result, soft margin SVM wishes to minimize

\[
\frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i,
\]

such that \(1 - y_i (w^T x_i + b) - \xi_i \geq 0\)

where \(C\) is the cost parameter, which is decided upon prior to training.

Using the KKT conditions [6] and simplifying, this problem can be rewritten as

Maximize \(L = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i y_i x_i^T x_j y_j \alpha_j\),

such that \(0 \leq \alpha_i \leq c, \forall i, \) and

\[\sum_{i=1}^{N} \alpha_i y_i = 0, \forall i\]

and finding

\[w = \sum_{i=1}^{N} \alpha_i y_i x_i \text{ and}\]

\[b = 1 - \min_{y_i=1, \alpha_i \neq 0} w^T x_i.\]

The only difference in the KKT conditions is that \(\alpha_i\) has an upper limit equal to the cost parameter. If the cost parameter is too high, then the solution will be identical to the regular SVM. If the cost parameter is too low, then the solution will allow for large errors, and the system will be flawed due to this [6].

The disadvantage of this method in comparison to SVM without the soft margin is the requirement to find a suitable cost parameter. Unfortunately,
there is little analysis of finding an optimal cost parameter [6].

2.1.6 Kernels

Given a system which is entirely depend on the inner products of vectors, e.g. $x_i^T x_j$, it is possible to redefine the inner product so that the system works in a higher dimensionality without ever actually working with higher dimensionality [6]. Furthermore, in some cases, we can utilize a kernel matrix to decrease the computation time required to train a classification system, although, generally speaking, it will not be faster than non-kernal based methods for testing.

The definition of the three most common kernels are [6]

- Linear kernel $k(x_i, x_j) = x_i^T x_j$,
- Polynomial kernel $k(x_i, x_j) = (x_i^T x_j + 1)^c$, and
- Gaussian kernel $k(x_i, x_j) = \exp\left(\frac{|x_i - x_j|^2}{2\sigma^2}\right)$.

Before an example is given, some terms need to be defined. As already indicated, kernelization treats the redefined inner product as the dot product of two vectors of higher dimension. These vectors are referred to as $\phi(x_i)$ [6]. This allows us to express the dot product as $\phi(x_i)^T \phi(x_j)$ [6]. This can also be written as $k(x_i, x_j)$. The kernel matrix, $K$, is a $n \times n$ matrix, where $n$ is the number of data samples, such that [6]

$$K_{ij} = k(x_i, x_j).$$

Also, it is not uncommon for a matrix of ones to be required. The symbol $1_{M,N}$ denotes a $M \times N$ matrix where every element is equal to 1. The matrix $1_N$ is equivalent to $1_{N,N}$.

To demonstrate the application of this, suppose we have a set of vectors, $\{x_i|1 \leq i \leq N\}$ and we have a vector $w$. We wish to find the mean and variance of the values formed by $\phi(w)^T \phi(x_i)$. For most kernelization processes, vectors not included in $\{x_i|1 \leq i \leq N\}$ are recognized as the linear combination of those vectors, kernelized. In this case, $w$ is the only
such vector, so a set of values \( \{ \alpha_i \mid 1 \leq i \leq N \} \) are added such that

\[
w = \sum_{i=1}^{N} \alpha_i x_i.
\]

The vector \( \alpha \) is a \( N \times 1 \) vector, defined such that \( \alpha(i) = \alpha_i \).

The mean is calculated as

\[
\mu = \frac{1}{N} \sum_{i=1}^{N} \phi(w)^T \phi(x_i)
= \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_j \phi(x_j)^T \phi(x_i)
= \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_j K(i, j)
= \frac{1}{N} 1_{1,N} K \alpha.
\]

For the variance, first, \( E[(\phi(w)^T \phi(x_i))^2] \) is calculated.

\[
E[(\phi(w)^T \phi(x_i))^2] = \frac{1}{N} \sum_{i=1}^{N} (\phi(w)^T \phi(x_i))^2
= \frac{1}{N} \sum_{i=1}^{N} \phi(w)^T \phi(x_i) \phi(x_i)^T \phi(w)
= \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{l=1}^{N} \alpha_j \phi(x_j)^T \phi(x_i) \phi(x_i)^T \phi(x_l) \alpha_l
= \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{l=1}^{N} \alpha_j K(j, i) k(i, l) \alpha_l
= \frac{1}{N} \alpha K^2 \alpha.
\]

Then, the variance is simply \( \sigma^2 = E[(\phi(w)^T \phi(x_i))^2] - \mu^2 \).

\[
\sigma^2 = \frac{1}{N} \alpha K^2 \alpha - \left( \frac{1}{N} 1_{1,N} K \alpha \right)^2
= \frac{1}{N} \alpha K^2 \alpha - \frac{1}{N^2} \alpha^T K 1_{1,N} 1_{1,N} K \alpha.
\]
It can be shown that $1_{N,1}1_{1,N} = 1_N$. Therefore:

$$\sigma^2 = \frac{1}{N} \alpha K^2 \alpha - \frac{1}{N^2} \alpha^T K_1 K \alpha.$$ 

### 2.1.7 Kernel SVM

SVM has the special property that all of the calculations involve inner products. This allows us to apply the kernelization process [6]. For this, the $b$ term is removed, since a constant term to replace it regularly appears in kernel functions. Modifying the SVM equations to include kernel functions results in maximizing $L$ where

$$L = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i K_{ij} \alpha_j,$$

such that $0 \leq \alpha_i \leq c \forall i$.

where

$$w^T x = \sum_{j=1}^{N} \alpha_j y_j k(x_i, x)$$

with a modification to $K$, such that

$$K_{i,j} = y_i y_j k(x_i, x_j).$$

This can be rewritten as maximizing $L$ where

$$L = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \alpha_i K \alpha_i,$$

such that $0 \leq \alpha_i \leq c \forall i.$

(2.1)

To predict a sample using kernel SVM, the following rule is used.

$$y_i = \begin{cases} -1 & \text{if } \sum_{j=1}^{N} \alpha_j y_j k(x_i, x) < 0, \text{ and} \\ +1 & \text{if otherwise.} \end{cases}$$
Since only the support vectors have non-zero terms, the calculation of the class label of the unseens sample requires the summation of at most $n_{SV}$ terms, where $n_{SV}$ is the number of support vectors. Assuming the kernel function takes $O(D)$ time to apply, this means that classifying a single sample takes $O(n_{SV} D)$ [6].

This uses the same methods of computation as the soft margin technique. In addition to the selection of the cost parameter, kernel SVM requires the selection of a kernel and any parameters that the selected kernel requires. Additionally, classifying a sample can be significantly more expensive, depending on the number of support vectors.

### 2.2 Dimensionality Reduction

#### 2.2.1 Introduction

When presented with features with a large number of dimensions, several problems occur [6]. A simple problem is the computation complexity of working with the features. For example, when working with the Nearest Neighbour method, testing a single point will take $O(DN)$ time. Using the ORL database as an example, we have 400 samples with 10304 dimensions. The time taken will be of the order $4 \times 10^6$ operations. If a linear dimensionality reduction technique is used to cut down from $D$ to $d$ dimensions, the time taken would be of the order $O(dD + dN)$. Assuming we reduce to 40 dimensions, the time taken will be of the order $4 \times 10^5$ operations. Furthermore, assuming that each feature of each sample took a byte of memory, the original data required 3.9 MB while the reduced samples requires 15.6 kB.

Furthermore, excessive features is likely to introduce noise into the system. It has been shown that the number of training samples should be at least 10 times the number of dimensions per feature to produce a well-trained classifier [6]. As a result, taking the ORL database again, we would need 103040 samples to create a well-trained classifier. Clearly, this is impractical: if each sample took one minute to generate, it would take over two months to produce that many samples. A more practical method
would be to reduce the number of dimensions. Given 400 samples, ideally, there should be at most 40 features.

Finally, it is difficult to imagine data in a set of dimensions greater than 3. As a result, dimensionality reduction can be used to aid in the visualization of data, by transforming a 10-dimensional data set to a mere 2 or 3 dimensions, and plotting the result.

With this in mind, it is necessary to find good methods of dimensionality reduction. Generally speaking, these methods are linear in nature, although kernels can be used to modify this. Linear Dimensionality Reduction is performed by finding some matrix $W$, of size $D \times d$ such that $W$ is optimal in some way. Then, to apply it, multiply the sample by $W$, and use the resulting vector as the sample.

In this section, Principal Component Analysis will be analysed, followed by the analysis of some variants of Principal Component Analysis. After that, Linear Discriminant Analysis will be analysed, followed by the analysis of some variants of Linear Discriminant Analysis.

### 2.2.2 Principal Component Analysis

One method to define the optimality of $W$ is to set $W$ such that, if we transform the original $D$-dimensional data to $d$-dimensional data, and then transform the $d$-dimensional data back to $D$-dimensional data, the error between the original data and the reconstructed data is minimal [6]. Before doing this, it is typical to zero-mean the data, by subtracting the mean of the data from each sample.

The aim of Principal Component Analysis (PCA) is to minimize the reconstruction error created by the transformation of multiplying by $W$, a matrix of size $D \times d$, where the reconstruction error is defined as [6]

$$e(W) = \frac{1}{N} \sum_{i=1}^{N} \| WW^T x_i - x_i \|^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} (x_i^T WW^T WW^T x_i - 2x_i^T WW^T x_i + x_i^T x_i).$$
Since \( W \) is orthonormal, \( W^TWW^T = W^T \). Substituting this into \( e(W) \), we find that

\[
e(W) = \frac{1}{N} \sum_{i=1}^{N} \left( x_i^T W W^T x_i - 2x_i^T WW^T x_i + x_i^T x_i \right) = \frac{1}{N} \sum_{i=1}^{N} (x_i^T x_i - x_i^T WW^T x_i).
\]

Since \( x_i \) is independent of \( W \), this is equivalent to maximizing

\[
J(W) = \frac{1}{N} \sum_{i=1}^{N} x_i^T WW^T x_i = \text{trace}\left( \frac{1}{N} \sum_{i=1}^{N} x_i^T W W^T x_i \right) = \text{trace}\left( \frac{1}{N} \sum_{i=1}^{N} W^T x_i x_i^T W \right) = \text{trace}\left( W^T \left( \frac{1}{N} \sum_{i=1}^{N} x_i x_i^T \right) W \right) = \text{trace}(W^T S_t W)
\]

where \( S_t = \frac{1}{N} \sum_{i=1}^{N} x_i x_i^T \).

This cannot be solved directly, so instead, it is solved as a series of vectors. Therefore, we maximize

\[
w^T S_t w \text{ such that } w^T w = 1.
\]

This can be solved using Lagrange multipliers.

\[
L = w^T S_t w - \lambda w^T w,
\]
Finding the partial derivate with respect to $\mathbf{w}$ and equating to 0, we find:

$$
\frac{\delta L}{\delta \mathbf{w}} = 2\mathbf{S}^T\mathbf{w} - 2\lambda \mathbf{w},
\quad 0 = \mathbf{S}^T\mathbf{w} - \lambda \mathbf{w}
$$

Therefore, the vectors that minimize the reconstruction error will be the eigenvectors of $\mathbf{S}^T$. To determine which eigenvalues to use, we note that we wish to maximize $\mathbf{w}^T\mathbf{S}^T\mathbf{w}$ under the constraint that $\mathbf{w}^T\mathbf{w} = 1$ and premultiply the eigenvector equation by $\mathbf{w}^T$.

$$
\mathbf{w}^T\mathbf{S}^T\mathbf{w} = \lambda \mathbf{w}^T\mathbf{w}
$$

$$
\mathbf{w}^T\mathbf{S}^T\mathbf{w} = \lambda
\quad (2.2)
$$

Therefore, the optimal vectors are the eigenvectors of $\mathbf{S}^T$ with the largest corresponding eigenvalues. The matrix $\mathbf{W}$ is formed of the $d$ eigenvectors of $\mathbf{S}^T$ with the largest corresponding eigenvalues.

Given the full data, the first step is to zero-mean the data, which can be done in $O(DN)$ time. The eigenvalues of the covariance matrix can be calculated through Singular Value Decomposition (SVD), which takes $O(DN^2)$ or $O(D^2N)$, whichever is faster. Therefore, for applications with more samples than features such as speech coding, the algorithm is of the order $O(D^2N)$, and for applications with more features than samples such as face recognition, the algorithm is of the order $O(DN^2)$. To transform a given sample, it is a matrix multiplication of a $d \times D$ matrix by a $D \times 1$ vector, which can be done in $O(Dd)$ time. As a result, training the system takes $O(min(DN^2, D^2N))$ time, and transforming the unseen sample can be done in $O(Dd)$ time.

While PCA is suitable for coding and visualization, it suffers drawbacks when we attempt to use it for classification. The information that best reconstructs the data may separate the data incorrectly [6]. As an example, in faces, the most descriptive eigenvectors can give information about lighting, facial expressions or other details that offer little helpful
information in determining whose face it is. These features help create the minimal error in reconstruction, but do not guarantee any use in classification.

In addition to that, if the data has stronger emphasis towards certain features, that will show in the features selected by PCA. As a method to help overcome this, it is helpful to divide each dimension by the standard deviation of the features in that dimension [6]. This takes $O(DN)$ time, so does not change the computational complexity.

Furthermore, if there are nonlinear dependencies between the data, then a linear method, such as PCA will do little to take advantage of it. In such a case, it is required to look for nonlinear dimensionality reduction techniques for use in this aspect [6].

2.2.3 Kernelized Principal Component Analysis

To overcome the issue of non-linear dependencies, it has been proposed to apply kernels to PCA [6]. To accomplish this, we assume that

$$k(w_j, x) = \sum_{i=1}^{N} \alpha_{i,j} k(x_i, x)$$

and that the $j^{th}$ row of $W$ is equivalent to $w_j$. The values for $\alpha_{i,j}$ are calculated as the eigenvectors of the matrix $K$, a modified kernel matrix where the samples have been zero-meaned in the kernel space [6]

$$\hat{K} = K - \mathbf{1}K - K\mathbf{1} + \mathbf{1}\mathbf{1},$$

where $\mathbf{1}$ is a $n \times n$ matrix where every element equals $1/N$.

To transform the vector $x$, the kernel function is used on $x$ and each training sample, and a weighted sum is taken. Specifically,

$$\hat{x}_j = \sum_{i=1}^{N} \alpha_{i,j} k(x_i, x),$$

where $\hat{x}$ is the transformation of $x$ [6].
Kernelized PCA has better performance than normal PCA, but with larger time and space requirements. To train the system, first, $K$ needs to be calculated. Assuming that $k(x_i, x_j)$ can be found in $O(D)$ time, this will take $O(DN^2)$ time. After that, a series of matrix multiplications are performed on $N \times N$ matrices to set the mean to zero, which can be done in $O(N^3)$ time. Finally, eigenvector analysis is performed on an $N \times N$ matrix, which can be done in $O(N^3)$ time. Therefore, the algorithm takes time $O(DN^2 + N^3)$. In comparison, PCA can be trained in $O(\min(DN^2, D^2N))$ time. In addition to that, to transform a given sample, first, it is necessary to find each $k(x_i, x)$, which takes $O(DN)$ time. Then, multiplying by the appropriate $\alpha_{i,j}$ and adding them up takes $O(NDd)$ time. Therefore, the kernel-PCA transformation takes $O(NDd)$ time, compared to the PCA transformation, which takes $O(dD)$ time.

This leads to the first drawback of kernel-PCA, that is, the difference in speeds. The main drawback is in the calculation of a reduced vector. Given a database with 20,000 samples and 16 features, reducing it to 8 features takes time proportional to $2^{3.5} \times 10^6$ operations with kernel-PCA, as opposed to the 128 operations performed by PCA.

Another drawback with kernelized PCA lies in the calculation of parameters for the kernels, and the selection of the kernel itself. Polynomial kernels, as an example, use the formula $k(x, x') = (\langle x, x' \rangle + 1)^c$, where $c$ can vary. These are selected heuristically, and minimal research has been done into determining an optimal value. Furthermore, choosing the correct kernel is also a process with minimal research, once again, it is usually heuristically selected. While the differences between kernels is small for SVM techniques, PCA is more sensitive, since PCA is affected by the scale of the data.

### 2.2.4 Two-dimensional Principal Component Analysis

Another modification made to PCA is to incorporate dependencies in-built into the features. To apply PCA to an image, first, we transform the image matrix into a vector, by concatenating each row. An image that was $100 \times 100$ would then become a $10000 \times 1$ dimensional vector. Yang and Zhang proposed a method called 2-Dimensional PCA (2DPCA) [29], where
the image is left as a matrix, and two transformational matrices are found, resulting in \( L^TXR \) as the transformed \( X \). To accomplish this, we wish to find \( L \) and \( R \) such that:

\[
\sum_{i=1}^{N} \text{trace}(L^TX_iRR^TX_i^TL)
\]

is maximized.

To obtain this, the algorithm first finds \( L \) as the eigenvectors of \( S_R \), defined as [29]:

\[
S_R = \sum_{i=1}^{N} X_iX_i^T.
\]

Then, \( R \) is found as the eigenvectors of \( S_L \), defined as [29]:

\[
S_L = \sum_{i=1}^{N} X_i^TLL^TX_i.
\]

In each case, we select the eigenvectors with the largest corresponding eigenvalues.

If 2DPCA is used to reduce a 100 \( \times \) 100 matrix to an 8 \( \times \) 8 matrix, there are still 64 features. As a result, it is common to apply PCA to the reduced matrix, resulting in 2DPCA+PCA [29]. For the purposes of this thesis, this will be shortened to 2DPCA, and the original 2DPCA will be considered obsolete.

To perform 2DPCA on a set of \( N \) \( (m \times n) \) matrices, where \( N > m \leq n \) labelled \( \{X_1, X_2, ..., X_N\} \), and reduce such matrices to size \( q \times q \), and reduce that to a \( d \)-dimensional vector, first, \( \sum_{i=1}^{N} X_i \) is calculated, which can be done in \( O(Nmn) \) time. SVD is applied so that the eigenvectors of \( S_L \) can be found, which can be performed in \( O(mn^2) \) time. This gives the matrix \( L \), which is size \( m \times q \). To find \( R \), \( \sum_{i=1}^{N} X_i \) is premultiplied by \( L^T \), which can be done in time \( O(mnq) \), and SVD is applied so that the eigenvectors of \( S_R \) can be found, which can be done in \( O(nq^2) \). This gives the matrix \( R \), which is size \( n \times q \). Finally, the original vectors are transformed, which can be done in \( O(Nmnq) \), and then PCA is used on the transformed features, which can be done in time \( O(Nq^2) \). The longest step is \( O(Nmnq) \), so training can be done in \( O(Nmnq) \) time.
Transforming a given sample is done by finding $L^TXR$, which can be done in $O(mnq)$ time, then we transform it to a $d$-dimensional vector, which can be done in $O(dq^2)$ time. Since $dq \ll mn$, transforming a matrix is done in $O(mnq)$ time.

2DPCA is limited to finding dependencies between elements in the same row or column, which limits its potential. In addition to that, the method is limited to matrices. Finally, 2DPCA requires more coefficients to get the same effect as PCA, although 2DPCA+PCA bypasses this.

### 2.2.5 Linear Discriminant Analysis

One of the drawbacks of PCA was that the features that were best for reconstruction were not necessarily the best for classification. As a result, Fisher proposed Linear Discriminant Analysis (LDA), a dimensionality reduction algorithm designed with classification in mind [6, 9]. It had two goals, to maximize the spread of the class means in the transformed space, and to minimize the spread of each individual class.

We begin by defining $\{m_i | 1 \leq i \leq C\}$ such that $m_i$ is the mean of all the samples in $C_i$, and $\{N_i | 1 \leq i \leq C\}$ such that $N_i$ is the number of elements in $C_i$. Also, it will be assumed that the data has been modified to have a mean of 0. Then, we can define $S_B$ and $S_W$ as follows [6, 9]:

$$
S_B = \frac{1}{N} \sum_{i=1}^{C} N_i m_i^T m_i, \quad \text{and}
$$

$$
S_W = \frac{1}{N} \sum_{i=1}^{C} \sum_{x_j \in C_i} (x_j - m_i)^T (x_j - m_i).
$$

It can be shown that $S_B + S_W = S_T$, which can help in our calculations later [6]. To maximize the spread of the class means, $\text{trace}(W^T S_B W)$ should be maximized. Similarly, to minimize the spread of the individual classes, $\text{trace}(W^T S_W W)$ should be minimized. With these in mind, Fisher
presented the Fisher criterion, defined as

$$J(W) = trace((W^T S_W W)^{-1}W^T S_B W).$$

To optimize this, $\hat{S}_B$ and $\hat{S}_W$ are defined as

$$\hat{S}_B = W^T S_B W,$$

and

$$\hat{S}_W = W^T S_W W.$$

Substituting these into the formula for $J(W)$ and finding the derivative, we find that

$$J(W) = trace(\hat{S}_W^{-1} \hat{S}_B)$$

$$\frac{\delta J(W)}{\delta W} = -2S_W W \hat{S}_W^{-1} \hat{S}_B \hat{S}_W^{-1} + 2S_B W \hat{S}_W^{-1}.$$

Equating the derivative to 0, it is found that

$$2S_W W \hat{S}_W^{-1} \hat{S}_B \hat{S}_W^{-1} = 2S_B W \hat{S}_W^{-1}$$

$$S_W W \hat{S}_W^{-1} \hat{S}_B = S_B W$$

$$W \hat{S}_W^{-1} \hat{S}_B = S_W^{-1} S_B W.$$  \hspace{1cm} (2.3)

It can be shown that there exists a square matrix $B$ such that

$$B^T \hat{S}_B B = \mu$$ where $\mu$ is diagonal,

$$B^T \hat{S}_W B = I,$$ and

$$\|B\| \neq 0.$$
Since \( \|B\| \neq 0 \), then \( B^{-1} \) exists.

\[
J(W) = trace(\hat{S}_{W}^{-1}\hat{S}_{B})
= trace(\hat{S}_{W}^{-1}\hat{S}_{B}BB^{-1})
= trace(B^{-1}\hat{S}_{W}^{-1}\hat{S}_{B}B)
= trace(B^{-1}\hat{S}_{W}^{-1}B^{-T}B^{T}\hat{S}_{B}B)
= trace((B^{T}\hat{S}_{W}B)^{-1}B^{T}\hat{S}_{B}B)
= trace(I^{-1}\mu)
= trace(\mu).
\]

Therefore, \( J(W) \) is the sum of the diagonal terms of \( \mu \). Additionally,

\[
J(W) = trace\{(B^{T}\hat{S}_{W}B)^{-1}B^{T}\hat{S}_{B}B\}
= trace\{(B^{T}\hat{S}_{W}B)^{-1}B^{T}W^{T}S_{B}WB\}
= J(WB).
\]

Therefore, \( WB \) will maximize \( J(W) \). Next, we note the following

\[
B^{T}\hat{S}_{W}B = I
\hat{S}_{W} = B^{-T}B^{-1},
\hat{S}_{W}^{-1} = BB^{T}.
\]

Similarly, for \( \hat{S}_{B} \)

\[
B^{T}\hat{S}_{B}B = \mu
\hat{S}_{B} = B^{-T}\mu B^{-1}.
\]

Therefore

\[
\hat{S}_{W}^{-1}\hat{S}_{B} = BB^{T}B^{-T}\mu B^{-1}
= B\mu B^{-1}.
\]
Noting the optimality condition from (2.3), we find

\[ S_W^{-1}S_B W = WB\mu B^{-1} \]
\[ S_W^{-1}S_B WB = WB\mu. \]

Therefore, \(WB\) are the eigenvectors of \(S_W^{-1}S_B\), and the eigenvalues are the diagonal values of \(\mu\) [9]. Therefore, \(J(\cdot)\) is optimized by a matrix consisting of the eigenvectors of \(S_W^{-1}S_B\) with the largest corresponding eigenvalues.

Since we require the inverse of \(S_W\), it is necessary that the matrix be non-singular [6, 9, 27, 33, 22, 3, 8, 15]. This will be the case as long as \(N > D\). As a result, LDA cannot cope with pattern classification tasks where the number of features in a given sample is larger than the number of samples. Several algorithms have been found to overcome this issue.

To perform Linear Discriminant Analysis (LDA), we need \(N > D\), so we do not need to discuss the alternative. To calculate the class means, \(O(ND)\) time is required. Calculating \(S_B\) and \(S_W\) requires \(O(CD^2)\) and \(O(ND^2)\) time, respectively. Finding the eigenvectors and eigenvalues of \(S_W^{-1}S_B\) will require \(O(D^3)\) time. As a result, the overall process takes \(O(ND^2)\) time. The time taken to transform a given vector into the lower dimensional space is \(O(Dd)\).

LDA has the major drawback that it cannot work with the so-called Small Sample Size (SSS) problem [6, 9]. When \(N < D\), \(S_W\) is singular, and therefore the inverse cannot be found. In addition to that, there are a number of finite solutions equal to the rank of \(S_B\). Typically, this results in a maximum of \(c-1\) classes.

Another drawback is that LDA assumes the data is Gaussian in nature [21]. For non-Gaussian data, Linear Discriminant Analysis can draw incorrect assumptions about the data. Although this is an uncommon problem, it does occur, and there are algorithms that attempt to cope with this. Furthermore, Linear Discriminant Analysis merely attempts to separate the class means, and does not fully utilize the discriminative information found in the differences between the covariance matrices, instead assuming that all classes have the same covariance matrix [18].
2.2.6 Fisher Linear Discriminant Analysis

The focus of most research has been the prominent drawback of LDA already mentioned and know as the SSS problem. Swets and Weng suggested an application of the PCA algorithm to reduce the dimensionality to the largest that LDA could work with, then apply LDA \cite{27}. This method has been named Fisher Linear Discriminant Analysis or PCA+LDA.

Since the method is only applied when the small sample size problem is present, we can assume that \( D > N \). Therefore, the PCA step transforms the \( D \)-dimensional data to \((N - C)\)-dimensional data, which takes \( O(DN^2) \) time. The data then is transformed, which takes \( O(DNC) \) time. Once the transformed data is obtained, the LDA step transforms the \((N - C)\)-dimensional data to \( d \)-dimensional data in \( O(N^3) \) time. The two matrices are then multiplied together, which can be done in \( O(DNC) \) time. As a result, finding the transform can be done in \( O(DN^2) \) time. Transforming a vector is still \( O(dD) \) time. As a result, this method takes similar time to Principal Component Analysis.

The main drawback of PCA+LDA is that information regarding classification is easily lost in the process. The PCA step only keeps information relevant to reconstruction, and several of these features can be shown to be of little use. Also, it is slower than some other methods that attempt to find a discriminant dimensionality reduction for data suffering the small sample size problem.

2.2.7 Direct Linear Discriminant Analysis

Yu and Jang \cite{33} suggested a method which they claimed would overcome the small sample size problem while still optimizing the Fisher criterion. Unfortunately, this was proven invalid by Gao and Davis \cite{10}, who showed that the method was instead solving for \( W \) such that

\[
S_B^+S_WW = DW.
\]
The $\mathbf{D}$ was a full-rank diagonal matrix with minimal trace. However, the technique still solves the small sample size problem, and therefore still sees some use. This technique is called Direct Linear Discriminant Analysis (DLDA).

To efficiently perform DLDA, a technique called diagonalization can be used [33]. First, $\mathbf{S}_B$ is decomposed using Singular Value Decomposition (SVD). This gives

$$
\mathbf{S}_B = \mathbf{U}_B \mathbf{D}_B \mathbf{U}_B^T.
$$

Where $\mathbf{D}_B$ is a diagonal matrix consisting only of non-zero terms and $\mathbf{U}_B \mathbf{U}_B^T = \mathbf{I}$. Then, set

$$
\mathbf{Z} = \mathbf{U}_B \mathbf{D}_B^{-1/2};
$$

and decompose $\mathbf{Z}^T \mathbf{S}_B \mathbf{Z}$ using SVD, finding

$$
\mathbf{Z}^T \mathbf{S}_B \mathbf{Z} = \mathbf{U}_W \mathbf{D}_W \mathbf{U}_W^T.
$$

Where $\mathbf{U}_W \mathbf{U}_W^T = \mathbf{I}$ and $\mathbf{D}_W$ is a diagonal matrix consisting only of non-zero terms. Select the rows of $\mathbf{U}_W$ with the smallest corresponding rows of $\mathbf{D}_W$, and combine them to make $\mathbf{U}$. The transformation matrix is then

$$
\mathbf{W} = \mathbf{D}_W^{-1/2} \mathbf{U}^T \mathbf{Z}^T.
$$

To avoid the computational complexity involved in decomposing $\mathbf{S}_B$, we introduce a matrix, $\mathbf{H}_B$, such that [33]:

$$
\mathbf{H}_B = \begin{bmatrix}
\sqrt{N_1} (m_1 - \mathbf{m}) & \sqrt{N_2} (m_2 - \mathbf{m}) & \cdots & \sqrt{N_C} (m_C - \mathbf{m})
\end{bmatrix}.
$$

Decomposing $\mathbf{H}_B$ using SVD gives us $\mathbf{H}_B = \mathbf{U}_B \mathbf{D}_B^{1/2} \mathbf{V}_B$.

To calculate this, the decomposition of $\mathbf{H}_B$ requires $O(DC^2)$ time. The matrix $\mathbf{Z}$ can then be calculated in $O(DC^2)$ time. A matrix $\mathbf{H}_W$ is introduced, such that the $i$th row of the matrix is $\sqrt{n} \mathbf{x}_i - \mathbf{m}_j$, where $\mathbf{x}_i \in \mathcal{C}_j$. Then, $\mathbf{S}_W = \mathbf{H}_W^T \mathbf{H}_W$. The SVD of $\mathbf{Z}^T \mathbf{S}_W \mathbf{Z}$ can be replaced by the SVD of $\mathbf{H}_W \mathbf{Z}$, which can be found in $O(DNC)$ time. Finally, $\mathbf{W}$ can be found in $O(NC^2)$. As a result, the process can be done in $O(DNC)$ time. In the same manner as other methods, applying the transform to a given vector can be done in $O(Dd)$ time.
Chapter 2

The main flaw of this method is it discards the nullspace of $S_B$ [10], with no regard for how discriminative this information may be. In addition to that, the method relies on finding the smallest values in a set of non-negative values, which can result in multiple zeroes. There is no way to distinguish between these zeroes. The claim that this method is equivalent to LDA is only valid when $D \leq c - 1$, in which case, there is little to gain from dimensionality reduction.

2.2.8 Psuedo-Linear Discriminant Analysis

Another suggested method to bypass the small sample size problem is to use the pseudo-inverse of the matrix $S_W$ in place of the actual inverse, and it has been titled Psuedo-Linear Discriminant Analysis (pLDA) [22]. This allows a similar treatment as Direct Linear Discriminant Analysis, but without some of the drawbacks. The method is mostly similar, in that diagonalization can still be utilized to make the process more efficient.

To efficiently perform pLDA, diagonalization can be used. First, $S_W$ is decomposed using Singular Value Decomposition (SVD). This gives $V_W D_W V_W^T$, where $D_W$ is a diagonal matrix consisting only of non-zero terms and $V_W V_W^T = I$. Then, set $Z = V_W D_W^{-1/2}$ and decompose $Z^T S_B Z$, finding $Z^T S_B Z = V_B D_B V_B^T$, where $V_B V_B^T = I$ and $D_B$ is a diagonal matrix consisting only of non-zero terms. Select the rows of $V_B$ with the largest corresponding rows of $D_B$, and combine them to make $U$. The transformation matrix is then $W = D_B^{-1/2} U Z^T$.

Decomposing $S_W$ can be done using $H_W$, and therefore will take $O(DN^2)$ time. Calculation of $Z$ can be done in $O(DN^2)$ time. Instead of decomposing $Z^T S_B Z$, we decompose $H_B Z$ instead. Constructing $H_B$ can be done in $O(nD)$ time, decomposing $H_B Z$ can be done in $O(DNC)$ time, and constructing $W$ can be done in $O(DN^2)$ time. Training can therefore be done in $O(DN^2)$ time. In the same manner as LDA, applying the transform to a given vector can be done in $O(Dd)$ time.

The main flaw of this method is it discards the nullspace of $S_W$, with no regard for how discriminative this information may be. It can be shown
that the null space of $S_W$ contains more discriminative information than the range space [3].

### 2.2.9 Nullspace Linear Discriminant Analysis

Since the nullspace of $S_W$ is considered to contain more discriminative information, Chen et al [3] suggested that the Nullspace Linear Discriminant Analysis (NLDA) method, where they maximize the spread of the class means within the nullspace of $S_W$. Thus, it optimizes the Fisher Criterion with an infinite value.

To efficiently find the nullspace of $S_W$, SVD is used to decompose $H_W$ to find the eigenvectors and eigenvalues of $S_W$ [?]. The eigenvectors of $S_W$ that have a corresponding eigenvalue of 0 form the nullspace of $S_W$ [?]. Define the matrix $Q$ such that the rangespace of $Q$ is the nullspace of $S_W$, and $Q$ is orthonormal. Then nLDA is performed by finding the eigenvectors of $QQ^T S_B QQ^T$ with the largest matching eigenvalues [?]. This can be found by decomposing $H_B QQ^T$ using SVD. Overall this method requires $16DN^2 + 4DNC + O(N^3)$ floating point operations (FLOPs), however alternate methods have been proposed to reduce this while still maintaining accuracy [32, 5, 19].

Lu and Wang [19] propose a QR-based method to find the optimal features as follows. First, for each class, choose a sample from that class and use that sample in place of the class mean when constructing $H_W$. One column will consist entirely of zeros, remove that column. This creates the $D \times (N - C)$ matrix $H^{diff}_W$. Similarly, select one class and use that class mean in place of the global mean when constructing $H_B$. Remove the column consisting entirely of zeros and label the resulting matrix $H^{diff}_B$, which will be of size $D \times (C - 1)$. Use QR decomposition to find the following matrices

$$
\left[ H^{diff}_W \ H^{diff}_B \right] = \left[ \tilde{Q}_1 \ \tilde{Q}_2 \right] \left[ \begin{array}{cc} R_{11} & R_{12} \\ 0 & R_{22} \end{array} \right].
$$

Then, $\tilde{Q}_2$ can be used as the transformation matrix for NLDA. This method performs $4DN^2 + O(N^3)$ FLOPs. This method will be noted as
30

Chapter 2

QR-NLDA.

The main flaw of this method is it discards the rangespace of $S_W$, with no regard for how discriminative this information may be. Additionally, the performance drops when $N - C$ is close to $D$, as the dimensionality of the nullspace of $S_W$ is too small.

2.2.10 Regularized Linear Discriminant Analysis

Since following Fisher’s criterion exactly disregards the rangespace of $S_W$, and there is evidence that the rangespace contains information, Friedman [8] suggested a modification to the formula: the addition of the identity matrix multiplied by a small positive value to $S_W$. The resulting method is called Regularized Linear Discriminant Analysis (RLDA), and aims to maximize the criterion

$$
\hat{J}(W) = \text{trace}((W^T(S_W + \alpha I)W)^{-1}W^T S_B W).
$$

If $\alpha$ is too large, the resulting discriminative subspace will only separate the class means, with no regard for the spread of the data in each class [16]. If $\alpha$ is too small, it will degenerate into nLDA. The algorithm is highly sensitive to the value of $\alpha$ and very few techniques exist to efficiently find a suitable value.

The only method in the literature to improve upon the computational complexity of rLDA is to apply PCA to remove the nullspace of $S_T$ [28]. This process takes $O(DN^2)$ time. Finding the largest eigenvalue of $S_W$ is done by the decomposition of $H_W$, which can be done in $O(N^3)$ time. Calculation of the inverse of $S_W + \alpha I$ is done in $O(N^3)$ time. Multiplying this by $S_B$ can be done in $O(N^3)$ time. Calculating the eigenvectors and eigenvalues of $(S_W + \alpha I)^{-1}S_B$ is done in $O(N^3)$ time. Returning the data to the original, pre-PCA dimensionality requires $O(DNC)$ time. As a result, the process is done in $O(DN^2)$ time, since $D \gg N$.

The main flaw of this method is it is no longer aiming to maximize Fisher’s criterion, instead, it is diluting the impact of $S_W$. Also, there is no specific formula for $\alpha$. While it is agreed that it should be obtained as a fraction to
the largest eigenvalue of $S_W$, the fraction is still determined by cross-validation. While it is capable of obtaining results on par or better than other methods, it is often ignored due to these issues.

### 2.2.11 Maximum Margin Criterion Linear Discriminant Analysis

As another alternative to Fisher’s criterion, Li and Jiang [15] proposed Maximum Margin Criterion Linear Discriminant Analysis (MMCLDA), where, instead of optimizing the ratio of the two classes, the difference is optimized instead. This allows the rangespace and nullspace of both matrices to appear in the output. While Liu et al [17] showed several flaws in the original paper, the method is still valid, and has potential. The method replaces the Fisher criterion with the formula [15]

$$J(W) = W^T(S_B - S_W)W.$$ 

Later, Liu et al [16] added an extra modification: they subtracted a multiple of $S_W$ from $S_B$, giving the formula

$$J(W) = W^T(S_B - \beta S_W)W.$$ 

Cheng [4] has proposed two methods to calculate $\beta$.

$$\beta_T = \frac{\text{trace}(Q^T S_B Q)}{\text{trace}(Q^T S_W Q)} + \frac{\text{trace}(S_B)}{\text{trace}(S_W)},$$

$$\beta_F = \frac{\|Q^T S_B Q\|}{\|Q^T S_W Q\|} + \frac{\|S_B\|}{\|S_W\|},$$

where $Q$ is the top $d$ eigenvectors of $S_B - S_W$.

Experimental results have been collected for these two methods, and show that $\beta_T$ is more effective [4].

As was the case for RLDA and NLDA, the nullspace of $S_T$ is removed from the data using PCA [17]. This is done in $O(DN^2)$. The matrices $S_B$ and $S_W$ are calculated in the reduced space, taking $O(N^2C)$ time and $O(DN^2)$ time, respectively. Eigenanalysis of the difference of these matrices can be
done in $O(N^3)$ time. Converting the data to the original, pre-PCA dimensionality requires $O(DNC)$ time. The largest step is $O(DN^2)$ time, so the algorithm can be done in $O(DN^2)$ time since $D \gg N$ [17].

The main flaw of this method is that little research has been placed into calculating a value for $\beta$ [4]. While it is capable of obtaining results on par or better than other methods, it is dependent on a value that has not been explored thoroughly, and is often decided by cross-validation or heuristic analysis.

2.2.12 Generalized Discriminant Analysis

To overcome the need for Gaussian data with similar covariance matrices, Baudat and Anouar [2] applied the kernel matrix theory to the process, resulting in Generalized Discriminant Analysis (GDA). Since the kernel matrix is $N \times N$, and there are $N$ independent vectors being used, the kernel matrix also allows us to bypass the small sample size problem. As a result, GDA overcomes most of the problems of regular LDA.

Similar to kernel PCA, we zero-mean the kernel matrix by using a modified version of $K$ [2]

$$\hat{K} = K - 1K - K1 + 1K1$$

where $1$ is a $N \times N$ matrix with every element being equal to $1/N$.

Additionally, the matrix $W$ is introduced, and is defined as a $N \times N$ matrix such that [2]:

$$W_{ij} = \begin{cases} 
\frac{1}{N_k} & \text{if } x_i \in C_k, x_j \in C_k, \\
0 & \text{if } x_i \in C_{k_1}, x_j \in C_{k_2}, k_1 \neq k_2. 
\end{cases}$$

GDA then aims to maximize $J(\alpha)$ where

$$J(\alpha) = \frac{\alpha^T\hat{KW}\hat{K}\alpha}{\alpha^T\hat{K}\hat{K}\alpha}.$$
If we apply SVD to $\hat{K}$, we get

$$\hat{K} = \mathbf{P}\Gamma\mathbf{P}^T.$$  

The vector $\beta$ is defined such that [2]:

$$\beta = \Gamma\mathbf{P}^T\alpha.$$  

As a result, $J$ can be expressed as [2]:

$$J(\beta) = \frac{\beta^T\mathbf{P}^T\mathbf{W}\beta}{\beta^T\beta},$$

which can be solved as [2]:

$$\mathbf{P}^T\mathbf{W}\beta = \lambda\beta.$$  

We obtain the $d$ eigenvectors with the largest corresponding eigenvalues, convert them back to $\alpha$, and normalize them as [2]:

$$\hat{\alpha}_i = \frac{\alpha_i}{\sqrt{\alpha_i^T K \alpha_i}}.$$  

To train the system, first, $K$ needs to be calculated. Assuming that $k(x_i, x_j)$ can be found in $O(D)$ time, this will take $O(DN^2)$ time. Calculating $\mathbf{W}$ can be done in $O(N^2)$ time. Calculating $\hat{K}$ requires a series of matrix multiplications, and therefore takes $O(N^3)$ time. Applying SVD to $K$ requires $O(N^3)$ time. Assuming that $K$ is full rank, calculating $\mathbf{P}^T\mathbf{W}$ can be done in $O(N^3)$ time. Eigen-analysis of this matrix requires $O(N^3)$ time. The vector $\alpha$ can be calculated as $\mathbf{P}\Gamma^{-1}\beta$, which can be done in $O(N^2)$ time. The whole process requires $O(DN^2)$ time. Converting a given vector to the lower dimensionality requires $O(ndD)$ time.

While this thesis is focused on the small sample size problem, it is important to note that the above technique requires $O(N^3)$ FLOPs to run when the small sample size problem does not apply. In such a case, the matrix inversion and eigenvector calculation can be replaced with a gradient descent algorithm [1]. This allows the algorithm to run in $O(N^2D)$ time still. This algorithm can be applied to the small sample size...
problem, but no notable speed-up is introduced, and the program will usually be more complicated.

While GDA overcomes most of the problems of normal LDA, it faces some drawbacks. First, reducing the dimensionality of a given sample takes longer for GDA than other methods of LDA. Using the ORL database as an example, transforming a vector to the reduced space would require a number of computations of the order of $10^5$. With GDA, this increases to $10^8$. The second drawback appears in the need to choose a kernel, and find suitable parameters. There are very few methods to determine the best kernel and the suitable parameters, although a radial kernel is usually acceptable.

\subsection*{2.2.13 Non-parametric Discriminant Analysis}

Kernel LDA overcomes several of the problems presented by LDA, but it did not overcome the fact that we are limited to $c - 1$ features. This limitation is caused by the rank of $S_B$ only being $c - 1$ [9]. In addition to this, it is still desirable to overcome the limitations of Gaussian parameterization. With these constraints in mind, Fukunaga [9] put forward Non-parametric Discriminant Analysis (NDA). It was originally proposed for a two-class problem, although the extension to multiclass has been completed.

First, the vector $x_{j^{NNE}}^{(i)}$ is defined as the $j^{th}$ nearest neighbour to $x_i$ that is not in the same class as $x_i$. The vector $M_k(x_i)$ is defined as [9]

$$M_k(x_i) = \frac{1}{k} \sum_{j=1}^{k} x_{j^{NNE}}^{(i)}.$$  

Fukunaga [9] defined $\hat{S}_B$ as

$$\hat{S}_B = \frac{1}{N} \sum_{i=1}^{N} (x_i - M_k(x_i))(x_i - M_k(x_i))^T.$$  

However, Fukunaga also pointed a problem with this. A sample further from other classes has more weight than a closer sample, even though the
closer sample has more chance of being classified incorrectly [9]. Fukunaga suggested a weighting to bypass this. Define the vector \( x_{jNNI}^{(i)} \) as the \( j \)th nearest neighbour to \( x_i \) that is in the same class as \( x_i \). The weight, \( w_i \), is defined as

\[
\begin{align*}
w_i = & \min\left( \sqrt{\|x_i - x_{kNNI}^{(i)}\|^{\alpha}}, \sqrt{\|x_i - x_{kNNE}^{(i)}\|^{\alpha}} \right) \times \sqrt{\|x_i - x_{kNNI}^{(i)}\|^{\alpha}} + \sqrt{\|x_i - x_{kNNE}^{(i)}\|^{\alpha}}, \\
& \sqrt{\|x_i - x_{kNNI}^{(i)}\|^{\alpha}} + \sqrt{\|x_i - x_{kNNE}^{(i)}\|^{\alpha}} \end{align*}
\]

where \( \alpha \) is non-negative control parameter. If \( x_i \) is close to the decision boundary, \( w_i \) approaches 0.5. As \( x_i \) heads away from the decision boundary, \( w_i \) approaches 0. The control parameter, \( \alpha \), determines how gradual this change is.

With the weight \( w_i \) added, \( \hat{S}_B \) is modified to:

\[
\hat{S}_B = \frac{1}{N} \sum_{i=1}^{N} w_i (x_i - M_k(x_i))(x_i - M_k(x_i))^T.
\]

Fukunaga [9] presents no change to \( S_W \), although the literature contains some modifications.

Assuming that no modifications are made to this algorithm, the small sample size is not overcome. As a result, \( N > D \). Calculating \( S_W \) requires \( O(D^2N) \) time. Determining \( x_{kNNI}^{(i)} \) and \( x_{kNNE}^{(i)} \) requires a total of \( O(DN) \) for each vector, for a total of \( O(DN^2) \) time. The Delaunay Triangulation and R-Trees methods can be used to reduce the average computational complexity of this step, but has no guarantees on the worst case [13, 23].

All weights can be found in \( O(N) \) time. Calculating all relevant \( M_k(x_i) \) values can be done in \( O(kND) \) time. With these values calculated, \( \hat{S}_B \) requires \( O(D^2N) \) operations. This results in two \( D \times D \) matrices, and we wish to find the eigenvalues of \( S_W^{-1}\hat{S}_B \), which can be done in \( O(D^3) \) time. As a result, the overall time taken to find \( W \) is \( O(DN^2) \).

While NDA overcomes the problems of requiring Gaussian nature with similar covariance matrices, and being limited to \( c - 1 \) feature vectors, it fails to overcome the small sample size problem. Other methods may be applied to correct this, such as nLDA, rLDA and MMC LDA. The requirement to calculate all distances is taxing, which makes NDA slower.
Chapter 2

than LDA [9]. Also, NDA relies on 2 parameters: \( k \) and \( \alpha \). The literature offers little aid in determining these values, so they are often chosen by cross-validation, which implies that further room for optimization is available.

\[ \text{2.2.14 Two-dimensional Linear Discriminant Analysis} \]

In the event that we are classifying matrices as opposed to vectors, 2-dimensional techniques similar to those used in 2DPCA can be applied to LDA. The resulting technique is 2-Dimensional Linear Discriminant Analysis (2DLD) [30]. Since the matrices \( S_B \) and \( S_W \) are now significantly smaller, this allows us to bypass the SSS problem represented by face images.

Given a set of matrices, \( \{X_i|1 \leq i \leq N\} \), and class labels for each of them \( \{y_i|1 \leq i \leq N\} \), such that \( y_i \in \{c_j|1 \leq j \leq C\} \), \( S^R_B \) and \( S^R_W \) are defined as follows [30]:

\[
M_c = \frac{1}{N_c} \sum_{y_i = c} X_i.
\]

\[
S^R_B = \frac{1}{N} \sum_{i=1}^{C} N_i M_i M_i^T.
\]

\[
S^R_W = \frac{1}{N} \sum_{i=1}^{C} \sum_{y_j = i} (X_i - M_j)(X_i - M_j)^T.
\]

The matrix \( L \) can be formed by solving the eigenvector equation:

\[
S^R_B = \lambda S^R_W L.
\]

and using the \( d \) eigenvectors with the largest corresponding eigenvalues to construct a matrix.
From this, $S_{B}^{L}$ and $S_{W}^{L}$ can be constructed as:

$$S_{B}^{L} = \frac{1}{N} \sum_{i=1}^{C} N_{i} M_{i}^{T} LL^{T} M_{c}$$

$$S_{W}^{L} = \frac{1}{N} \sum_{i=1}^{C} \sum_{y_{j} \in C_{i}} (X_{i} - M_{j})^{T} LL^{T} (X_{i} - M_{j}).$$

The matrix $R$ can be formed by solving the eigenvector equation:

$$S_{B}^{L} \mathbf{r} = \lambda S_{W}^{L} \mathbf{r}$$

and using the $d$ eigenvectors with the largest corresponding eigenvalues to construct a matrix.

Then, to apply dimensionality reduction, $\hat{X}_{i} = L^{T} X_{i} R$. LDA is applied to this reduced space in a similar manner to the application of PCA in 2DPCA.

The computational complexity of 2DLDA shall be calculated assuming we start with $N$ training samples of size $m \times n$, reduced to $q \times q$ through 2DLDA and then applying regular LDA to reduce it to $d$ features. The calculation of the class means can be done in $O(Nmn)$ time. Calculating $S_{B}^{R}$ requires $c$ multiplications of $m \times n$ by $n \times m$ matrices, totalling $O(cm^{2}n)$ time. Calculating $S_{W}^{R}$ requires $N$ sets of similar calculations, so therefore, requires $O(Nm^{2}n)$ time. Calculating the eigenvectors of $(S_{B}^{R})^{-1} S_{B}^{L}$ can be done in $O(m^{3})$ time. Calculating $M_{i}^{T} L$ requires $O(mnq)$ time, from which, calculating $S_{B}^{L}$ requires $(cn^{2}q)$ time. Similarly, calculating $S_{W}^{L}$ requires $O(Nn^{2}q)$ time. From this, calculating the eigenvectors of $(S_{W}^{L})^{-1} S_{B}^{L}$ can be done in $O(n^{3})$ time. Therefore, calculating $L$ and $R$ can be done in $O(Nm^{2}n)$. This leaves us with $N$ samples each with $q^{2}$ features, which we wish to reduce to $d$ dimensions. This can be done in $O(Nq^{4})$. Generally speaking, $q^{4} < m^{2}n$, so training the system can be done in $O(Nm^{2}n)$ time. Reducing the dimensionality of a sample can be done in $O(mn(q + d))$ time.

The drawbacks of this method are similar to the drawbacks of 2DPCA. It is limited to matrix-structured data, meaning its application is limited. Additionally, it can only rely on classification information found in the
rows or columns, and cannot find information that is scattered amongst various columns and rows. Furthermore, 2DLDA, without the additional LDA stage, requires more co-efficients to achieve similar results to other methods, although the additional LDA stage helps correct this [30].

2.2.15 Databases

A large number of databases are used to compare methods. In this thesis, we will limit the databases used to the ORL face database, the Yale face database and the Feret face database.

The ORL database [7] contains 400 samples evenly distributed between 40 classes. Each class contains 10 photos of a given individual’s face, and each individual’s face only appears in one class. Classification is therefore built towards recognizing a person by an image of their face. The photos vary in lighting, facial expressions (smiling, frowning, open/closed eyes) and facial details (glasses/no glasses). For all pictures, the background is constant, and the subject always looks towards the camera. Each image has a width of 92 pixels and a length of 112 pixels, totalling 10304 pixels. All 10304 pixels are used as features.

The Yale database [11] contains 165 samples evenly distributed between 15 classes. Each class contains 11 photos of a given individual’s face, and each individual’s face only appears in one class. Classification is therefore built towards recognizing a person by an image of their face. For each subject, the 11 photos contain 3 photos with varying light conditions: strong light from the left, the right, and the center, 6 with varying facial expressions: normal, happy, sad, surprised, sleepy and winking, and 2 with varying facial details: glasses and no glasses. In all cases, the subject looks towards the camera, and the background is constant. The database was modified to rotate and center the face where necessary. The modified images have widths of 51 pixels and lengths of 65 pixels, totalling 3315 pixels.

The Feret database [20] contains 1400 samples evenly distributed between 100 classes. Each class contains 14 photos of a given individual’s face, and each individual’s face only appears in one class. Classification is therefore built towards recognizing a person by an image of their face. For each
subject, the 14 photos vary in facial expression, lighting conditions. In all cases, the subject looks towards the camera, and the background is constant. The database was modified to rotate and center the face where necessary. If a subject wore glasses for any picture, they wore glasses for all pictures. The modified images have widths of 60 pixels and lengths of 83 pixels, totalling 4980 pixels.
Chapter 3

A Comparison of Linear Discriminant Analysis Algorithms for the Small Sample Size Problem

3.1 Introduction

As already discussed, Linear Discriminant Analysis (LDA) suffers the problem that it cannot cope with the Small Sample Size [6, 9], and a variety of methods have been introduced to correct this [6, 9, 27, 33, 22, 3, 8, 15]. The methods have different aims and in some cases create vector spaces that are orthonormal to other ones (for example, the vectors produced by PLDA are orthonormal to the vectors produced by NLDA) [22, 3]. With such a variety of available techniques, it is desirable to know how they perform in a variety of conditions.

In this chapter, we examine the accuracy of each given method with a variety of databases and under some modifications to examine varying effects. These effects include varying the number of classes, varying the number of samples in each class, adding noise to the training data, and adding noise to the testing data. This allows us to see not only which
algorithms are better in general, but also what algorithms are better in a
niche application (for example, when we have an abundance of training
data or a low number of classes). We will be using the Feret Database [20],
the ORL Database [7] and the Yale Database [11]. We will examine
methods the Direct Linear Discriminant Analysis (DLDA) method [33],
the Psuedo-Linear Discriminant Analysis (PLDA) method [22], the
Nullspace LDA (NLDA) method [3], the Maximum Margin Criterion LDA
(MMCLDA) method [15] and the Regularized LDA (RLDA) method [8].

This chapter is seperated as follows. In Section 3.2, we will compare the
methods with no change to the data. In Section 3.3, we will compare the
methods with a varying number of classes. In Section 3.4, we will compare
the methods with a varying number of training samples per class. In
Section 3.5, we will compare the methods when noise has been added to
the training data. In Section 3.6, we will compare the methods when noise
has been added to the testing data. Finally, in Section 3.7, we will
conclude the chapter.

3.2 Results for Original Data

In this section, we take half the data for training and half the data for
testing. We compare the accuracy of the methods under these
circumstances. We record the result using all the produced features, and
with half the features used (rounding up). While it is rare to use half the
features, it does give an indication of the tolerance to noise. For RLDA, we
have tested with the value of $\alpha$ being equal to $0.01 \times \lambda$, where $\lambda$ is the
largest eigenvalue of $S_T$.

As can be seen, NLDA consistently outperforms all other methods, with
MMC LDA (using $\beta_T$) and RLDA being reasonable replacements. When
performing MMC LDA, using the value of $\beta_T$ appears to perform better
than the value of $\beta_F$. 
Table 3.1: Accuracy of a variety of methods under normal conditions

<table>
<thead>
<tr>
<th></th>
<th>Direct LDA</th>
<th>Psuedo LDA</th>
<th>NLD A</th>
<th>MMC LDA with $\beta_T$</th>
<th>MMC LDA with $\beta_F$</th>
<th>RLDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feret (all)</td>
<td>74.9%</td>
<td>73.4%</td>
<td>84%</td>
<td>75.6%</td>
<td>70.1%</td>
<td>76.4%</td>
</tr>
<tr>
<td>ORL (all)</td>
<td>89%</td>
<td>89.5%</td>
<td>91%</td>
<td>89%</td>
<td>79%</td>
<td>90%</td>
</tr>
<tr>
<td>Yale (all)</td>
<td>89.3%</td>
<td>88%</td>
<td>97.3%</td>
<td>96%</td>
<td>77.3%</td>
<td>93.3%</td>
</tr>
<tr>
<td>Feret (half)</td>
<td>81.4%</td>
<td>70%</td>
<td>76.9%</td>
<td>72.1%</td>
<td>64%</td>
<td>71.1%</td>
</tr>
<tr>
<td>ORL (half)</td>
<td>67.5%</td>
<td>87%</td>
<td>87.5%</td>
<td>88%</td>
<td>74.5%</td>
<td>87%</td>
</tr>
<tr>
<td>Yale (half)</td>
<td>85.3%</td>
<td>85.3%</td>
<td>93.3%</td>
<td>93.3%</td>
<td>93.3%</td>
<td>82.7%</td>
</tr>
</tbody>
</table>

3.3 Results for Reduced Number of Classes

In this section, we take half the data for training and half the data for testing. Furthermore, we only use the first 50% of classes, ignoring the other half. Other than that, this is identical to the previous section. For reference, the Feret database originally contained 100 classes [20], the ORL database originally contained 40 classes [7], and the Yale database originally contained 15 classes [11]. After reduction, they contain 50, 20 and 8 classes, respectively.

Table 3.2: Accuracy of a variety of methods with half the number of classes

<table>
<thead>
<tr>
<th></th>
<th>Direct LDA</th>
<th>Psuedo LDA</th>
<th>NLD A</th>
<th>MMC LDA with $\beta_T$</th>
<th>MMC LDA with $\beta_F$</th>
<th>RLDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feret (all)</td>
<td>82.3%</td>
<td>78.6%</td>
<td>93.4%</td>
<td>88.9%</td>
<td>70.6%</td>
<td>84.3%</td>
</tr>
<tr>
<td>ORL (all)</td>
<td>94%</td>
<td>85%</td>
<td>95%</td>
<td>96%</td>
<td>75%</td>
<td>95%</td>
</tr>
<tr>
<td>Yale (all)</td>
<td>87.5%</td>
<td>90%</td>
<td>97.5%</td>
<td>97.5%</td>
<td>97.5%</td>
<td>90%</td>
</tr>
<tr>
<td>Feret (half)</td>
<td>84.6%</td>
<td>71.1%</td>
<td>84%</td>
<td>80.9%</td>
<td>60.3%</td>
<td>75.4%</td>
</tr>
<tr>
<td>ORL (half)</td>
<td>85%</td>
<td>83%</td>
<td>88%</td>
<td>87%</td>
<td>88%</td>
<td>92%</td>
</tr>
<tr>
<td>Yale (half)</td>
<td>72.5%</td>
<td>85%</td>
<td>92.5%</td>
<td>90%</td>
<td>92.5%</td>
<td>85%</td>
</tr>
</tbody>
</table>

When working with a smaller number of classes, MMC becomes a more viable option, although only when using $\beta_T$. The Feret database has significantly more classes, which would explain the larger difference between the results for that database in comparison to the others. Using only half the reduced feature space, not much changes.
3.4 Results for Reduced Number of Training Samples

In this section, we take half the data for training and half the data for testing as usual. Furthermore, we only use the first 2 supplied faces as training samples, leaving the 5 remaining as test samples. Other than that, this is identical to the Section 3.2. For reference, the Feret, ORL and Yale databases contain 14, 10 and 11 samples for each class, respectively.

<table>
<thead>
<tr>
<th></th>
<th>Direct LDA</th>
<th>Pseudo LDA</th>
<th>NLDA</th>
<th>MMC LDA with $\beta_T$</th>
<th>MMC LDA with $\beta_F$</th>
<th>RLDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feret (all)</td>
<td>62%</td>
<td>44.8%</td>
<td>66.3%</td>
<td>63.9%</td>
<td>44.4%</td>
<td>61.8%</td>
</tr>
<tr>
<td>ORL (all)</td>
<td>83.4%</td>
<td>64.1%</td>
<td>83.8%</td>
<td>85.3%</td>
<td>63.1%</td>
<td>84.4%</td>
</tr>
<tr>
<td>Yale (all)</td>
<td>88.9%</td>
<td>60.7%</td>
<td>88.9%</td>
<td>88.1%</td>
<td>58.5%</td>
<td>87.4%</td>
</tr>
<tr>
<td>Feret (half)</td>
<td>55.3%</td>
<td>43.2%</td>
<td>61.3%</td>
<td>58.1%</td>
<td>38.6%</td>
<td>56.8%</td>
</tr>
<tr>
<td>ORL (half)</td>
<td>73.8%</td>
<td>62.5%</td>
<td>81.9%</td>
<td>81.9%</td>
<td>55.3%</td>
<td>80.3%</td>
</tr>
<tr>
<td>Yale (half)</td>
<td>74%</td>
<td>59.3%</td>
<td>83.7%</td>
<td>83%</td>
<td>45.2%</td>
<td>77.7%</td>
</tr>
</tbody>
</table>

Table 3.3: Accuracy of a variety of methods with 2 training samples per class

When working with a smaller number of training samples, Direct LDA is competitive. This is most likely because the nullspace of $S_B$ that doesn’t overlap with the nullspace of $S_T$ is significantly smaller, so we are excluding less of the data. Since DLDA has a more efficient training time, there may be applications where DLDA is worth using if the number of training samples is low and the training time needs to be fast. Other than that, NLDA is generally speaking better than the alternatives, however, MMC is still quite competitive when using $\beta_T$ and RLDA is still a reasonable alternative.

3.5 Results for Increased Number of Training Samples

In this section, we take half the data for training and half the data for testing as usual. Furthermore, we only use the last supplied faces as test samples, leaving the majority of given samples as training samples. Other
than that, this is identical to the Section 3.4. For reference, the Feret, ORL and Yale databases contain 14, 10 and 11 samples for each class, respectively.

<table>
<thead>
<tr>
<th></th>
<th>Direct LDA</th>
<th>Psuedo LDA</th>
<th>NLDA</th>
<th>MMC LDA with $\beta_T$</th>
<th>MMC LDA with $\beta_F$</th>
<th>RLDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feret (all)</td>
<td>83%</td>
<td>80%</td>
<td>99%</td>
<td>100%</td>
<td>70%</td>
<td>98%</td>
</tr>
<tr>
<td>ORL (all)</td>
<td>100%</td>
<td>95%</td>
<td>97.5%</td>
<td>95%</td>
<td>95%</td>
<td>95%</td>
</tr>
<tr>
<td>Yale (all)</td>
<td>100%</td>
<td>93.3%</td>
<td>100%</td>
<td>100%</td>
<td>70%</td>
<td>100%</td>
</tr>
<tr>
<td>Feret (half)</td>
<td>100%</td>
<td>77%</td>
<td>99%</td>
<td>100%</td>
<td>61%</td>
<td>93%</td>
</tr>
<tr>
<td>ORL (half)</td>
<td>95%</td>
<td>95%</td>
<td>95%</td>
<td>95%</td>
<td>95%</td>
<td>95%</td>
</tr>
<tr>
<td>Yale (half)</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>61%</td>
<td>93.3%</td>
</tr>
</tbody>
</table>

Table 3.4: Accuracy of a variety of methods with an increased number of training samples per class

When working with a larger number of training samples, neither NLDA nor MMC with $\beta_T$ are strictly better than the other, however both are better than other methods.

### 3.6 Results for Noisy Training Data

In this section, we repeat the procedure for Section 2, but this time, add noise to the training samples. The noise was Gaussian with 0 mean and a standard deviation equal to 3 times the standard deviation of all values in the training data. This results in a mostly noisy image, barely recognizable by the naked eye.

<table>
<thead>
<tr>
<th></th>
<th>Direct LDA</th>
<th>Psuedo LDA</th>
<th>NLDA</th>
<th>MMC LDA with $\beta_T$</th>
<th>MMC LDA with $\beta_F$</th>
<th>RLDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feret (all)</td>
<td>46.6%</td>
<td>3.5%</td>
<td>69.6%</td>
<td>69.9%</td>
<td>1.9%</td>
<td>50.3%</td>
</tr>
<tr>
<td>ORL (all)</td>
<td>80%</td>
<td>5.5%</td>
<td>84%</td>
<td>84%</td>
<td>5.5%</td>
<td>59%</td>
</tr>
<tr>
<td>Yale (all)</td>
<td>78.7%</td>
<td>10.7%</td>
<td>89.3%</td>
<td>89.3%</td>
<td>6.7%</td>
<td>50.7%</td>
</tr>
<tr>
<td>Feret (half)</td>
<td>8.6%</td>
<td>6.4%</td>
<td>19.7%</td>
<td>20.3%</td>
<td>1.7%</td>
<td>18.7%</td>
</tr>
<tr>
<td>ORL (half)</td>
<td>5.5%</td>
<td>2.5%</td>
<td>21%</td>
<td>22%</td>
<td>2.2%</td>
<td>30.5%</td>
</tr>
<tr>
<td>Yale (half)</td>
<td>21.3%</td>
<td>12%</td>
<td>29.3%</td>
<td>29.3%</td>
<td>8%</td>
<td>8%</td>
</tr>
</tbody>
</table>

Table 3.5: Accuracy of a variety of methods with noisy training data
MMC is marginally better with noisy training data, but NLDA is still a reasonable option. Direct LDA surprisingly outperforms most other methods.

### 3.7 Results for Noisy Testing Data

In this section, we repeat the procedure for Section 3.6, but this time, add noise to the test samples instead of the training samples. The noise was Gaussian with 0 mean and a standard deviation equal to 3 times the standard deviation of all values in the testing data. This results in a mostly noisy image, barely recognizable by the naked eye.

<table>
<thead>
<tr>
<th></th>
<th>Direct LDA</th>
<th>Pseudo LDA</th>
<th>NLDA</th>
<th>MMC LDA with ( \beta_T )</th>
<th>MMC LDA with ( \beta_F )</th>
<th>RLDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feret (all)</td>
<td>70.6%</td>
<td>69.7%</td>
<td>27.4%</td>
<td>60.6%</td>
<td>63.6%</td>
<td>66%</td>
</tr>
<tr>
<td>ORL (all)</td>
<td>89%</td>
<td>83.5%</td>
<td>85.5%</td>
<td>88.5%</td>
<td>78.5%</td>
<td>89.5%</td>
</tr>
<tr>
<td>Yale (all)</td>
<td>88%</td>
<td>84%</td>
<td>92%</td>
<td>89.3%</td>
<td>69.3%</td>
<td>89.3%</td>
</tr>
<tr>
<td>Feret (half)</td>
<td>42%</td>
<td>64.7%</td>
<td>22%</td>
<td>57.3%</td>
<td>57.7%</td>
<td>71.6%</td>
</tr>
<tr>
<td>ORL (half)</td>
<td>75%</td>
<td>80%</td>
<td>83%</td>
<td>86.5%</td>
<td>75%</td>
<td>86%</td>
</tr>
<tr>
<td>Yale (half)</td>
<td>73.3%</td>
<td>73.3%</td>
<td>81.3%</td>
<td>82.7%</td>
<td>81.3%</td>
<td>78.7%</td>
</tr>
</tbody>
</table>

Table 3.6: Accuracy of a variety of methods with noisy testing data

In general, MMC and RLDA are the most robust methods to noise in the testing data.

### 3.8 Conclusions

In this section, we have investigated several of the techniques introduced in Chapter 2 under a variety of different conditions. Generally speaking, NLDA has been the best method, however MMC LDA has given more accurate results in some situations, especially when noise was added to the system. Also, when using MMC LDA, the results were significantly more accurate when the value of \( \beta \) was set to \( \beta_T \). Further research should be restricted to MMC LDA, NLDA and possibly RLDA.
Chapter 4

Efficient Implementation of LDA-based methods for Small Sample Size problems

4.1 Introduction

As already discussed, LDA suffers the problem that it cannot cope with the Small Sample Size \([6, 9]\), and a variety of methods have been introduced to correct this. Many such algorithms take a common framework introduced by Yang and Yang \([28]\), where the algorithm first uses SVD to remove the nullspace of \(S_T\) and then an optimal transform is found in the reduced space. In this chapter, we introduce a new method of performing these steps by replacing this framework with one focused on keeping the rangespace of \(S_T\), which is equivalent to removing the nullspace \([12]\).

In this chapter, we introduce a new and fast implementation for many SSS-based LDA methods that does not perform eigendecomposition or SVD on \(D \times N\) matrices. We replace these operations with 2 matrix multiplications. Matrix multiplication is one of the least computationally expensive procedures available to perform on matrices \([12]\). Therefore, the proposed algorithm is less computationally expensive than the current algorithms for NLDA. This allows algorithms such as MMC LDA and RLDA to obtain improved efficiency while still keeping the same accuracy.
While this method can be applied to most SSS-based LDA technique, we will only demonstrate using MMC LDA and RLDA. The experimental results for these algorithms should be sufficient to demonstrate the efficiency of the technique over all algorithms.

Of note, while this algorithm can be applied to Large Sample Size problems, it will typically increase the computational complexity except in unusual corner cases. The algorithm is specifically designed to improve upon the efficiency of Small Sample Size problems.

The rest of this chapter is structured as follows. Section 4.2 will contain algorithms for existing SSS-based LDA methods. Section 4.3 will contain the theory behind the alternate implementation introduced in this chapter. Section 4.4 will demonstrate the application of this method on MMC LDA. Section 4.5 will apply the same techniques to RLDA to demonstrate that the technique is not specific to MMC. Lastly, Section 4.6 will conclude the chapter.

### 4.2 Existing Algorithms

The general structure to these algorithms has remained the same since their initial development [28]. The literature covers few ways to make the process more efficient. We will demonstrate the existing and proposed methods on MMC-LDA, which aims to optimize the modified Fisher criteria [15]

\[
\hat{J}_{MMC}(w) = w^T(S_B - S_W)w.
\]

Additionally, a constraint is added such that \(w^Tw = 1\). With this constraint, the function \(\hat{J}_{MMC}(w)\) is maximized by the eigenvectors of \(S_B - S_W\) with the largest corresponding eigenvalues [15], that is:

\[
(S_B - S_W)w = \lambda w. 
\] (4.2)

Calculating the eigenvectors of these matrices directly is computationally inefficient, requiring \(O(D^3)\) time. The most common algorithm to efficiently perform this task is presented in Table 3.1, MMC LDA Algorithm 1 [28, 17, 12].
**MMC LDA Algorithm 1**

<table>
<thead>
<tr>
<th>Steps</th>
<th>FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Use reduced SVD to find $H_T = U\Sigma V^T$.</td>
<td>$6DN^2 + O(N^3)$</td>
</tr>
<tr>
<td>2) Find $\hat{H}_B = U^T H_B$ and $\hat{H}_W = U^T H_W$.</td>
<td>$2DN^2 + 2DNC$</td>
</tr>
<tr>
<td>3) Find $\hat{S}_B = \hat{H}_B \hat{H}_B^T$ and $\hat{S}_W = \hat{H}_W \hat{H}_W^T$.</td>
<td>$O(N^3)$</td>
</tr>
<tr>
<td>4) Calculate $\hat{W}$ as the eigenvectors of $\hat{S}_B - \hat{S}_W$.</td>
<td>$O(N^3)$</td>
</tr>
<tr>
<td>5) Calculate $W = U\hat{W}$.</td>
<td>$2DNC$</td>
</tr>
</tbody>
</table>

Table 4.1: **MMC LDA Algorithm 1**.

MMC LDA Algorithm 1 requires $8DN^2 + 4DNC + O(N^3)$ floating point operations (FLOPs). It has a space complexity of $O(DN)$ since the largest matrix used is $D \times N$.

The MMC LDA Algorithm 1 can be made more efficient by replacing the SVD computation in step 1 with the PCA trick [6, 12].

**MMC LDA Algorithm 1 with PCA trick**

<table>
<thead>
<tr>
<th>Steps</th>
<th>FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Use SVD to find $H_T^T H_T = V\Sigma V^T$.</td>
<td>$DN^2 + O(N^3)$</td>
</tr>
<tr>
<td>2) Find $U = H_T V^T \Sigma^{-\frac{1}{2}}$.</td>
<td>$2DN^2$</td>
</tr>
<tr>
<td>3) Find $\hat{H}_B = U^T H_B$ and $\hat{H}_W = U^T H_W$.</td>
<td>$2DN^2 + 2DNC$</td>
</tr>
<tr>
<td>4) Find $\hat{S}_B = \hat{H}_B \hat{H}_B^T$ and $\hat{S}_W = \hat{H}_W \hat{H}_W^T$.</td>
<td>$O(N^3)$</td>
</tr>
<tr>
<td>5) Calculate $\hat{W}$ as the eigenvectors of $\hat{S}_B - \hat{S}_W$.</td>
<td>$O(N^3)$</td>
</tr>
<tr>
<td>6) Calculate $W = U\hat{W}$.</td>
<td>$2DNC$</td>
</tr>
</tbody>
</table>

Table 4.2: **MMC LDA Algorithm 1 with PCA trick**.

MMC LDA Algorithm 1 with the PCA trick only requires $5DN^2 + 4DNC + O(N^3)$ FLOPs, an improvement of $3DN^2$ FLOPs.

### 4.3 Alternative Implementation

In this section, we will introduce the theory for the alternative implementation using MMC LDA. While we focus on MMC LDA, the method is generic and can be applied to most LDA-based methods with ease. As a demonstration of this, we have demonstrated using RLDA in Section 5.
It can be shown that, if \( \mathbf{w} \) is a discriminant vector for any LDA-based method, then there exists a \( \mathbf{p} \) such that \( \mathbf{w} = \mathbf{Up} \), where \( \mathbf{U} \) consists of the eigenvectors of \( \mathbf{S}_T \) with non-zero eigenvalues [28, 17]. The matrix \( \mathbf{U} \) can also be obtained from taking the thin SVD of \( \mathbf{H}_T^T \): \( \mathbf{H}_T^T = \mathbf{U}\Sigma \mathbf{V}^T \) [25, 12]. Note that \( \Sigma \) is a full rank diagonal matrix [25, 12], and \( \mathbf{U} \) and \( \mathbf{V} \) are both orthonormal [25, 12], and therefore, \( \mathbf{U}^T \mathbf{U} = \mathbf{V}^T \mathbf{V} = \mathbf{I} \) [25, 12]. Additionally, the pseudoinverse of \( \mathbf{H}_T \) can be expressed as \( \mathbf{H}_T^+ = \mathbf{V}\Sigma^{-1}\mathbf{U}^T \) [25, 12]. Therefore, we have the following derivation.

\[
\mathbf{w} = \mathbf{Up} \\
= \mathbf{UU}^T \mathbf{Up} \\
= \mathbf{UU}^T \mathbf{w} \\
= \mathbf{U}\Sigma\Sigma^{-1}\mathbf{U}^T \mathbf{w} \\
= \mathbf{H}_T \mathbf{H}_T^+ \mathbf{w} \\
= \mathbf{H}_T \mathbf{w}, \tag{4.3}
\]

where \( \mathbf{H}_T \mathbf{w} \). To clarify, it is unnecessary to calculate the pseudo-inverse of \( \mathbf{H}_T \) or any other \( D \)-dimensional matrix in this method.

Substituting Eq. (4.3) into Eq. (4.2) gives the following equation.

\[
(\mathbf{S}_B - \mathbf{S}_W)\mathbf{H}_T \mathbf{w} = \lambda \mathbf{H}_T \mathbf{w}.
\]

Premultiplying each side by \( \mathbf{H}_T^T \) gives:

\[
\mathbf{H}_T^T(\mathbf{S}_B - \mathbf{S}_W)\mathbf{H}_T \mathbf{w} = \lambda \mathbf{H}_T^T \mathbf{H}_T \mathbf{w}, \quad \text{or} \\
(\mathbf{H}_T^T \mathbf{S}_B \mathbf{H}_T - \mathbf{H}_T^T \mathbf{S}_W \mathbf{H}_T) \mathbf{w} = \lambda \mathbf{H}_T^T \mathbf{H}_T \mathbf{w}. \tag{4.4}
\]
Define the following 5 matrices by the following expressions.

\[ \hat{H}_B = H_T^T H_B, \]  
\[ \hat{H}_T = H_T^T H_T, \]  
\[ \hat{S}_B = H_T^T S_B H_T, \]  
\[ \hat{S}_W = H_T^T S_W H_T, \text{ and} \]  
\[ \hat{S}_T = H_T^T S_T H_T. \]

Furthermore, given \( \hat{H}_T \), it can be easily seen that \( \hat{H}_B \) can be calculated by the expression

\[ (\hat{H}_B)_{(i,j)} = \sqrt{N_j} \sum_{x_k \in C_j} (\hat{H}_T)_{(i,k)}. \]

This allows us to write Eq. (4.4) as:

\[ (\hat{S}_B - \hat{S}_W) \hat{w} = \lambda \hat{H}_T \hat{w}. \]

Since \( S_B = H_B H_B^T \) and \( S_T = H_T H_T^T \), it follows that:

\[ \hat{S}_B = H_T^T S_B H_T \]
\[ = H_T^T H_B H_B^T H_T \]
\[ = \hat{H}_B \hat{H}_B^T, \text{ and} \]
\[ \hat{S}_T = H_T^T S_T H_T \]
\[ = H_T^T H_T H_T^T H_T \]
\[ = (H_T^T H_T)^2 \]
\[ = \hat{H}_T^2. \]

Since \( S_W = S_T - S_B \) [6], it follows that

\[ S_W = S_T - S_B, \]
\[ H_T^T S_W H_T = H_T^T S_T H_T - H_T^T S_B H_T; \text{ that is} \]
\[ \hat{S}_W = \hat{S}_T - \hat{S}_B. \]

From this, we write the algorithm MMC LDA Algorithm 2.
Chapter 4

**MMC LDA Algorithm 2**

<table>
<thead>
<tr>
<th>Steps</th>
<th>FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Calculate $\hat{H}_T = H_j^T H_T$.</td>
<td>$D N^2$</td>
</tr>
<tr>
<td>2) Calculate $\hat{H}_B$ as described in Eq. (4.10).</td>
<td>$O(N^2)$</td>
</tr>
<tr>
<td>3) Calculate $\hat{S}_B = \hat{H}_B \hat{H}_B^T$ and $\hat{S}_T = \hat{H}_T^2$.</td>
<td>$O(N^3)$</td>
</tr>
<tr>
<td>4) Calculate $\hat{S}_W = \hat{S}_T - \hat{S}_B$.</td>
<td>$O(N^2)$</td>
</tr>
<tr>
<td>5) Solve $(\hat{S}_B - \hat{S}_W)\hat{w} = \lambda \hat{H}_T \hat{w}$ for $\hat{w}$.</td>
<td>$O(N^3)$</td>
</tr>
<tr>
<td>6) Calculate $w = H_T \hat{w}$.</td>
<td>$2DNC$</td>
</tr>
</tbody>
</table>

Table 4.3: **MMC LDA Algorithm 2.**

In Step 5, only the eigenvectors with finite, non-zero corresponding eigenvalues are kept.

MMC LDA Algorithm 2 requires $D N^2 + 2DNC + O(N^3)$ FLOPs. This is a notable upgrade over the $5D N^2 + 4DNC + O(N^3)$ FLOPs and $8D N^2 + 4DNC + O(N^3)$ FLOPs required by MMC LDA Algorithm 1 with and without the PCA trick, respectively.

The MMC LDA Algorithm 2 requires matrices of maximum size $D \times N$, the same as the MMC LDA Algorithm 1. While we have only demonstrated using MMC LDA, the same method can be applied to other LDA-based methods aimed at overcoming the SSS problem. Further improvements can be created with Strassen’s Algorithm [26], allowing the algorithm to run in $O(D N^{1.8})$.

### 4.4 Experimental Results

In this section, we show experimental results using the alternate method. To show that the algorithms are equivalent, we calculate the accuracy of each algorithm over several databases. We only document the results using the maximum number of features, as this is the most common situation. These results are documented in Table 4.4.

Furthermore, for each database, we also compared the values of the optimization criteria $\text{trace}(W^T (S_B - S_W) W) / \text{trace}(W^T W)$. Using the Feret database, the optimization criteria were equal to $4.5 \times 10^7$ for all 3 methods. Using the
Chapter 4

Error Rates of MMC-LDA Algorithms

<table>
<thead>
<tr>
<th>Database</th>
<th>Algorithm 1</th>
<th>Algorithm 1</th>
<th>Algorithm 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Matlab SVD</td>
<td>PCA-trick</td>
<td></td>
</tr>
<tr>
<td>Feret</td>
<td>24.1%</td>
<td>24.1%</td>
<td>24.1%</td>
</tr>
<tr>
<td>ORL</td>
<td>10.5%</td>
<td>10.5%</td>
<td>10.5%</td>
</tr>
<tr>
<td>Yale</td>
<td>5.3%</td>
<td>5.3%</td>
<td>5.3%</td>
</tr>
</tbody>
</table>

Table 4.4: Accuracy of MMC LDA with varying algorithms.

ORL database, the optimization criteria were equal to $4.8 \times 10^7$ for all 3 methods. Finally, using the Yale database, the results were equal to $3.6 \times 10^7$ for all 3 methods. The difference between value of the optimization criteria were less than $10^{-6}$ for all cases.

We ran the method over 3 face databases, using Algorithm 1 with MATLAB’s implementation of SVD, Algorithm 1 with the PCA trick, and Algorithm 2 for MMC LDA. Data was split evenly between testing and training to determine the equivalence of each method. For each database, we ran each method 10 times and averaged the computational time for those 10 runs. The experiments were run on a Intel(R) Core(TM)2 Quad CPU with a CPU running at 2.83 GHz. The results are documented in Table 4.4.

<table>
<thead>
<tr>
<th>Database</th>
<th>Computational Times of MMC-LDA Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Algorithm 1</td>
</tr>
<tr>
<td></td>
<td>Matlab SVD</td>
</tr>
<tr>
<td>Feret</td>
<td>6.745s</td>
</tr>
<tr>
<td>ORL</td>
<td>0.9565s</td>
</tr>
<tr>
<td>Yale</td>
<td>0.1219s</td>
</tr>
</tbody>
</table>

Table 4.5: Computational time of MMC LDA with varying algorithms.

Finally, we demonstrate the effects of varying the number of training samples and the number of features in regards to computational time, by plotting the comparisons in Figure 4.1 and Figure 4.2. In Figure 4.1, the number of features is held constant at 10304, the number of classes is held constant at 40, and the number of training samples is varied from 40 to 360, such that the training samples are evenly distributed between each class. In Figure 4.2, the number of training samples is held constant at
200, the number of classes is held constant at 40, and the number of features in each sample is varied from 500 to 10000. In each case, the data is a subset of the ORL database.

Figure 4.1: The computational time required by the three MMC-LDA algorithms for varying amounts of training samples.

Figure 4.2: The computational time required by the three MMC-LDA algorithms for varying amounts of features.

4.5 Application to Regularized LDA

As an additional example of how to apply this algorithm to LDA-based techniques, we apply it to RLDA. The aim of RLDA is to maximize the following function [8].

$$J_{RLDA}(w) = \frac{w^T S_B w}{w^T (S_W + \alpha I) w}. \quad (4.11)$$

The current method to efficiently calculate RLDA is described in Table 4.5.
Chapter 4

RLDA Algorithm 1 requires $8DN^2 + 4DCN + O(N^3)$ FLOPs. The PCA trick can be applied to create the RLDA Algorithm 1 with PCA trick. The RLDA Algorithm 1 with PCA trick requires $5DN^2 + 4DCN + O(N^3)$ FLOPs.

<table>
<thead>
<tr>
<th>Steps</th>
<th>FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Use reduced SVD to find $H_T = U\Sigma V^T$.</td>
<td>$6DN^2 + O(N^3)$</td>
</tr>
<tr>
<td>2) Find $\hat{H}_B = U^T H_B$ and $\hat{H}_W = U^T H_W$.</td>
<td>$2DN^2 + 2DCN$</td>
</tr>
<tr>
<td>3) Find $\hat{S}_B = \hat{H}_B \hat{H}_B^T$ and $\hat{S}_W = \hat{H}_W \hat{H}_W^T$.</td>
<td>$O(N^3)$</td>
</tr>
<tr>
<td>4) Calculate $\hat{W}$ as the eigenvectors of $(\hat{S}_W + \alpha I)^{-1} \hat{S}_B$.</td>
<td>$O(N^3)$</td>
</tr>
<tr>
<td>5) Calculate $W = U\hat{W}$.</td>
<td>$2DCN$</td>
</tr>
</tbody>
</table>

Table 4.6: RLDA Algorithm 1.

Similar to MMC, it has been shown that the solution to this will be of the following form [8, 27].

\[
\mathbf{w} = U\mathbf{p},
\]

where $H_T = U\Sigma V^T$ is the thin SVD of $H_T$. By definition of the thin SVD, the matrices $U$ and $V$ are orthonormal matrices of sizes $D \times R$ and $R \times N$ respectively, and the matrix $\Sigma$ is a full-rank diagonal matrix of size $R \times R$, where $R$ is the rank of the matrix $H_T$ [12]. From this, we find that Eq. (4.3) applies to RLDA as well. Substituting this into Eq. (4.11) gives

\[
\hat{J}_{RLDA}(\mathbf{w}) = \frac{\hat{w}^T H_T^T S_B H_T \hat{w}}{\hat{w}^T H_T^T (S_W + \alpha I) H_T \hat{w}}.
\]

This is optimized by the generalized eigenvalue equation

\[
H_T^T S_B H_T \hat{w} = \lambda H_T^T (S_W + \alpha I) H_T \hat{w}.
\]

These matrices will be singular, since $H_T$ is not full rank. This leads to an infinite solution found in the nullspace of $H_T$. Therefore, the nullspace can be discarded, and we accomplish this by taking the pseudo-inverse first.
Therefore
\[
[H_T^T(S_W + \alpha I)H_T]^{+}H_T^T S_B H_T \hat{w} = \lambda \hat{w}, \text{ or} \\
(H_T^T S_W H_T + \alpha H_T^T H_T)^{+}H_T^T S_B H_T \hat{w} = \lambda \hat{w}.
\]

Using the matrices introduced in Eq. (4.5), Eq. (4.6), Eq. (4.7), Eq. (4.8), and Eq. (4.9), we develop RLDA Algorithm 2, shown in Table 4.5. RLDA Algorithm 2 requires $DN^2 + 2DNC + O(N^3)$ FLOPs.

\begin{center}
RLDA Algorithm 2
\begin{tabular}{ll}
Steps & FLOPs \hline
1) Calculate $H_T = H_T^T H_T$ & $DN^2$ \\
2) Calculate $\hat{H}_B$ as described in Eq. (4.10) & $O(N^2)$ \\
3) Calculate $\hat{S}_B = \hat{H}_B \hat{H}_B^T$ and $\hat{S}_T = \hat{H}_T^2$ & $O(N^3)$ \\
4) Calculate $\hat{S}_W = \hat{S}_T - \hat{S}_B$ & $O(N^2)$ \\
5) Calculate $\hat{W}$ as the eigenvectors of $(\hat{S}_W + \alpha \hat{H}_T)^{+} \hat{S}_B$. & $O(N^3)$ \\
6) Calculate $W = H_T \hat{W}$. & $2DNC$
\end{tabular}
\end{center}

Table 4.7: RLDA Algorithm 2.

As with MMC LDA, we implemented these algorithms using Matlab to compare the accuracy and efficiency of the three algorithms. The accuracy is documented in Table 4.5, and the computational times are documented in Table 4.5.

\begin{center}
<table>
<thead>
<tr>
<th>Database</th>
<th>Error Rates of RLDA Algorithms</th>
<th>Algorithm 1</th>
<th>Algorithm 1</th>
<th>Algorithm 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feret</td>
<td>Matlab SVD</td>
<td>24.1%</td>
<td>24.1%</td>
<td>24.1%</td>
</tr>
<tr>
<td>ORL</td>
<td>Matlab SVD</td>
<td>10.5%</td>
<td>10.5%</td>
<td>10.5%</td>
</tr>
<tr>
<td>Yale</td>
<td>Matlab SVD</td>
<td>2.7%</td>
<td>2.7%</td>
<td>2.7%</td>
</tr>
</tbody>
</table>
\end{center}

Table 4.8: Accuracy of RLDA with varying algorithms.
### Table 4.9: Computational time of RLDA with varying algorithms.

<table>
<thead>
<tr>
<th>Database</th>
<th>Algorithm 1 Matlab SVD</th>
<th>Algorithm 1 PCA-trick</th>
<th>Algorithm 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feret</td>
<td>7.5047s</td>
<td>6.3899s</td>
<td>4.3661s</td>
</tr>
<tr>
<td>ORL</td>
<td>0.9977s</td>
<td>0.7777s</td>
<td>0.3700s</td>
</tr>
<tr>
<td>Yale</td>
<td>0.1085s</td>
<td>0.06496s</td>
<td>0.03933s</td>
</tr>
</tbody>
</table>

4.6 Conclusions

In this chapter, we have introduced a method that allows for most LDA-based techniques that solve the SSS problem to be computed in $DN^2 + 2DNC$ time. The proposed method uses the same amount of space as other methods for implementing these techniques. The main computational complexity of our implementation is caused by matrix multiplications, as opposed to QR-decomposition, eigenanalysis or SVD. While not described above, fast matrix multiplication algorithms such as Strassen’s algorithm [26] can be applied to reduce the complexity further, producing an algorithm which can be run in $3.3DN^{1.8} + 2DNC$ time. Further reductions may be possible by taking advantage of any sparsity in the matrix.

Furthermore, this technique has been applied to MMC LDA and RLDA and showed experimentally that the proposed method is more efficient than the existing methods. We have also shown this theoretically, and shown the proposed algorithm uses the same space requirements as the existing methods. Finally, while we have demonstrated only on MMC LDA and RLDA, the methods shown here should be applicable to most LDA-based methods that attempt to solve the SSS problem.
Chapter 5

Fast Implementation of Null Linear Discriminant Analysis using Simple Matrix Operations

5.1 Introduction

Null Linear Discriminant Analysis (NLDA) is a popular method for dimensionality reduction when there are an insufficient number of samples to perform the original Linear Discriminant Analysis (LDA). As a result, more efficient methods to perform NLDA are required. Previous attempts to make an efficient method require calculations of eigenvectors, QR decomposition or Singular Value Decomposition (SVD) on large matrices. In this chapter, we introduce an algorithm which performs only matrix multiplications, pseudo-inverse calculations and a modified Gram-Schmidt decomposition. These are all simple algorithms that can be implemented by hand on most hardware platforms.

The first NLDA algorithm found the nullspace of $S_W$ directly, which requires both storing the $D \times D$ matrix and performing a number of operations on it of the order $O(D^3)$. Both these tasks can be infeasible,
even for modern computers. To overcome this, Huang [14] suggested removing the null space of $S_T$ first and presented an algorithm for NLDA which used $O(DN)$ memory usage and a number of FLOPs (floating point operations) equal to $16DN^2 + 4DNC + O(N^3)$. This was reduced further by Ye [32] by noting that the transformation matrix was orthonormal, allowing for an algorithm which executed $14DN^2 + 4DNC + O(N^3)$ FLOPs and requiring $O(DN)$ memory usage. More recently, Chu and Thye [5] proposed a method utilising QR decomposition, reducing the number of FLOPs required by the algorithm to $4DN^2 + 2DNC + O(N^3)$, while keeping the storage at $O(DN)$ memory usage. In this chapter, we introduce a method which reduced the complexity to $DN^2 + 2DNC + O(N^3)$ FLOPs with naive implementations of all methods and still using $O(DN)$ memory. With advanced methods such as Strassen’s algorithm [26], this is reduced to $3.3DN^{1.8} + 4DNC$.

The algorithm presented in this chapter is a specific application of the method introduced in Chapter 4. However, the application is different to the prior applications due to the nature of NLDA. We will demonstrate that the algorithm introduced in this paper achieves these same result using only matrix multiplication, matrix inversion and QR decomposition. The algorithm assumes naive implementations of all these procedures, which are competitive with more advanced procedures. This allows the algorithm to be performed on any form of hardware without having to include libraries or complex algorithms for tasks such as SVD, eigenvector analysis, and similar. For the purposes of testing, Matlab implementations of the existing functions were used.

This chapter is separated as follows: Section 5.2 will introduce a set of necessary and sufficient conditions for NLDA. Section 5.3 will introduce the theory used to implement the proposed method, and show that the proposed method is equivalent to NLDA. Experimental results will be included in Section 5.4. Then, conclusions will be drawn in Section 5.5.
5.2 Necessary and Sufficient Conditions for NLDA

NLDA is one of the most popular methods for handling the problem of SSS in the literature [6]. Several methods have been developed to improve upon the speed of this [5, 19, 24], however, the method of proving equivalence between their method and NLDA are based off faulty criteria. We will demonstrate in this section that the existing conditions presented are not sufficient. As a result, these methods, while they may be a valid approach, are not proven to be theoretically equivalent, even though they claim to be [24, 5, 19]. In this section, we aim to develop necessary and sufficient conditions for such claims. Determining a set of necessary and sufficient conditions is essential to ensuring new methods meet the requirements.

Several papers in the literature [24, 5, 19] introduce new methods for NLDA and attempt to prove the equivalence with the following criteria.

\[ S_B W \neq 0, \quad \text{and} \quad S_W W = 0. \]

Additionally, some papers attempt to prove that their proposed algorithms maximize the value of \( W^T S_B W \) under the constraints of NLDA by proving these conditions to be true [5, 19].

We demonstrate that these are insufficient with the following proof:

**Theorem 5.2.1.** The conditions \( S_B W \neq 0 \) and \( S_W W = 0 \) are insufficient for the purposes of NLDA.

**Proof.** Define the columnspace of the matrix that satisfies the constrained optimization problem as \( S \). Let \( w \) be any vector from \( S \). Let \( \hat{w} \) be a vector in the nullspace of \( S^T \). Therefore, \( S_B \hat{w} = 0 \) and \( S_W \hat{w} = 0 \). Yang and Yang proved that \( \hat{w} \notin S \) [28]. Let \( v = w + \hat{w} \). By definition of the vectorspace, we know that \( v \notin S \) [25]. However, it can be trivially shown that \( S_B v \neq 0 \) and \( S_W v = 0 \). Therefore, \( v \) meets the conditions but is not in the vectorspace that optimizes NLDA. \( \square \)
These results may be surprising, so we illustrate with an example. We take the smallest number of samples possible: 3 samples with one belonging to a first class and the other two belonging to a second class. These samples shall be represented by the vectors \(v_1 = [1000, 5, 4, 3]^T\), \(v_2 = [2, 1, 0, -1]^T\) and \(v_3 = [-2, -3, -4, -5]^T\). We assign class labels such that \(v_1 \in C_1\), \(v_2 \in C_2\) and \(v_3 \in C_2\). From inspection, we can see that the separating vector should be close to \([1, 0, 0, 0]^T\). After removing the mean, we find the matrices \(S_B\) and \(S_W\) are equal to the following.

\[
S_B = \begin{bmatrix}
22222.2 & 1333.3 & 1333.3 & 1333.3 \\
1333.3 & 8 & 8 & 8 \\
1333.3 & 8 & 8 & 8 \\
1333.3 & 8 & 8 & 8 \\
2.67 & 2.67 & 2.67 & 2.67 \\
2.67 & 2.67 & 2.67 & 2.67 \\
2.67 & 2.67 & 2.67 & 2.67 \\
2.67 & 2.67 & 2.67 & 2.67 \\
\end{bmatrix}
\]

(5.1)

\[
S_W = \begin{bmatrix}
2.67 & 2.67 & 2.67 & 2.67 \\
2.67 & 2.67 & 2.67 & 2.67 \\
2.67 & 2.67 & 2.67 & 2.67 \\
2.67 & 2.67 & 2.67 & 2.67 \\
\end{bmatrix}
\]

(5.2)

From this, we find the optimal vector for NLDA to be the vector \(w = [0.866, -0.289, -0.289, -0.289]^T\). Using this value, we find the values \(w^T S_B w = 1.65 \times 10^5\) and \(w^T S_W w = 0\). However, we also find that the vector \(q = [0, 0.8165, -0.4082, -0.4082]^T\) is in the nullspace of \(S_B\) and \(S_W\). If we construct a vector \(\hat{w} = w + 100\times q\), we get the vector \(\hat{w} = [0.0087, 0.8136, -0.4111, -0.4111]^T\). Using this value of \(\hat{w}\), we find that \(\hat{w}^T S_B \hat{w} = 16.5\) and \(\hat{w}^T S_W \hat{w} = 0\). Quick inspection shows us that the vector \(\hat{w}\) meets the conditions \(S_B w \neq 0\) and \(S_W w = 0\), and yet it does not optimize the criterion.

As another demonstration, we use the ORL database. We construct the optimal matrix from training data, and then find an equal number of vectors from the nullspace of \(S_T\) and form a matrix from this. Then these matrices are added together, weighted towards the vectors from the nullspace and the result is orthonormalized using QR decomposition.

Finally, the optimal matrix and the new matrix are used to classify the test data. The training data will consist of the first 5 samples of each class, while the test data will consist of the last 5 samples. When using the
optimal matrix, we find that the value of \( \text{trace}(W^T S_B W) \) is of the order of \( 10^8 \), while the value of \( \text{trace}(W^T S_W W) \) is of the order of \( 10^{-20} \). When using the optimal matrix is a transformation matrix, the accuracy on the test data is 91%. In comparison, when using the modified matrix, we find the value of \( \text{trace}(W^T S_B W) \) is of the order of \( 10^4 \), the value of \( \text{trace}(W^T S_W W) \) is of the order of \( 10^{-22} \), and the accuracy using the modified \( W \) is 26.5%. Even though the modified matrix meets the conditions for NLDA, it is clearly not the optimal matrix, and it underperforms in comparison to the optimal matrix.

The problem occurs because the condition \( S_B W \neq 0 \) does not result in a vector space [25]. Therefore, we attempt using a vectorspace instead. It is known that the matrix \( W \) is part of the rangespace of \( S_T \) [28], so we replace the condition that \( S_B W \neq 0 \) with the condition that \( W \) is a subspace of \( S_T \), giving us the following conditions:

\[
\exists \hat{w} \quad w = S_T \hat{w}, \quad \text{and} \quad (5.3) \\
S_W w = 0. \quad (5.4)
\]

Eq. (5.3) has been shown to be a necessary condition by Yang and Yang [28]. Eq. (5.4) is a requirement of the initial constrained optimization problem [3] and is, therefore, a necessary condition. As a result, it has already been proven that both of these conditions are necessary. We provide a proof that these conditions are sufficient below, under specific circumstances.

**Theorem 5.2.2.** Eq. (5.3) and Eq. (5.4) are sufficient conditions for null LDA, under the assumption that the data is linearly independent.

**Proof.** Let the vectorspace \( N \) be a vectorspace with a basis formed by the vectors that optimize the null LDA conditions. From the original methods for NLDA, it is clear that the rank of the vectorspace \( N \) is equal to \( C - 1 \).

Let \( M \) the vectorspace of the vectors that meet the conditions represented in Eq. (5.3) and Eq. (5.4). Since the nullspace of the matrix \( S_T \) is a subspace of the matrix \( S_W \), this vectorspace will have a rank equal to \( \text{rank}(S_T) - \text{rank}(S_W) \). Since \( \text{rank}(S_B) + \text{rank}(S_W) = \text{rank}(S_T) \), this means that the rank of the vectorspace containing all vectors that meet those
conditions is equal to the rank of $S_B$.

Since the vectors in the vectorspace $N$ are in the vectorspace $M$, and the ranks are equal, therefore, the vectorspaces are equal. Therefore, Eq. (5.3) and Eq. (5.4) are necessary and sufficient conditions for null LDA when the data is linearly independent. In the event that the data is not linearly independent, these requirements will be met by the

This ends the proof of this theorem.

Therefore, Eq. (5.3) and Eq. (5.4) are necessary and sufficient conditions for NLDA under the specific assumption that the input data is linearly independent. In other words, if all the vectors in the columnspace of the orthonormal matrix $W$ are in the nullspace of $S_W$ and the rangespace of $S_T$, and $W \in \mathbb{R}^{D \times (C-1)}$, then $\text{trace}(W^T S_B W)$ is maximized under the conditions that $W^T W = I$ and $W^T S_W W = 0$.

### 5.3 Derivation of the New Algorithm

We note that if $W = S_B^+ H_B$, then $W$ is claimed to be a solution to NLDA [31]. From this, we derive the following.

\[
W = S_B^+ H_B
= (H_T H_T^T)^+ H_B
= H_T^+ H_B^+ H_B
= [H_T (H_T^T H_T)^+ (H_T^T H_T^T)^+ H_T^T] H_B
= H_T (H_T^T H_T)^+ 2 H_T^T H_B.
\]

As stands, the resulting $W$ is not orthonormal, which is a requirement if the transformation matrix is to be used for classification [5]. To overcome this, we use a modification of the Gram-Schmidt Algorithm [12], where we are given a set of vectors $\{u_1, u_2, \cdots, u_C\}$ and a symmetric matrix $A$, and
we wish to find a set of vectors, \( \{ v_1, v_2, \cdots v_C \} \) such that \( v_i^T A v_j = \delta(i, j) \).

\[
f_k = u_k - \sum_{i=1}^{k-1} u_i^T A v_i,
\]

\[
v_k = \frac{f_k}{\sqrt{f_k^T A f_k}}.
\]

If we denote \( \hat{W} = (H_T^T H_T)^{+2} H_T^T H_B \), then it can be easily seen that the requirement that \( W^T W = I \) is equivalent to \( \hat{W}^T H_T^T H_T \hat{W} = I \). Therefore, applying the above equations to the columns of the matrix \( \hat{W} \) and using \( A = \hat{H}_T \) will result in the orthonormality requirements being met.

This allows us to create the following algorithm:

**NLDA Algorithm 1**

<table>
<thead>
<tr>
<th>Steps</th>
<th>FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Calculate ( \hat{H}_T = H_T^T H_T )</td>
<td>( DN^2 )</td>
</tr>
<tr>
<td>2) Calculate ( \hat{H}_B ) as described in Eq. (4.10)</td>
<td>( O(N^2) )</td>
</tr>
<tr>
<td>3) Calculate ( X = \hat{H}_T^+ )</td>
<td>( O(N^3) )</td>
</tr>
<tr>
<td>4) Calculate ( \hat{W} = X(X \hat{H}_B) )</td>
<td>( O(N^2C) )</td>
</tr>
<tr>
<td>5) Use the modified Gram-Schmidt equations above to orthonormalize ( \hat{W} )</td>
<td>( O(N^2C) )</td>
</tr>
<tr>
<td>6) Calculate ( W = H_T \hat{W} )</td>
<td>( 2DNC )</td>
</tr>
</tbody>
</table>

Table 5.1: NLDA Algorithm 1.

Note that the matrix multiplication in step 1 can be done in half the time of a regular matrix multiplication by taking advantage of the symmetry of the output.

To prove that this meets the conditions expressed earlier in this chapter, we note that proving (5.3) is trivial. Proof of (5.4) is supplied by Ye [31]. Therefore, this algorithm meets the necessary and sufficient conditions for equivalence to NLDA.

In the event that the data is not linearly independent, then in place of the orthonormalization step above, we instead focus on maximizing \( \text{trace}(W^T S_B W) \). There are two constraints in place here. The first is that
$W^TW = I$ and the second is that $W = H_T W^*$, where $V$ is a matrix with dimensionality $C \times R$, where $R$ is dependent on the rank of the matrices. A quick substitution allows us to eliminate the second condition and instead maximize $\text{trace}(V^T W^T H_T S_B H_T W V)$ under the condition that $V^T W^T H_T S_B H_T W V = I$. This can be reduced to maximizing the value of $\text{trace}(\hat{V}^T \hat{W}^T \hat{H}_B^T \hat{H}_B W V)$ under the condition that $V^T \hat{W}^T \hat{H}_T W V = I$, which is equivalent to finding the eigenvectors of the matrix $(\hat{W}^T \hat{H}_T W)^{-1} \hat{W}^T \hat{H}_B^T \hat{H}_B W$. This can be done in $O(N^3)$ time, and reconstructing $\hat{W}$ takes an additional $O(N^2 R)$ time. Since $R \leq C$, this means that these additional steps take $O(N^3)$ time, leaving the overall algorithm requiring a total number of FLOPs equal to $DN^2 + 2DNC$.

5.4 Experimental Results

In this section, we show experimental results using the alternate method.

We ran both the proposed method and QR-NLDA over three databases, the Feret database, the ORL database and the Yale database. The results are shown below. QR-NLDA has been shown to be equivalent in results to NLDA through experiments, and has been shown to be faster than the existing implementations both theoretically and experimentally [5, 19].

<table>
<thead>
<tr>
<th>Database</th>
<th>Error rate</th>
<th>NLDA Algorithm 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QR-NLDA</td>
<td></td>
</tr>
<tr>
<td>Feret</td>
<td>16%</td>
<td>16%</td>
</tr>
<tr>
<td>ORL</td>
<td>9%</td>
<td>9%</td>
</tr>
<tr>
<td>Yale</td>
<td>2.7%</td>
<td>2.7%</td>
</tr>
</tbody>
</table>

Table 5.2: Error rate of NLDA with varying algorithms.

Furthermore, to show that NLDA Algorithm 1 meets the conditions presented in Chapter 5, we include the values for $\text{trace}(W^T S_W W)$ and $\text{trace}[(H_T H_T W - W)^T (H_T H_T W - W)]$ in Table 5.4. For comparison, we also include these values using QR-NLDA in Table 5.4. These values will be zero if and only if $W$ meets the conditions presented in Chapter 5. All values are within reasonable tolerance and can be attributed to rounding.
Finally, we demonstrate the effects of varying the number of training samples and the number of features in regards to computational time, by plotting the comparisons in Figure 5.1 and Figure 5.2. In Figure 5.1, the number of features is held constant at 10304, the number of classes is held constant at 40, and the number of training samples is varied from 40 to 360, such that the training samples are evenly distributed between each class. In Figure 5.2, the number of training samples is held constant at 200, the number of classes is held constant at 40, and the number of features in each sample is varied from 500 to 10000. In each case, the data is a subset of the ORL database.

<table>
<thead>
<tr>
<th>Database</th>
<th>(\text{trace}(W^T S_W W))</th>
<th>(\text{trace}[(H_T H_T^T W - W)^T (H_T H_T^T W - W)])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feret</td>
<td>(1.09 \times 10^{-22})</td>
<td>(3.42 \times 10^{-26})</td>
</tr>
<tr>
<td>ORL</td>
<td>(2.97 \times 10^{-23})</td>
<td>(1.93 \times 10^{-27})</td>
</tr>
<tr>
<td>Yale</td>
<td>(1.00 \times 10^{-23})</td>
<td>(4.31 \times 10^{-28})</td>
</tr>
</tbody>
</table>

Table 5.3: *Computational time of NLDA with varying algorithms.*

<table>
<thead>
<tr>
<th>Database</th>
<th>(\text{trace}(W^T S_W W))</th>
<th>(\text{trace}[(H_T H_T^T W - W)^T (H_T H_T^T W - W)])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feret</td>
<td>(6.92 \times 10^{-15})</td>
<td>(3.46 \times 10^{-20})</td>
</tr>
<tr>
<td>ORL</td>
<td>(1.48 \times 10^{-17})</td>
<td>(1.65 \times 10^{-27})</td>
</tr>
<tr>
<td>Yale</td>
<td>(3.39 \times 10^{-18})</td>
<td>(3.60 \times 10^{-28})</td>
</tr>
</tbody>
</table>

Table 5.5: *Accuracy of NLDA Algorithm 1 with respect to the necessary and sufficient conditions presented in Chapter 5.*
Figure 5.1: *The computational time required by the two NLDA algorithms for varying amounts of training samples.*

Figure 5.2: *The computational time required by the two NLDA algorithms for varying amounts of features.*

### 5.5 Conclusion

In this section, we have shown necessary and sufficient conditions for a set of vectors to be optimal according to the criteria for NLDA. Additionally, we have applied the methods introduced in Chapter 4 to the Null Linear Discriminant Analysis method. This allows for an efficient implementation of NLDA that can be executed on any form of hardware, due to the simplicity of the methods used. Experimental results have been included to show that this method is faster than the current algorithm being used for NLDA, achieves the same accuracy, and meets the requirements presented earlier in this chapter.
Chapter 6

Conclusion

In this thesis, we have examined the problem of efficient methods of dimensionality reduction. Dimensionality reduction algorithms are a necessity due to several reasons including

- supervised learning algorithms work better (produce more accurate classifiers) when the input offers a large number of samples relative to the dimensionality of the samples, and
- data sets with smaller dimensionality demand less computational cost.

To set the context, in Chapter 2, we provided an overview of some well-established techniques for pattern classification and dimensionality reduction. This overview allows us to observe the requirements that dimensionality reduction approaches must meet in order to support the learning of classifiers. An overview of the unsupervised and supervised dimensionality reduction techniques allows us to identify the current state of knowledge.

In Chapter 3, we compared the methods presented in Chapter 2 under a variety of situations to determine which methods were best under normal conditions and how robust they were to various modifications. This included changes to the number of classes, changes to the amount of training data, addition of noise to the training data, and addition of noise to the test data. It was shown that while NLDA produced accurate
classifiers under most circumstances, other algorithms were more accurate when the conditions were not ideal.

In Chapter 4, we introduced a more efficient version of an existing dimensionality reduction algorithm and demonstrated that the results of these algorithms are equivalent to existing algorithms (that is, there is no penalty with respect to accuracy of the classifiers). The benefit of the new proposed algorithms is that the computational time required is less. These results were confirmed both theoretically and experimentally. The experimental confirmation also illustrated the equivalence of the solution at the dimensionality reduction stage because we showed that the vectors that are obtained with the new methods or other methods constitute solutions with equal values on the optimisation criterion.

In Chapter 5, we demonstrated that what the community believes are sufficient conditions for proving that an algorithm supplies equivalent results to NLDA are in fact insufficient conditions. This does not establish that any algorithm is incorrect, just that the evidence that they are correct is not accurate. We then proceeded to present a set of new conditions. We prove these conditions are necessary and sufficient conditions (in the sense of a mathematical if-the-and-only-if theorem) to establish when an algorithm produces equivalent results for NLDA. One can now properly show with these conditions the claims regarding the algorithms in the literature. It is pleasing to see that, to the best of our knowledge and literature review, published algorithms meet these conditions.

Finally, in Chapter 6, we extended the algorithm introduced in Chapter 3 to produce an efficient implementation of NLDA and showed that it was equivalent to NLDA using the results of Chapter 5. As in Chapter 4, the results were proven theoretically (that is, with a mathematical proof) and confirmed experimentally (over several datasets) to be equivalent to NLDA. The experimental evaluation also allows to verify and validate that the promised improvement in the theoretical computational time is reflected in practical settings. The datasets used vary in aspects like the number of features or on the number of training samples. Furthermore, all algorithms and implementations use operations with well-known implementations (no need for sophisticated and elaborate implementation of matrix multiplication, for example) and therefore can be coded without the need
for libraries that offer such advanced implementations, improvements could be even more dramatic if those sophisticated implementations were used.
Bibliography


