ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

LINEAR ALGEBRAIC METHODS FOR MACHINE LEARNING

M.Sc. THESIS

Elif ALTUNOK

Department of Mathematical Engineering

Mathematical Engineering Programme

DECEMBER 2019

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

LINEAR ALGEBRAIC METHODS FOR MACHINE LEARNING

M.Sc. THESIS

Elif ALTUNOK
(509161286)

Department of Mathematical Engineering

Mathematical Engineering Programme

Thesis Advisor: Prof. Dr. Atabey KAYGUN

DECEMBER 2019

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

MAKİNE ÖĞRENMESİ İÇİN DOĞRUSAL CEBİRSEL YÖNTEMLER

YÜKSEK LİSANS TEZİ

Elif ALTUNOK
(509161286)

Matematik Mühendisliği Anabilim Dalı

Matematik Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Atabey KAYGUN

ARALIK 2019

Elif ALTUNOK, a M.Sc. student of ITU Graduate School of Science Engineering and Technology 509161286 successfully defended the thesis entitled “LINEAR ALGEBRAIC METHODS FOR MACHINE LEARNING”, which she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Atabey KAYGUN** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Serkan SÜTLÜ** Işık University	.....
	Asst. Prof. Dr. Gül İNAN Istanbul Technical University	.....
		.....

Date of Submission : **15 November 2019**
Date of Defense : **11 December 2019**

To my family,

FOREWORD

I would like to express my deepest sincere gratitude to my thesis advisor Prof. Dr. Atabey Kaygun who gave me support and valuable guidance in the process of writing my thesis. I am also grateful to him for spending his valuable time in giving us machine learning seminars on the weekends during the year. His experiences that he shared with us, motivated and paved our way.

I would like to sincerely thank Prof. Dr. Oğuzhan Külekci for giving me the motivation and the opportunity to do research in his laboratory.

Last but not least, my gratitude also goes to my father, mother, sisters, and friends for their love and their pray in this process. Thank you all who helped or contributed to finish my Master's program.

December 2019 Elif ALTUNOK

TABLE OF CONTENTS

Page

ABBREVIATIONS

ANOVA : Analysis of Variance
ARM : Association Rule Mining
CCA : Canonical Correlation Analysis
CV : Cross Validation
EVD : Eigenvalue Decomposition
EOF : Empirical Orthogonal Functions
GCV : Generalized Cross-Validation
ICA : Independent Component Analysis
KKT : Karush-Kuhn-Tucker
KPCA : Kernel Principal Component Analysis
LDA : Linear Discriminant Analysis
LR : Logistic Regression
LOOCV : Leave-One-Out-Cross-Validation
MCA : Multiple Correspondence Analysis
NDA : Normal Discriminant Analysis
PCA : Principal Component Analysis
POD : Proper Orthogonal Decomposition
QDA : Quadratic Discriminant Analysis
SVD : Singular Value Decomposition
SVM : Support Vector Machine
TSVM : Transductive Support Vector Machines

LIST OF TABLES

Page

LIST OF FIGURES

Page

LINEAR ALGEBRAIC METHODS FOR MACHINE LEARNING

SUMMARY

In this thesis, we investigate mathematical foundations of commonly used linear algebraic methods in machine learning: Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Support Vector Machines (SVM). We also demonstrate their performances on the Olivetti Faces data set [4], Fashion-MNIST data set [5] and MADELON [6] data set. We present a comparative study on challenging clustering and classification problems on these data sets using the linear algebraic methods we listed above.

One of the contributions of this thesis is to present a comprehensive and 2-way statistical comparison of different models on different data sets. In one direction, we used different types of models on the same data set for comparison purposes, while in the other direction we compared the performance of a chosen method on different data sets. We also evaluated the performances of the models by varying the chosen parameters on each data set.

Our result indicate that varying number of dimensions did not affect the accuracy of LDA, and LDA showed the same performance across all data sets, even in 2 dimensions. However, models performances have changed from data set to data set. In our experiments we obtained that the Olivetti face data set yielded an accuracy score of 97%, while on the Fashion-MNIST data set the score reduced to 82%. The MADELON was on the other hand yielded the worst performance with an accuracy of 55%.

In SVM, we used four different kernel functions: linear, radial basis, polynomial and sigmoid. The results we obtained showed that the type of kernel function chosen has considerable effect on the performance of the model. For example, the linear kernel gave excellent recognition accuracy of 94%, while the polynomial kernel had a limited accuracy of 58% on the Olivetti faces data set.

When the methods we investigate in this thesis are combined, in some cases, they produce far better results when they were used on their own. For example, on the Olivetti face set, the accuracy of the SVM with linear kernel after we reduced the data set to 30 dimensions using PCA had an accuracy score of 97%, approximately the same as the 2-dimensional LDA model. However, PCA-SVM combination yielded 15 times faster results. It should be noted that the LDA method is also 15 times better in terms of reducing the file size.

We achieved with an accuracy of 82% in 6.76 seconds using a 2-dimensional LDA model, and 89% accuracy in approximately 31 minutes using the SVM with RBF kernel on the Fashion-MNIST data set. Thereafter, we applied the SVM with RBF kernel on reduced the data to 15 dimensions by PCA accelerated the process 29 times

by obtaining an accuracy score of 85%. However, the 2-dimensional LDA method is still about 10 times faster and saved about 7 times more space.

We also found that with an accuracy of 58% in 3.29 seconds using SVM with RBF kernel on the MADELON data set. Afterward, we applied SVM with RBF kernel on the reduced data set by PCA from 500 to 5 dimensions. Results showed that by using the combined method PCA with SVM, the data points were classified with an accuracy of 81% in 0.08 seconds that was approximately 40 times faster than SVM alone. Moreover, we have achieved a reduction in file size by 99%.

MAKİNE ÖĞRENMESİ İÇİN DOĞRUSAL CEBİRSEL YÖNTEMLER

ÖZET

Bu tezde, makine öğrenmesi algoritmalarında sıklıkla kullanılan doğrusal cebirsel yöntemlerin matematiksel temellerini analiz ettik ve Temel Bileşen Analizi (PCA), ikinci olarak Lineer Ayrimcılık Analizi (LDA) ve son olarak Destek Vektor Makineleri (SVM) yöntemlerini inceledik. Sonra da bu yöntemlerin performanslarını farklı veriler üzerinde test ettik.

Makine öğrenmesi, belirli bir görevi tamamlamak için göreve ait geçmiş deneyimlerin veya örnek verilerin kalıplarını keşfederek modeller oluşturan ve bu modellerle tahminler yapıp karar almaya yarayan bir süreç olarak özetlenebilir. Makine öğrenimi, e-posta filtrelemeden yüz tanımaya kadar birçok farklı alandaki sorunları çözmek için yaygın olarak kullanılmaktadır. Hatta makine öğrenimi algoritmaları veri hakkında bir bilginin olmadığı ya da az bilginin olduğu durumlarda da kullanılabilir. Ancak uygulayıcının eldeki yöntemden en iyi performansı alabilmesi için bir takım karar basamaklarından geçmesi gerekmektedir. Burada uygulayıcının veri kümesini hangi ön işlem basamaklarından geçirmesi, problem için hangi yöntemi seçmesi ve hatta seçtiği yöntemle modeli oluştururken hangi parametreleri kullanması gerektiğine karar vermesi gerekir. Bu karar basamakları uygulanan modelden en iyi performansı alabilmek için bir temel olarak ele alınabilir.

Literatürde halihazırda bir çok farklı yöntem mevcut olması sebebiyle yöntem seçim işlemi zor olabilir. Makine öğrenmesi süreçlerinde yöntem seçildikten sonra oluşturulacak modelden en iyi performansın alınabilmesi için model parametrelerinin doğru tayin edilmesi büyük önem taşımaktadır. Bu sebeble modelin performansının seçilen parametre değerlerine göre nasıl değiştiğinin istatistiksel olarak saptanması gereklidir. Burada uygulayıcının önemli gorevlerinden birisi modele en uygun parametrelerin seçimi için model sonuçlarını istatistiksel olarak analiz etmesidir. Bu istatistiksel bakış açısı, iyi bir öngörü sağlayarak modelin en iyi biçimini kullanmamızın yolunu bize daha hızlı açacaktır. Bu nedenle değişik veri kümesi üzerinde uygulanan değişik makine öğrenimi yöntemleri ile elde edilen modellerin performanslarının karşılaştırıldığı bir çalışmanın yararlı olacağını düşünüyoruz.

Temel Bileşen Analizi, veri kümelerindeki örüntülerin tespitinde, öznitelik çıkarmada, ve veri kümesinin boyutunu küçültmede yaygın olarak kullanılan denetimsiz bir öğrenme yöntemidir. Yöntem, veri kümesindeki bilgilerin çoğunu koruyarak çok boyutlu verileri daha düşük boyutlara indirmek için kullanılır. Bu işlemi veri kümesindeki en yüksek değişkenliğe sahip olan bileşenleri veri kümesinde tutup, az değişkenliğe sahip olan bileşenleri çıkartarak yapar. Bunu yapmak için veri kümesine ait kovaryans matrisi ve bu matrisin özdeğerlerini bularak ilerler. Bu özdeğerler bize bu özdeğerlere karşılık gelen özvektörler boyunca uzanan veri noktalarında ne kadar varyans olduğunu söyler. Bu şekilde özdeğerler sıralandığında karşılık gelen

özvektörlerdeki veri bilgisinin de önem sırası belirlenmiş olur. Buradan hareketle en büyük özdeğere karşılık gelen özvektör veri kümesinin temel bileşeni olacaktır. En küçük özdeğerler boyunca bulunan bileşenler boyut sayısını azaltma amacıyla veri kümesinden çıkartılabilir. Bu yöntem örüntü tanımadan yüz tanımaya kadar farklı uygulamalarda sıklıkla kullanılmaktadır. Bu tezde bu yöntemle veri kümeleri farklı boyutlara indirgenip, ardından destek vektör makineleri yöntemiyle sınıflandırılmıştır.

Lineer Ayrımcılık Analizi (LDA), veri kümesinin boyutunu küçültmede, öznitelik çıkarımında, örüntü sınıflandırmada ve çok sınıflı sınıflandırma görevleri için sıklıkla kullanılan denetimli bir öğrenme yöntemidir. Yöntem, sınıf ayrılabilirliğini maksimuma çıkartmayı amaçlayacak şekilde bir öznitelik altuzayı bulmayı amaçlar. LDA veri kümesindeki gözlemlerin sınıflara ait olma olasılıklarını belirlemek için Bayes Teoremini kullanır ve ayrım fonksiyonu bulur. Bulduğu ayrım fonksiyonunda gözlemin aldığı değeri her bir sınıf için hesaplayarak bu değerlerin en büyüğüne göre noktanın sınıf atamasını yapar. Bu tezde LDA yöntemi ile veri kümeleri farklı boyutlara indirgenip sınıflandırılmış ve farklı parametrelerin verdiği sonuçlar karsılaştırılmıştır.

Destek vektör makineleri (SVM), sınıflandırma ve regresyon analizi için kullanılan denetimli bir öğrenme yöntemidir. Bu yöntem, yüz tanımadan zaman serisi tahminlerine kadar birçok farklı uygulamada kullanılmaktadır. SVM'in amacı veri setimizi sınıflandırma için ayrık altkümelere bölen bir dizi optimal hiper düzlemi bulmaktır. Buradaki hiper düzlem farklı sınıfların veri noktaları arasındaki mesafeyi maksimize edecek şekilde seçilmektedir. Yöntemde saptanan destek vektörler arasındaki mesafe marjin olarak adlandırılır. Her zaman doğrusal olarak iki sınıfı birbirinden ayırmak mümkün olmayabilir. Bu yöntem doğrusal olarak çalışmasına rağmen doğrusal olarak ayrılayaman veri kümelerinde de çekirdek hilesini kullanarak başarılı bir şekilde sınıflandırma yapabilir. Bu yaparken geri planda doğrusal olmayan çekirdekler kullanarak veri kümesini bulunduğu boyuttan daha üst boyutlara çıkartıp yeni uzaya özgü lineer bir çekirdek kullanarak yapar. Yaygın olarak kullanılan ve bu tezde de kullandığımız çekirdek fonksiyonları, doğrusal, radial tabanlı, polinomiyal ve sigmoid olarak sıralanabilir. Bu tezde bu fonksiyonların hepsi için modellerin performansları mukayese edilmiştir.

Bu tez çalışmasında, yüz tanıma problemi için Olivetti yüz kümesini, çoklu sınıflandırma problemi için Fashion-MNIST veri kümesini ve ikili sınıflandırma problemi için MADELON veri kümesini kullandık.

Olivetti yüz veri kümesi 40 farklı insanın 10'ar adet farklı yüz pozlarını içeren toplam 400 adet görüntüden oluşmaktadır. Görüntüler ışıklandırma, yüz ifadeleri, gözük detayı gibi farklı özelliklerle birlikte farklı zamanlarda, aynı homojen arka planda, pozisyonlarda ve labaratuar ortamında çekilmiştir. Buradaki amacımız yüz tanıma yaparak, görüntüleri 40 farklı sınıftan birisine atayarak kime ait olduğunu tespit etmektir.

Fashion-MNIST veri kümesi 70,000 adet hazır giyim parçası görüntülerinden oluşmaktadır. Veri kümesinde 10 çeşit giyim ürününün görüntüleri bulunmaktadır. Buradaki amacımız bu görüntülerin hangi giyim ürünü olduğunu belirlemektir.

Son olarak, yüksek derecede doğrusal olmayan, çok değişkenli ve yapay olan MADELON veri kümesini kullandık. Veri kümesi +1 ya da -1 sınıfına karşılık gelen 2200 adet pozitif ve 2200 adet negatif veri noktalarından oluşmaktadır. Bu veri

noktaları 5 boyutlu hiperküpün 32 köşesinde gruplanmıştır. Buradaki amacımız veri noktalarına ikili sınıflandırma yapmaktır.

Bu tezde bu üç farklı veri kümesi üzerinde bahsettiğimiz metodları kullanarak toplam 93 farklı model geliştirdik. Tez çalışması boyunca her bir veri kümesi için yukarıda bahsettiğimiz yöntemleri kullanarak farklı modeller oluşturduk. Bu sayede her bir metodun performansını farklı problemler için değişen veri kümeleri bazında gözlemleyebildik. Aynı zamanda, her veri kümesi için modeldeki parametrelerin farklı değerlerinin modelin performansını nasıl etkilediğini de sergiledik.

Modellerimizin sonuçlarını özetleyecek olursak, öncelikle çok boyutlu olan veri kümelerini LDA yöntemiyle farklı boyutlara indirgeyerek sınıflandırma işlemlerini yaptık. Daha sonra her veri kümesi için SVM yöntemini farklı çekirdek fonksiyonlarıyla kullanarak modellerimizi oluşturduk. Son yaklaşımımız ise veri kümelerimize PCA uygulayarak onları farklı boyutlara indirgeyip, sonrasında sınıflandırma işlemi için yine SVM'i farklı çekirdek fonksiyonlarıyla birlikte kullanmak oldu.

Yaptığımız deneylerle metodların farklı veri kümelerinde farklı performans gösterdiğini gözlemledik. Değişen boyut sayısı LDA'in doğruluk oranını etkilemedi: LDA 2 boyutta dahi aynı performansını tüm veri kümelerinde gösterdi. Örneğin, 2 boyutlu LDA Olivetti yüz veri kümesinde %97, benzer dinamikte olan Fashion-MNIST veri kümesinde %82 doğruluk oranı verirken MADELON veri kümesinde %55 olan doğruluk oranıyla diğer veri kümelerinin oldukça gerisinde kaldı.

SVM yöntemiyle geliştirdiğimiz modellerde ise değişen çekirdek fonksiyonlarının performansı etkilediğini saptadık. Örneğin, Olivetti yüz veri kümesinde doğrusal çekirdek fonksiyonuyla SVM %94 doğruluk oranıyla 1,27 saniyede sonuç verirken, polinomiyal çekirdek fonksiyonu %58 doğruluk oranıyla 1,35 saniyede oldukça kötü bir sonuç sergiledi.

Daha sonra, yine SVM yönteminde farklı çekirdek fonksiyonlarını kullanarak bu kez PCA ile farklı boyutlara indirgediğimiz veri kümeleri üzerinde sınıflandırma işlemini gerçekleştirdik. Bu sayede, hem PCA yönteminin boyut indirgemede farklı sayıdaki bileşenlerle performansını sonrasındaki sınıflandırma işlemi için gözlemleme imkanına sahip olurken hem de SVM yönteminin performansını desteklemeyi amaçladık. Gerçekten de SVM yöntemini LDA ile kıyaslandığımızda çok yavaş bir yöntem iken, PCA uygulanmış veri kümesi üzerinde SVM uyguladığımızda oldukça hızlandı, hatta geliştirdiğimiz onlarca modelde yer yer bu kombinasyonun LDA yönteminden daha hızlı çalıştığını gözlemledik. Üstelik SVM'i tek başına kullandığımız modellerdeki doğruluk oranlarından daha iyi sonuçlar elde edebildik. PCA ile verimizin boyutunu indirgediğimiz için dosya boyutundan da tasarruf edebilme imkanımız oldu. Örneğin, MADELON veri kümesinde SVM'i tek başına kullanarak polinomiyal çekirdek fonksiyonu ile en iyi sonucu %61 doğruluk oranı ve 3,08 saniye çalışma süresiyle almıştık. PCA ile veri kümesini 500 boyuttan 5 boyuta indirgeyip yine RBF çekirdek fonksiyonuyla SVM uyguladığımızda %81 doğruluk oranı ve 0,08 saniye çalışma süresiyle yaklaşık 38 kat daha hızlı sonuç alabildik. Üstelik dosya boyutunda %99 oranında bir küçülme sağladık. PCA uygulanmış veri kümeleri üzerinde SVM uyguladığımız modeller ile LDA yöntemini mukayese ettiğimizde aldığımız sonuçlar yine veri kümelerine göre değişkenlik gösterdi.

MADELON veri kümesinde LDA yönteminin iyi çalışmadığından bahsetmiştik. Olivetti yüz kümesinde en iyi sonucu aldığımız PCA-SVM kombinasyonu ile en

iyi sonuç aldığımız LDA modelini karşılaştıracak olursak 2 boyutlu LDA ile 30 boyutlu PCA ve doğrusal çekirdekli SVM modellerinin doğruluk oranları %97 ile aynı çıktı. Ancak PCA-SVM kombinasyonu 15 kat daha hızlı sonuç verdi. Burada LDA yönteminin de dosya boyutunu küçültme açısından 15 kat daha iyi olduğu unutulmamalıdır.

Fashion-MNIST veri kümesindeki en iyi LDA ve en iyi PCA-SVM kombinasyonu ile sonuç aldığımız modelleri karşılaştıracak olursak, 2 boyutlu LDA'de %82 doğruluk oranını 6,76 saniyede alırken, PCA ile veri boyutunu 784 boyuttan 15 boyuta indirip RBF çekirdek fonksiyonlu SVM modeliyle %85 doğruluk oranını 64,70 saniyede alabildik. Her ne kadar tek başına RBF çekirdek fonksiyonu kullanarak %89 doğruluk oranını 31 dakikada elde ettiğimiz veriye 15 boyutlu PCA uygulayarak işlemi 29 kat hızlandırıp %85 doğruluk oranı sağlayabilsek de, 2 boyutlu LDA yöntemi hala yaklaşık 10 kat hızlıydı ve yaklaşık 7 kat daha fazla alan tasarrufu sağlıyordu.

Oluşturduğumuz 93 farklı modellerden hareketle çıkarttığımız ortak sonuç, yöntemlerin başarısının veri kümelerine bağlı olarak değişmesi ve yöntemleri kullanarak oluşturduğumuz modellerin başarısının da kendi içerisinde seçilen parametre değerlerine bağlı olarak değişmesi oldu. Ayrıca modelden beklentimizin ne olduğuna göre seçim yapmanın da önemini gözlemledik; çünkü farklı yöntemler kullanarak hemen hemen aynı doğruluk oranlarını yakaladığımız modellerden birisi daha hızlıyken bir diğeri verimizin boyutunu daha fazla düşürerek bize daha fazla alan kazandırabildi.

1. INTRODUCTION

1.1 Overview

Machine learning is a collection of practices with a particular goal of creating a computational models from past experiences in the form of sample data to discover patterns, or a solution for a given task. The fitted model may then be used to make predictions for future observations. Machine learning has been widely used both in academia and industry to solve a wide range problems in many different areas from e-mail filtering to face recognition. It can even be used when we have little to no idea what the internal structure of the data set at hand is. However, to get the best results, the practitioner would still need to go through certain decision-making steps such as deciding on how to pre-process the data set, choosing a suitable model, setting the correct parameters as a starting point for building models with acceptable performance and cost characteristics. As many different models are currently available in the literature, their selection can be challenging, and for this reason, experimenting with different models on the same data set is a quite useful practice.

It is important to note that choosing parameters correctly leads to better performance for the chosen model. In order to set the right parameters, we need to see any changes in the performance of the model with each selection of the parameters. Hence, an important task for any practitioner is analyzing the performance of a model statistically to extract the most suitable parameters effectively that generates the most suitable model for the data set at hand. In order to be able to do these in the most effective way, it is necessary to investigate mathematical and statistical foundations of machine learning algorithms. Having a firm grasp of the theoretical foundations will give us a better control in which methods to use and how to use them for the problem at hand.

1.2 Literature Review

Principal component analysis (PCA) is an unsupervised learning method used in dimensional reduction, and pattern recognition. It has been used in many different applications such as face recognition [7–9], human-made object recognition [10], hand writing recognition [11], mobile robotics [12], and industrial robotics [13]. PCA also can also be combined with many different machine learning methods such as factor analysis, canonical correlation analysis (CCA), correspondence analysis, K-means clustering, non-negative matrix factorization.

PCA is first formulated by Pearson [14] in 1901 to find "lines and planes of closest fit to systems of points in space". Fisher and MacKenzie [15] observed that PCA is a successful approach to modeling of response data. The version of the PCA method commonly used was developed by Hotelling [16] in the 1930s. It found use in different scientific fields in solving a wide range of problems such as singular value decomposition [17, 18] and eigenvalue decomposition (EVD) [19] in linear algebra, Karhunen-Loeve expansion [20] in electrical engineering, Eckart–Young theorem (Harman, 1960) [21], or empirical orthogonal functions (EOF) [22] in meteorological science, proper orthogonal decomposition (POD) [23] in mechanical engineering. One can find a thorough statistical analysis of the subject in Gnanadesikan [24], Mardia et al. [25], Johnson and Wichern [26] and Joliffe [27].

Linear discriminant analysis (LDA) is a supervised learning method that most widely used for statistical pattern classification, dimensional reduction, feature extraction and multi-class classification. The LDA method also known as normal discriminant analysis (NDA) or discriminant function analysis, a generalization of Fisher's linear discriminant. LDA is an alternative method to PCA that maximizes class separability. LDA in many ways forms is related to other methods such as analysis of variance (ANOVA), regression analysis, logistic and probit regression, factor analysis and principal component analysis. LDA has been used in many different applications such as face recognition [28], image retrieval [29], micro array data classification [30], mobile robotics [31], bankruptcy prediction [32], earth science [33] and biomedical

studies [34]. LDA was first developed by Fisher [35] in 1936. In this thesis, our main reference is [1] for the use of linear discriminant analysis in machine learning.

Support vector machine (SVM) is a supervised learning method used for classification and regression analysis. SVM has been used in many different applications such as recognition [36, 37], shallow semantic parsing [38], image segmentation [39], medical decision support [40], and time series prediction [41]. SVM was first introduced by Vapnik and Chervonenkis [42] in 1963, and then by Boser, Guyon and Vapnik extended SVM to non-linear classifiers by improving the kernel trick it in 1992 [43]. Cortes and Vapnik further improved on SVM to its present form by adding soft margin [44]. Veropoulos [45] have been successfully applied SVM to biological data processing for medical diagnosis. Tefas [46] has shown that an SVM model using a kernel based on Fisher Linear Discriminant performs better than a standard linear SVM for a face recognition. We use [3] by Bishop for our analyses of SVM presented in this thesis.

1.3 Aim of Thesis

The objective of this thesis is to investigate the mathematical foundations of three main linear algebraic methods used in machine learning: PCA, LDA and SVM. We then use these methods to create models on different data sets, and then to compare their performances. We observed that the performances of the models we obtain vary depending on the data set, and did a 2-way comparison of the performances: across data sets and across different methods.

1.4 Thesis Structure

This thesis is organized in seven chapters. In Chapter 1 we present an overview of the thesis, do our literature survey, present the aim of our thesis, and the thesis structure. Chapter 2 provides an overview of machine learning and introduce supervised, unsupervised and semi-supervised learning, and cross-validation that will be used in the thesis.

In Chapter 3 we cover the basic machine learning algorithms logistic regression and Bayes classifier that we used this thesis. In Chapter 4 cover the theory of linear algebraic methods PCA, LDA and SVM. In Chapter 5 we presented the data sets

and methods we use in this thesis. In Chapter 6, we presented and discussed the experimental results. Finally, in Chapter 7, we presented conclusions of our research and the suggestions for further study.

2. MACHINE LEARNING

2.1 What Is Machine Learning?

In this Section we primarily use [47, Chapter 1.1] and also use [2, Chapter 1] and [3, Chapter 1].

Machine learning can be summarized as completing a task, or making predictions or decisions based on available data using computational methods and algorithms. Machine Learning methods are particularly useful when we are faced with complex tasks that we cannot solve directly by writing computer programs based on preset rules. Machine learning effectively uses mathematics and statistics to build an algorithmic models to solve a problem. We use such models to implement solutions written as computer programs. In such cases where there is no way to create a direct model, we need sample data or experience for the computer to learn.

There are two main reasons why we might prefer using machine learning. The first is the absence of human expertise, or the inability of people to explain their expertise. For instance, we can easily recognize spoken speech and convert the acoustic speech signal into ASCII text. But explaining how we do cannot be done easily. People of different ages, genders, and geographic regions pronounce the same words differently. The machine learning approach in such a task is to collect a large sample utterances from different people and learn to match words with them.

The other main reason why machine learning would be preferable in solving a problem is when the problem changes over time, or when the problem depends on a particular environment. Instead of writing different programs for each change of circumstances for dynamic problems, we can create a general purpose system that can adapt to all changes using machine learning. For example, let us consider the problem of routing packets managed over a computer network. In this example, the optimal path for a packet sent over the network is constantly changing as the network traffic on the path

changes. Therefore, the model needs to update itself continuously and adapt to the best path.

An algorithm is a sequence of instructions that should be carried out to transform an input to an output. However, in some tasks an explicit algorithm may not be given. For instance, we do not use an algorithm to distinguish spam e-mails from legitimate e-mails. In this example, the input is an e-mail document. The output we need is a label: *spam* or *non-spam*. In fact, we do not know how to transform the input into output. In addition, spam changes over time, or may vary from person to person. In the absence of such information, we make up for this lack of explicit algorithm with data. First, we compile thousands of sample messages, some of them we know are spam and then we *learn* from this data what constitutes as spam. The computer then can extract the most suitable parameters for a statistical algorithm that solves this task automatically. Since we cannot know which person does not want which e-mail, we cannot write code directly on a preset collection of rules that sorts e-mails as spam and not-spam. What we do here is to collect the data and hope that the computer will do it correctly. While we do not know the explicit details of the process underlying the detecting spam emails, we also know that this process is not entirely random. Machine learning focuses on detecting certain patterns or regularities in the data to understand the process, and then comes up with the most suitable parameters for a statistical sorting process. Although we do not know these parameters beforehand, and the result may not be 100% accurate, we expect to create *a good and useful approximation*. This approach can take some of the data into account, even if it does not explain everything. The patterns may help to understand the process and make predictions.

2.2 Supervised Learning

In supervised learning we build a statistical model for estimating, or predicting the response variables from each observations using a model based on examples where a response is given to each example data point. In this setup, we have input variables x_i where $i = 1, \ldots, n$ for each data point of the predictor measurement along with their corresponding response measurement y_i. *A predictor variable* is also called as an independent variable, or as an input variable. Predictor variables explain changes for a particular response in a data set. *Response variable* refers to the variable that

measured using a predictor variable. The goal of the supervised learning is to fit a model for approximate the mapping function to predict the response variable y for future points or to find the relationship between the response and the predictors.

There are many different methods in supervised learning, the most common are:

- Linear Regression: Linear regression finds the best fitting linear equation in order to predict a quantitative response variable. If there is only one predictor and one response variable, then the method is called *univariate* or *simple linear regression*, and when the number of predictors is greater than one, it is called *multivariate linear analysis*. For details see [1, Chapter 3].

- Logistic Regression: Logistic regression is a statistical method that models a binary dependent variable using a logistic function. Logistic regression does not directly model the response variable Y, this method models the probability that Y belongs to a particular category. See Section 3.1.

- Linear Discriminant Analysis (LDA): LDA is a dimensionality reduction method for classification problems. LDA finds linear combinations of features to explain the data. This method works by modeling the difference between data classes. See Section 4.2.

- Support Vector Machines (SVM): SVM is a machine learning model that is used for both classification and regression analysis. SVM finds an optimal hyperplane that separates classes. See Section 4.3.

- K-Nearest Neighbors: This is a classification method where for a given a positive integer k, we find (in the best case) k-cluster centroids from the training data set, and then we determine the class of a new given data point depending on the closest centroid. For details see [1, Chapter 2.2.3].

- Naive Bayes: The naive Bayes classifier is based on the Bayes' Theorem and the maximum posterior estimation. This classifier is the simplest instance of a probabilistic induction. The method assumes that the features are independent in a data set to simplify learning. So that, this learning method is fast and easy to implement. For details see [48].

- Decision Trees: The approach of this method is based on multi-stage decision making. In order to build decision trees, this method uses a series of splitting rules. For details see [1, Chapter 8.1].

- Random Forests: This method is an ensemble learning that is used in both classification and regression by building a large number of individual decision trees. For details see [1, Chapter 8.2.2].

2.3 Unsupervised Learning

Contrary to supervised learning, in unsupervised learning there is no associated response variable for the observations. Unsupervised learning is more challenging than supervised learning since we only have the input data x_i without a prescribed output variable y_i, and we want to learn the structure of the data from the data alone. For example, it is not possible to apply a classification or regression problems directly, because there are no response variables to predict. That is why this learning method is referred to as *unsupervised* since there are no response variables that can guide our algorithm. Unsupervised learning needs to find the relationships between the variables or between the observations to understand the data.

The most common methods operate in the unsupervised learning domain are

- Cluster analysis or clustering: It aims to group or segmenting observations in a data set into clusters. The observations in the same cluster are more closely each other than observations the different clusters. Grouping a data set with similar patterns reduces the complexity of the data and makes it easier to interpret the data. For details see [2, Chapter 14.3]

- Principal Component Analysis (PCA): PCA is a method to dimensionality reduction of a data set. PCA chooses the most meaningful component with the high variation along the features in a data set to filter out the noise. See Section 4.1.

- Anomaly detection: Anomaly detection classifies the input data as *normal* and *abnormal* based on past data. For example, this method is used in the computer security area for intrusion detection by labeling each the behavior as normal or abnormal with past behavior using user profiles. For details see [49].

- Association rule mining (ARM): ARM aims to find patterns, interrelations, associations or casual structures among the observations in a data set. The commonly used ARM algorithms are *Apriori Algorithm* and *Eclat Algorithm*. For details see [50], [51].

Figure 2.1 shows a simple illustration of the clustering problem [1, Chapter 2.1.4]. It is shows a plot of 150 observations on two variables, X_1 and X_2. Each data point corresponds to one of three distinct group. On the left-hand side of Figure 2.1, clustering of this is more easy than the left side. Because these groups are well-separated. On the contrary the right plot is some serious challenging problem, since an observation can be assigned to more than one group. There is some overlap between the groups. In this situation, a clustering algorithm could not be expected to classify to each overlapping points correctly (blue, green, or orange in Figure 2.1.

2.4 Semi-Supervised Learning

Majority of the problems that fall within Machine Learning domain are solved either with supervised or with unsupervised learning methods. However, while some of the data contain only input variables, the remaining data may be labeled with the response variable. For example, suppose our data set contains of n observations $X = \{x_1, \ldots, x_m, \ldots, x_n\}$ where $m < n$. There is a number of m observations have both predictor measurements and response measurement $Y = \{y_1, \ldots, y_m\}$, whereas the remaining $n - m$ observations in X have only predictor measurements.

In generally we have a small amount of labeled data in X, since collecting response measurement may be more expensive than predictors. Because there is a need to be human annotators to build classifiers. When is comes to the unlabeled data, unlabeled data is relatively easy to collect.

When there are both labeled and unlabeled samples in the data, semi-supervised learning is useful because it combines supervised and unsupervised learning [1, Chapter 2.1.4]. The most common semi-supervised learning techniques are:

- Generative Mixture Models and EM: This is a probabilistic way of assumes a generative model $p(x,y) = p(y)p(x|y)$ where $p(x|y)$ is an identifiable mixture distribution [52]. When we have a large amount of unlabeled data, we can identify

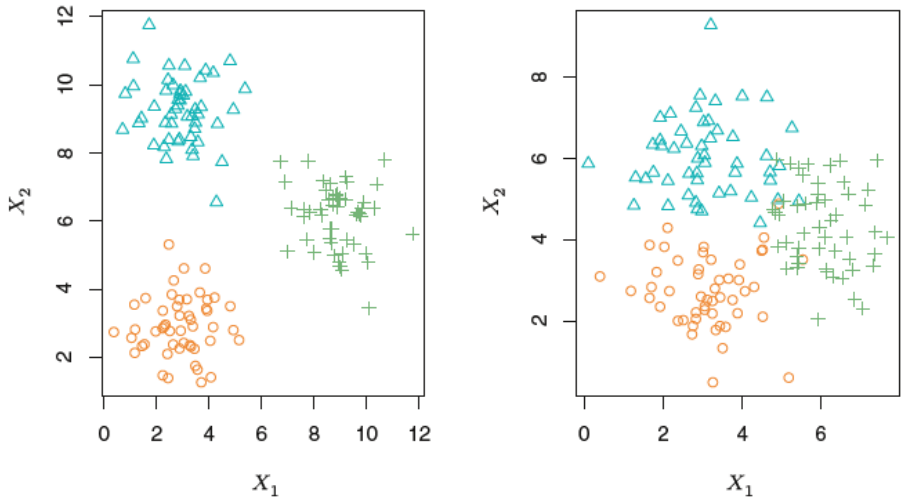


Figure 2.1 : In this clustering data set, each group is shown in different colors. The left plot shows the well separated groups. Clustering in these groups will be successful. The right plot shows there is some overlap among the groups. In this case, it is more difficult to cluster. [1, Chapter 2.1.4].

the mixture components. The model needs to only one labeled data to identify the per component to construct the mixture distribution. We need to pay attention to a few things while building a model. The model should be identifiable. The model has to be also correct to improve the accuracy. Otherwise, unlabeled data may actually lead to lower accuracy. Even if the model is correct, in practise mixture components may be found wrongly. Because the model uses Expectation-Maximization (EM) algorithm to founding local maximum. In the case of the local maximum is far from the global maximum, the unlabeled data does not contribute to the accuracy of the model. For details see [53].

- Transductive Support Vector Machines (TSVM): An extension of the classical support vector machines with unlabeled points is called Transductive Support Vector Machines. Standard SVM maximizes the the margin for only labeled points, TSVM maximizes the margin both for labeled and unlabeled points when construct the model. For details see [54].

- Graph-Based Methods: This method defines a graph to represent the data by classifying the nodes are labeled and unlabeled points. This method also reflects the similarity of the samples with edges. Thus, the data can be represented by a graph-based approach to semi-supervised learning. For details see [55] and [56].

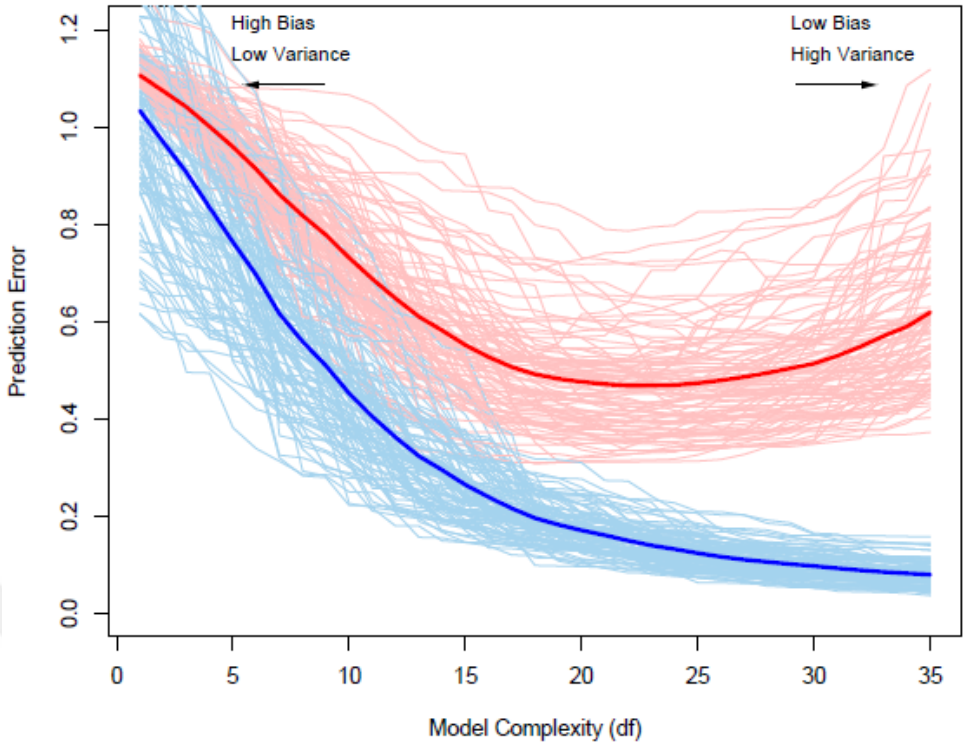


Figure 2.2 : The behavior of test error and training error according to the change of the model complexity. The training error $\overline{err}$ indicated by blue curve, and the conditional test error Err_τ indicated by red curve. The solid curves also indicate the expected test error Err and the expected training error $E[\overline{err}]$. There are simulated 100 training sets each of size 50 [2, Chapter 7.2].

2.5 Cross-Validation

In this Section we primarily use [2, Chapter 7.10]. We have two main objectives that are model selection and model assessment to the best approaches of our problems. In the model selection, we compare the estimating performance of candidate models and select the best one. After selecting the best model, we estimate the prediction error of the chosen model on the future data set.

Cross-validation is a statistical method that assesses model performance. In the model assessment part of our task, note that we will use a tuning parameter α to control our model complexity by varying α. The estimation of the prediction error of our model denoted by $\hat{f}(x,\alpha)$ under the tuning parameter α. The parameter α controls the learning process to minimize the error while the optimize main parameters of our model. The tuning parameter α allows us vary the model and allows us to control trade-off between the bias and variance in minimizing the error. (See Figure 2.2).

2.5.1 Test and training error

Before talking about cross-validation methods, let us look at some definitions. We optimize the parameters of the model by using sample observations from the data. We both train the model and test it using the data. To do this, we split the data into the training and the test set. The data that the model uses to construct the model is called *training data set*, and *the test data set* is the data used for testing the generated model. The error calculated on the training data set is called *training error rate*. We calculate the training error by averaging the loss on the training set

$$\overline{err} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}(x_i, \alpha_i)) \tag{2.1}$$

We predict response on a new observation using the model on the test data set. The average error resulting from the prediction is called the *test error*, and it is also called as generalization error and it is denoted by

$$Err_\tau = E[L(Y, \hat{f}(X, \alpha)|\tau] \tag{2.2}$$

Suppose we have a model $\hat{f}(X, \alpha)$ that we constructed using the training set τ where our data consists of points X vectors of inputs $\{x_1, \ldots, x_n\}$ and a response variable Y. We predict the response variable Y using the model with some error. There are we use a loss function to measure the difference between our prediction of the target value and the real response value. The loss function is denoted by $L(Y, f(\hat{X, \alpha}))$.

The two most common loss functions are:

$$L(Y, \hat{f}(X, \alpha)) = \begin{cases} (Y - \hat{f}(X, \alpha))^2 & \text{squared error} \\ |Y - \hat{f}(X, \alpha)| & \text{absolute error} \end{cases} \tag{2.3}$$

In cross validation we estimate the expected test error:

$$Err = E[L(Y, \hat{f}(X, \alpha))] = E[Err_\tau] \tag{2.4}$$

For example, if we use a linear fitting method is defined as

$$\hat{y} = Sy. \tag{2.5}$$

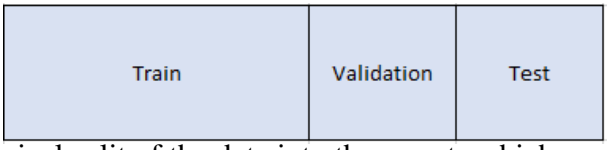


Figure 2.3 : A typical split of the data into three parts which are 50% for training, 25% for validation and the remaining 25% for testing.

where S is the NxN matrix depending on the x_i only not on y. The $\hat{y}$ indicates the predictions of the outcomes y_i using the model. The set of models for many linear fitting methods by choosing the loss function is squared error,

$$\frac{1}{N}\sum_{i=1}^{N}[y_i - \hat{f}^{-i}(x_i, \alpha_i)]^2 = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{y_i - \hat{f}(x_i, \alpha_i)}{1 - S_{ii}}\right]^2, \tag{2.6}$$

where S_{ii} is the ith diagonal element of S. Thus, the generalized cross-validation approximation is given by

$$GCV(\hat{f}, \alpha) = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{y_i - \hat{f}(x_i, \alpha_i)}{1 - trace(S)/N}\right]^2. \tag{2.7}$$

where $trace(S)$ is the effective number of parameters that is equal to the sum of the elements on the main diagonal of S. We use $trace(S)$ instead of the S_{ii} for easier to compute. In cases, matrix S is the projection of a set which is spanned by P features, $trace(S) = P$. The model complexity controls in this way, since the model complexity is associated with the number of parameters.

2.5.2 Cross-validation methods

We will explore the following cross-validation techniques below:

1. Resubstitution validation

2. Hold-out validation

3. K-fold cross validation (CV)

4. Leave-one-out-cross-validation (LOOCV)

The simplest kind of validation set approach is the *resubstitution validation*, the whole data set is used to train the model and tests the model on the same data set. This

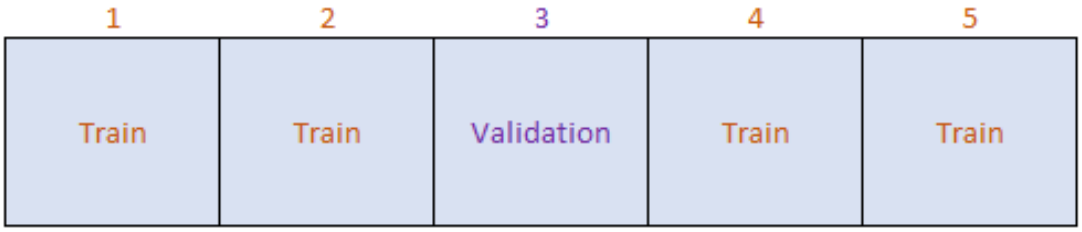


Figure 2.4 : In 5-fold cross-validation, the data set is randomly splitted into 5 equal sized subsets.

procedure therefore suffers from over-fitting. The model works well in the available data but does not work well to the future data points.

One other approach for cross-validation is the *hold-out validation*. This method randomly split the data points into two distinct sets are training and test. There is no general rule to split the data points in which for a train set, a validation set and a test set. In general, we split 50% the observations for the training set, 25% for the validation set and 25% for the test set. We build a model on the training set and test its performance on the test set. In cross-validation we run the model multiple times for testing, in contrast we test it once in the hold-out validation method. In order to avoid misleading results, we need to consider this when evaluating our model with hold-out validation.

When we have enough data, we can set aside a validation set to use it to assess the performance of our forecasting model. But when we train the model and test it once using a simple validation approach like the hold-out cross-validation, we find the performance of the model once. The results depends on how we split the data as the test and the training set. When we do this only for once, we might make a mistake in assessing the model and the result is not confident. To handle this problem, *K-fold cross validation* uses part of the data to fit the model, while the different part tests the model.

For K-fold cross validation, we split the data into K equal-sized parts. For example Figure (2.4) depicts the case for $K = 5$. We fit the model for the Γth part on the remaining $K-1$ parts of the data set as shown in Figure 2.4. Then, we predict the Γth part of the data and calculate the prediction error by applying the fitted model for $\Gamma = 1,2,\ldots,K$ and combine these K estimates of prediction errors.

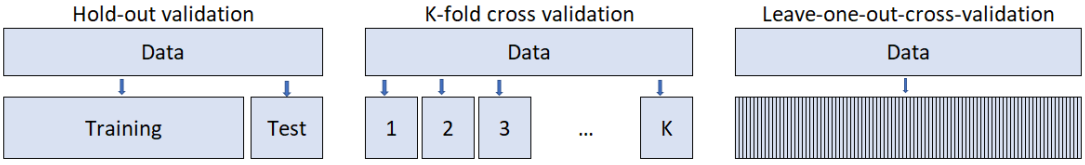


Figure 2.5 : Cross-validation set approaches

Let us define an indexing function that indicates the partition to which observation i is allocated by the randomization. $\Gamma : \{1,\ldots,N\} \mapsto \{1,\ldots,K\}$. $\hat{f}^{-\Gamma}(x)$ is the fitted function to remove the Γth part of the data. The cross-validation estimates the error by averaging these K estimates of the prediction errors:

$$CV(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}^{-\Gamma(i)}(x_i)). \tag{2.8}$$

The special case of cross-validation where $K = N$ the number of points in the data set is called *leave-one-out-cross-validation*. This means that a number of folds equals to the number of instances. For a data point (X,Y), if $\Gamma(X) = i$ we are going to use (X,Y) for validation, i.e. to test the model. The $\hat{f}^{-\Gamma}(x,\alpha)$ denotes the model that we fit the data by leaving out the Γ-th partition using α as the tuning parameter.

Then, we calculate the average error

$$CV(\hat{f},\alpha) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}^{-\Gamma(i)}(x_i,\alpha_i)). \tag{2.9}$$

Tuning parameter α_i minimizes the error for the function $L(y_i, \hat{f}^{-\Gamma(i)}(x_i,\alpha_i))$

2.5.3 The right way to do cross-validation

Now assume that we are given a classification problem that has a large number of predictors (features). A typical way of implementing cross-validation analysis can be:

1. Randomly dividing the data set into K folds.

2. Apply the following steps for each fold $\Gamma = 1,2,\ldots,K$.

   (a) Choose a subset of predictors that depending on the correlation between the class labels, using all of the samples except those in the fold Γ to reduce the error rate. When we have a large number of predictors in the data set, some predictors might have weak correlation with the features and may mislead the model.

(b) Use only this subset of predictors when building the classifier. We choose a random set of samples using K-fold cross validation. Then we compute the correlations of pre-selected predictors with class labels over just this samples chosen without those in fold Γ.

(c) Predict the class labels for the samples in the Γ-th fold using the classifier we built in the step before.

3. Estimate the tuning parameters using the cross-validation and find the prediction error using cross-validation estimator of the final model. The process results of (2c) for each K folds are the estimate of prediction error. The K-fold CV estimate is computed by averaging all results for each K-fold using 2.9.

3. BASIC MACHINE LEARNING ALGORITHMS

3.1 Logistic Regression

3.1.1 The logistic model

The logistic model is used to model the binary classification problems having output such as $0/1$. For instance, when we have a data set with binary response variable Y falls into one of the categories, Yes/No, logistic regression (LR) finds the probability of these two possible values. In the logistic model, we do not directly model the response variable Y, we model the probability of Y belonging to one of the two classes. $Pr(Y = Yes|X)$ indicate that the probability of Y given X where X is predictor. $Pr(Y = Yes|X)$ can be written as $p(X)$ shortly and its range between 0 and 1. Usually, we use a threshold value and when the probability satisfies $p(X) > 0.5$, we predict $Y = Yes$.

We need to relationship between the $p(X)$ and X using a function. In this method, we use a logistic function to build a model:

$$p(X) = \frac{e^{\phi_o+\phi_1 X}}{1 + e^{\phi_0+\phi_1 X}} \tag{3.1}$$

where ϕ_0 and ϕ_1 are the model parameters and X is the predictor. After simple manipulation, we find

$$\frac{p(X)}{1-p(X)} = e^{\phi_0+\phi_1 X}. \tag{3.2}$$

The ratio $\frac{p(X)}{1-p(X)}$ which is left side of the equation is called the *odds* and its range between 0 and ∞. After taking logarithm of the equation,

$$log\left(\frac{p(X)}{1-p(X)}\right) = \phi_0 + \phi_1 X. \tag{3.3}$$

In this equation $log\left(\frac{p(X)}{1-p(X)}\right)$ is called the *log-odds* or *logit*.

3.1.2 Estimating the regression coefficients

In logistic regression, the coefficients ϕ_0 and ϕ_1 in the equation (3.1) need to be estimated. Logistic regression uses the maximum likelihood estimation (MLE) to find the estimates of the values of the unknown parameters, thus achieving the predicted values of probability $p(x_i)$ for the each observation in the data set using (3.1). Thus we plug the estimated value of coefficients $\hat{\phi_0}$ and $\hat{\phi_1}$ in 3.1 to the model for $p(X)$.

We use the likelihood function to find the estimated coefficients by maximizing:

$$l(\phi_0, \phi_1) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i} \tag{3.4}$$

where x_i is the training data points, and the response variable is Y. After finding the coefficients, we make a prediction with computing the probability of Y. The products turn into the sums in (3.4) by taking logarithm:

$$\log(l(\phi_0, \phi_1)) = \sum_{i=1}^{n} y_i \log p(x_i) + (1 - y_i) \log(1 - p(x_i)) \tag{3.5}$$

$$= \sum_{i=1}^{n} y_i \log p(x_i) + \log(1 - p(x_i)) - y_i \log(1 - p(x_i)) \tag{3.6}$$

$$= \sum_{i=1}^{n} y_i \log \frac{p(x_i)}{(1 - p(x_i))} + \log(1 - p(x_i)) \tag{3.7}$$

We know $log\left(\frac{p(X)}{1-p(X)}\right)$ is the logit from the equation (3.3). So we substitute the equation $\phi_0 + \phi_1 x_i$ in the last step:

$$= \sum_{i=1}^{n} y_i(\phi_0 \phi_1 x_i) + \log((1 - p(x_i)) \tag{3.8}$$

And then substitute the logistic function to the $p(x_i)$, the equation becomes:

$$= \sum_{i=1}^{n} -log1 + e^{\phi_0 \phi_1 x_i} + \sum_{i=1}^{n} y_i(\phi_0 + \phi_1 x_i) \tag{3.9}$$

We take the derivative of the log-likelihood with respect to the unknown parameters ϕ_0, ϕ_1 and equal it to zero to find maximum likelihood estimations of the parameters approximately:

$$\frac{\partial l}{\partial \phi_0} = 0 \quad \text{and} \quad \frac{\partial l}{\partial \phi_1} = 0 \tag{3.10}$$

3.1.3 Multiple logistic regression

It is possible to extend (3.3) when we have multiple predictors in a data set.

$$log\left(\frac{p(X)}{1-p(X)}\right) = \phi_0 + \phi_1 X_1 + \cdots + \phi_p X_p, \tag{3.11}$$

where $X = (X_1, \ldots, X_p)$ are the predictors we have. Thus, the probability $p(X)$ is as follows

$$p(X) = \frac{e^{\phi_0+\phi_1 X_1+\cdots+\phi_p X_p}}{1+e^{\phi_0+\phi_1 X_1+\cdots+\phi_p X_p}}$$

We use the maximum likelihood estimation to the find the coefficients $\phi_0, \phi_1, \ldots, \phi_p$ again.

3.2 The Bayes Classifier

The Bayes classifier calculates the conditional probability of each observation $X = x_0$ and assigns the observation to the class for which largest probability. The conditional probability of Y given X

$$Pr(Y = j|X = x_0) \tag{3.12}$$

where $X = x_0$ is a predictor vector to the class $Y = j$. The Bayes classifier rule in the binary problem is that assign an observation to the class 1 or class 2 using conditional probability (3.12) like below.

$$\begin{cases} class\,1 & \text{if } Pr(Y = 1|X = x_0) > 0.5 \\ class\,2 & \text{otherwise} \end{cases} \tag{3.13}$$

Consider the example of data set simulated in a binary problem. There are the features X_1 and X_2 in the dimensions. Each class in the data set has the $N = 100$ points that showed a different color whose color depends on the class in the figure. Also, the region of the class represented by the background grid color. Bayes classifier finds the conditional probability $Pr(Y = orange|X) > 0.5)$ of the points in order to obtain their back ground color identifying the region of the class. The Bayes decision boundary occurs after assign the class of each observation. The boundary shown by purple dashed line in the mentioned figure.

The Bayes classifier gives the lowest error rate on the test data. The error rate is called Bayes error rate. The reason for the lowest error rate is that the Bayes classifier assigns

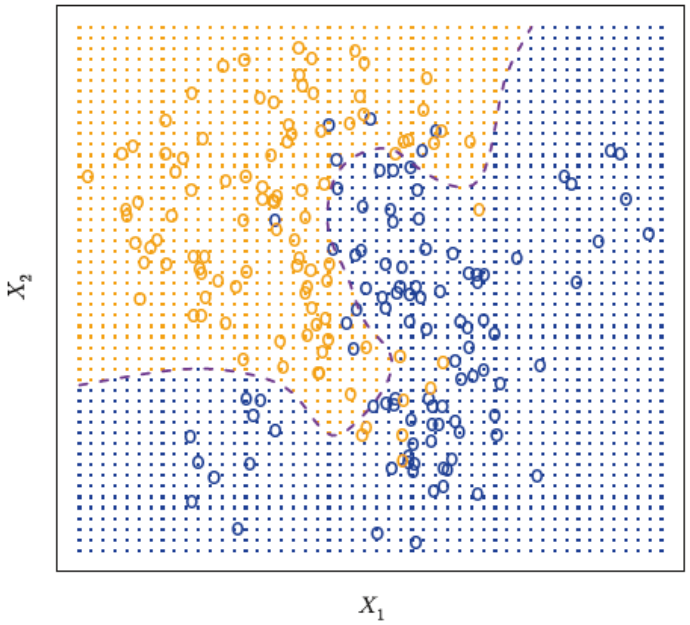


Figure 3.1 : The data points are simulated in different groups, shown in blue and in orange. Bayes decision boundary is shown in purple dashed line. [1, Chapter 2.2.3]

the point to the target value which for largest conditional probability (3.1). The Bayes error rate defined as follows:

$$1 - E\Big(max_j Pr(Y = j|X)\Big), \tag{3.14}$$

where the E denotes the averages the probability of the all observations X in the data set.

3.2.1 Mathematical background

3.2.1.1 Using Bayes' Theorem for classification

[1, Chapter 4.4.1]

Suppose we have more than two classes $K \geq 2$. Let π_k denotes the prior probability of an observation from the kth class. The $density function$ of X for any observation in which for kth class is defined as follows:

$$f_k(x) \equiv Pr(X = x|Y = k) \tag{3.15}$$

The *Bayes' Theorem* is given by

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x).} \tag{3.16}$$

Note that the $p_k(X)$is an abbreviation of $Pr(Y = k|X)$. We will see that in the linear discriminant analysis chapter, LDA finds estimate the values of π_k and $f_k(X)$. The LDA such a classifier that approximates the Bayes classifier.

4. LINEAR ALGEBRAIC METHODS

4.1 Principal Component Analysis

Principal Component Analysis (PCA) is an unsupervised machine learning method for reducing the number of features given in a data set. This procedure of reducing the number of features in a data set is called *dimensionality reduction*. This method allows us to reduce the number of variables correlated among themselves by selecting the basis vectors with higher variability for the original set. In short, PCA finds the directions along which the data has maximum variability.

4.1.1 Mathematical background

We used [57] for most of the material covered in this subsection. Now, let us summarize the procedure of applying PCA to a given data set.

4.1.1.1 Calculation of the Covariance Matrix

We calculate the covariance matrix of the data. If n is the number of features, the resulting matrix will be a $n \times n$-matrix.

$$\Sigma = \begin{bmatrix} Cov(x_1,x_1) & Cov(x_1,x_2) & \cdots & Cov(x_1,x_n) \\ Cov(x_2,x_1) & Cov(x_2,x_2) & \cdots & Cov(x_2,x_n) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(x_n,x_1) & Cov(x_n,x_2) & \cdots & Cov(x_n,x_n) \end{bmatrix}$$

where $x_1, x_2, \cdots, x_n$ are the features of the data and

$$Cov(x_i,x_j) = \sum_{k=1}^{m} \frac{(x_i^{(k)} - \bar{x_i})(x_j^{(k)} - \bar{x_j})}{(m-1)} \tag{4.1}$$

where m is the number of data points. We start by subtracting the mean from each of the features. Below, we are going to use *Data Adjust* to denote the data features with the means subtracted.

$$DataAdjust = (x_i^{(k)} - \bar{x_i}), \quad i = 1,\ldots,n \quad k = 1,\ldots,m \tag{4.2}$$

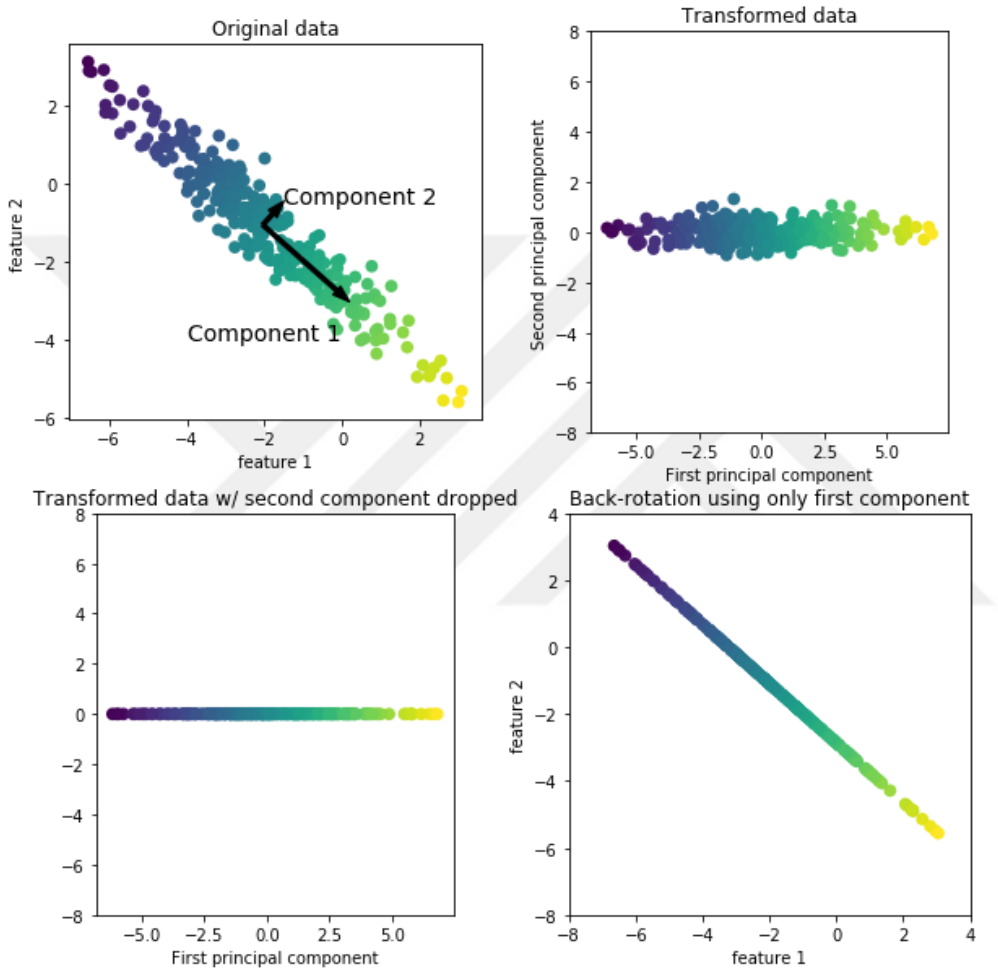


Figure 4.1 : A simple illustration of a kind of two-dimensional data on the top left panel. The plot of transformed data applying PCA is shown on the right top panel. The bottom left panel shows the data after apply PCA keeping only first principal component. The bottom right panel shows the illustration of the reconstruction data using only the most significance component. We used the Mglearn library to plot this illustration.

We use the terms *feature* and *dimension* interchangeably. All data points in each dimension minus the average of the dimension yields a feature centered with mean 0.

The entries on the non-diagonal of Σ is the covariances between each features. A positive covariance means the features are positively related, while a negative covariance means the features are inversely related. By this we mean, with positively correlated features if one increases the other one increases as well, and with inversely correlated features if one increases the other one decreases.

4.1.1.2 Calculation of the eigenvectors and eigenvalues of the covariance matrix

Finding eigenvalues and corresponding eigenvectors of the covariance matrix gives us important information about the data. The eigenvalues tell us how much variance there is in the data set along the direction of the eigenvector corresponding to this eigenvalue. Since the covariance matrix is symmetric and positive, all these eigenvalues are positive real numbers. The covariance matrix is a real $n \times n$ symmetric matrix, then the matrix has n orthogonal eigenvectors with the real eigenvalues according to Spectral Theorem [58, Theorem 5.14]. Thus the eigenvectors are of unit length, and are orthogonal to each other. This allows the PCA to express the data in these eigenvectors rather than in the original axes.

Since the eigenvalues tells us the variability of the data along the eigenvector corresponding to that spesific eigenvalue, when we sort the eigenvalues from largest to smallest we also sort the corresponding eigenvectors with respect to the significance of information of the data along the corresponding eigenvectors. The corresponding eigenvector of the largest eigenvalue is the principal component of the data set. It is the most significant direction of variance in the data, and the components along the smaller eigenvalues can be excluded to reduced the number of features.

After this dimensionality reduction step, the final data has fewer dimensions than the original one. To do this we need to form a new set of *feature vectors* [57]. If we choose the e eigenvectors that correspond to the e largest eigenvalues,

$$FeatureVector = (eig_1 \dots eig_e) \tag{4.3}$$

where $e \leq n$.

In order to construct the new features we use a transformation matrix that sends a vector written in the standart basis to a new basis given by the eigenvectors follow by a projection matrix that deletes the eigenvectors with small eigenvalues.

4.1.1.3 Deriving the new data set

To derive the new data set according to the chosen eigenvectors, firstly, we need to multiply transpose of the $FeatureVector$ by transpose of the $DataAdjust$. Thus the final data set is

$$FinalData = RowFeatureVector \times RowDataAdjust \tag{4.4}$$

The latest data gives us only the original data for the eigenvectors we selected. The eigenvectors are perpendicular, it means that the components are perpendicular too. This helps us an efficient expression of the data. Data have projected to the eigenvectors. Thus, we reduced the number of dimensions of the original data by PCA.

On the other hand, we can keep all eigenvectors on the data using PCA. There are the eigenvectors create the directions of the new axis because their lengths are 1. This gives us a projected to a data sets on the new axis to analyzing the data efficiently. Principal components are derived from projecting the data to the eigenvectors that maximize the variance along that vectors which are the new axis of our data. Even if we do not reduce the number of features, by changing coordinates we get a new set of features, ordered with respect to their variability.

4.1.1.4 Can we get back to the old data?

New data set can get back to the original data set depending on the previously chosen eigenvectors. If we have chosen them all, we will get the original data back exactly. Since we removed some eigenvectors, we lost some information from the original data when we reconstruct the data. Recall that the final transform expressed in the form:

$$FinalData = RowFeatureVector \times RowDataAdjust, \tag{4.5}$$

from this transform, to get the original data back,

$$RowDataAdjust = RowFeatureVector^{-1} \times FinalData \tag{4.6}$$

where $RowFeatureVector^{-1}$ is the inverse of $RowFeatureVector$.

The equation can be easily writing that

$$RowDataAdjust = RowFeatureVector^{T} \times FinalData \tag{4.7}$$

Then to get the original data back, adding the mean that subtracted before. So, for completeness,

$$RowDataAdjust = (RowFeatureVector^{T} \times FinalData) + OriginalMean \tag{4.8}$$

This formula still makes the right transformation to retrieve old data back even if we do not have all the eigenvectors in the feature vector.

4.1.2 The advantages and disadvantages of using PCA

When the data is high dimensional, and therefore, hard to represent graphically PCA helps us to reduce the number of features and makes it easier to visualize and recognize patterns to analyze. Another advantage of using PCA is reducing the noise since PCA chooses a basis with maximum variation deleting vectors yielding small variations. The difficulty in using PCA is the fact that it is difficult to evaluate the covariance matrix accurately [59].

There are some limitations to apply PCA on a data set. PCA has an assumption that the data has a multivariate Gaussian distribution even if the data has a non-Gaussian distribution. If the data has multivariate normal distributions, the correlations of the features are zero. This implies that the features are orthogonal each other. In this case PCA works well to projection of the data points in the lower dimensional. On the other hand, PCA can keep very few data points for data having non-Gaussian distributions. Independent component analysis (ICA) is an extension of PCA to challenge this issue to fit the data by finding better bases that do not have to orthogonal [60]. The other limitation is that PCA gives us the linear components to fit the data even if the components are not appropriate to keep in our data. In some cases, we need a non-linear principal components. Kernel PCA (KPCA) is an extension of PCA to solve this problem. For details see [61]. We also cannot apply PCA directly to the categorical

data. In this case, different techniques are needed to calculate the covariance matrix and apply the method. One special extension is multiple correspondence analysis (MCA) for categorical data set [62].

4.1.3 Singular value decomposition

X be a real $n \times m$ matrix where $n \geq m$ and the decomposition of X is

$$X = U\Gamma V^T \tag{4.9}$$

where U is a $n \times m$ orthonormal matrix $(U^T U = I)$, V is $m \times m$ orthonormal matrix $(VV^T = I)$, Γ is a $m \times m$ diaogonal matrix, which consists of the square-roots of the eigenvalues of XX^T and $X^T X$ called singular values in descending order.

4.1.4 Relationship between PCA and singular value decomposition

Implementing PCA on data which is very large data sets can be expensive, but applying PCA using SVD provides us an advantage.

Assume X is an $n \times m$ data matrix where $n \geq m$. The rows in X denote the features $j = 1, \ldots, n$ and the columns denote the samples $\zeta = 1, \ldots, m$. It means that the components are $X_{j,\zeta} = x_j^{\zeta}$. The mean vector is

$$\langle x \rangle \equiv \frac{1}{m} \sum_{\zeta=1}^{m} x^{\zeta} \tag{4.10}$$

and the empirical covariance matrix is given by

$$\Sigma \equiv \frac{1}{m} \sum_{\zeta=1}^{m} (x^{\zeta} - \langle x \rangle)(x^{\zeta} - \langle x \rangle)^T \tag{4.11}$$

Covariance matrix of X can be written using the matrix formulation as below:

$$\Sigma \equiv \frac{1}{m} XX^T \tag{4.12}$$

where we have subtracted the mean of the data set: $X_{j,\zeta} := X_{j,\zeta} - \langle x_j \rangle$. X is decomposed using SVD, i.e.

$$X = U\Gamma V^T \tag{4.13}$$

Decomposed X can be written in the equation of covariance matrix. The columns of U are the eigenvectors of covariance matrix Σ.

$$\Sigma = \frac{1}{m} XX^T = \frac{1}{m} U\Gamma^2 U^T \tag{4.14}$$

where U is $n \times m$ matrix, if $n < m$, the first n columns in U corresponds to the sorted eigenvalues of Σ and if $m \geq n$, the first m corresponds to the sorted non-zero eigenvalues of Σ in descending order.

The final data which is transformed using by SVD can thus be given by

$$FinalData = \hat{U}^T X = \hat{U}^T U \Gamma V^T \tag{4.15}$$

where $\hat{U}^T U$ is a $n \times m$ matrix which entries on the diagonal are all one and zero everywhere else. Thus we derived the final data in terms of the SVD decomposition of X [63].

4.2 Linear Discriminant Analysis

Our main source for this Section is [1, Chapter 4.4].

Linear Discriminant Analysis (LDA) is a supervised feature extraction method for classification of data and dimensionality reduction. LDA maximizes the ratio of the variance between the classes to the variance within the classes in data set, thus achieving maximal separability [64].

We know that the logistic regression model the probability $Pr(Y = k|X = x)$ of the response variable in which for one of the kth classes using the logistic function (Section 3.1). LDA models the distributions of the predictors X in which each class separately. And then LDA uses Bayes' Theorem (See Section 3.2) to estimate the probabilities $Pr(Y = k|X = x)$.

4.2.1 LDA for $p = 1$

In this section, we show that approach of the LDA when we have one only one dimension $p = 1$ in the data set. We shall suppose that the density function $f_k(x)$ is Gaussian or normal as follows:

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} exp\left(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right), \tag{4.16}$$

where μ_k is the mean and σ_k^2 is the variance in which for the k-th class. We shall assume that the variances of all k classes are the same $\sigma_1^2 = \cdots = \sigma_k^2$, where we set σ^2 as the variance for the all classes. Plugging (4.16) into the Bayes' Theorem (3.16) we get:

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{1}{2\sigma 2}(x-\mu_k)^2)}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{1}{2\sigma 2}(x-\mu_l)^2)} \tag{4.17}$$

Bayes classifier assigns the class for the data point $X = x$ in which for the value of $p_k(x)$ is largest.

Taking the log and rearranging terms of equation (4.17), we get the equation to assign the class for an observation with the same rule as follows:

$$\eta_k(x) = x.\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + log(\pi_k) \tag{4.18}$$

In the case of we have two classes $K = 2$ and $\pi_1 = \pi_2$, the assignment rule is given by the Bayes classifier

$$\begin{cases} class1 & \text{if } 2x(\mu_1 - \mu_2) > \mu_1^2 - \mu_2^2 \\ class2 & \text{otherwise} \end{cases} \tag{4.19}$$

The Bayes decision boundary is given by

$$x = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} = \frac{\mu_1 + \mu_2}{2} \tag{4.20}$$

LDA classifier uses the Bayes classifier by estimating the parameters $\mu_1, \ldots, \mu_k, \pi_1, \ldots, \pi_k, \sigma^2$ and plugging the estimates into (4.18).

$$\hat{\mu_k} = \frac{1}{n_k} \sum_{i:y_l=k} x_i \quad \text{and} \quad \hat{\sigma^2} = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_l=k} (x_i - \hat{\mu}_k)^2 \tag{4.21}$$

where n is the number of training observations, n_k is the number of training observations in kth class. In practice, we need to estimate the parameter π_k

$$\hat{\pi_k} = \frac{n_k}{n} \tag{4.22}$$

The estimated $\hat{\mu}_k, \hat{\sigma^2}$ and $\hat{\pi}_k$ plugs into (4.18) and the LDA classifier can assign an observation for which

$$\hat{\eta}_k(x) = x.\frac{\hat{\mu}_k}{\hat{\sigma^2}} - \frac{\hat{\mu_k^2}}{2\hat{\sigma^2}} + log(\hat{\pi}_k) \tag{4.23}$$

is largest.

4.2.2 LDA for $p > 1$

Predictor variable X extend the multiple predictors $X = (X_1, X_2, ..., X_p)$. We shall further assume that the predictors come from multivariate Gaussian or normal distribution that has mean vector for each class and has the common covariance matrix. It means that p-dimensional variable X has a multivariate Gaussian distribution that denotes by $X \sim N(\mu, \Sigma)$ where the mean of X is $E(X) = \mu$ and covariance matrix of X is $Cov(X) = \Sigma$. In the multivariate dimensional setting with $p > 1$, the multivariate Gaussian density is given by

$$f(x) = \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma|^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)). \tag{4.24}$$

where μ_k is a mean vector of the kth class, $X \sim N(\mu_k, \Sigma)$ is a Gaussian distribution and Σ is a common covariance matrix for the all classes. We again plug the density function into (3.16). Thus Bayes classifier assigns to each point $X = x$ using the largest value of discriminant function

$$\eta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + log\pi_k \tag{4.25}$$

where μ_k is a mean vector of the kth class, and Σ is a covariance matrix. The class of the observation again determined by the largest value of the discriminant function which for the k class. The discriminant function is the same with (4.18). The difference of this discriminant function is that we need vector and matrix because we have the multidimensional observations.

Bayes decision boundary can separate the classes from each other by two by two. The intuition of the determination of the boundary is that we need to find the points that may be assign to both classes. The set of the points $X = x$ build us the decision boundary for $\eta_k(x) = \eta_l(x)$ as follows:

$$x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k = x^T\Sigma^{-1}\mu_l - \frac{1}{2}\mu_l^T\Sigma^{-1}\mu_l \tag{4.26}$$

for $k \neq l$. Thus, the predictor space is divided by Bayes decision boundaries. The unknown parameters $\mu_1, ..., \mu_k, \pi_1, ..., \pi_k$, and Σ are estimated with the same formulas as in (4.21) that we are already showed the one-dimensional case. The estimated values plugs into (4.25) to assign the data points $X = x$. Thus LDA classifies a point for which class $\eta_k(x)$ is largest.

4.2.3 The advantages and disadvantages of using LDA

The LDA method, which is easy to implement is well-suited for multi-class problems for classification and dimensional reduction. In contrast to PCA, LDA provides a feature subspace that maximizes class separability. However, there are also some disadvantages of using LDA. The method LDA assumes that the samples are drawn from a Gaussian or normal distribution with a common covariance matrix for each class. For this reason, the LDA may not work well if the observations do not have a Gaussian distribution. Quadratic discriminant analysis (QDA) extends LDA by allowing the covariance matrices for each class [65]. The LDA basically provides a linear classifier by finding a linear decision boundary between the classes so it may not perform well in non-linear data.

4.3 Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm used mainly for classification problems than regression problems. The purpose of the SVM is to find a set of optimal hyperplanes that divide our data set into disjoint subsets for classification. SVM chooses these hyperplane by maximizing the margins between data points of different classes.

4.3.1 Maximum margin classifier

Our main source for throughout this Section [3].

Suppose there are two classes need to classify using linear model of the form.

$$y(x) = w^T \theta(x) + b \tag{4.27}$$

where $\theta(x)$ is a feature space mapping, w is a weight vector and b is the bias parameter. We assume the training data set consist of N observations $x_1, .., x_N$, and $k_1, \ldots, k_N$ are

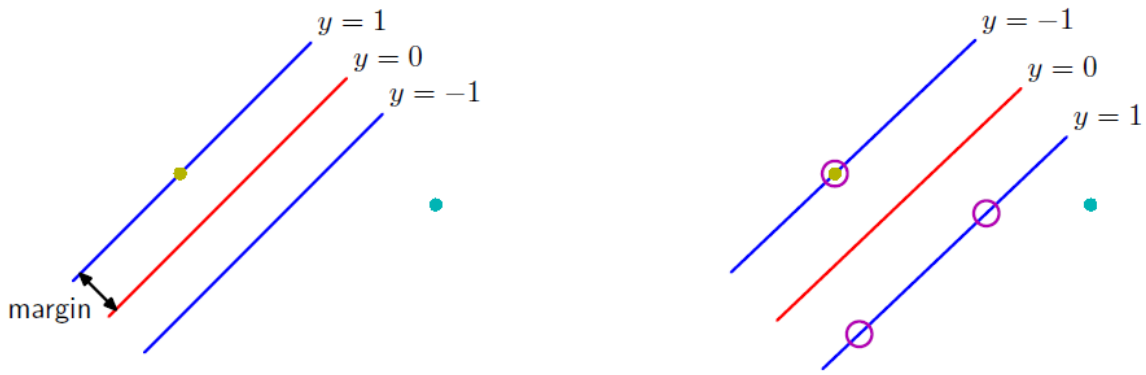


Figure 4.2 : The left figure shown is the margin. The right figure shown that the decision boundary found by maximizing the margin. The location of this boundary is determined by support vectors by maximizing margin. The indicated points by circles are the elements of the subset of the data points, that uses to determine the support vectors. [3, Chapter 7.1.]

the response variables of the given input vectors where $k_n \in \{-1, 1\}$. We predict the target values of the new data point x determined by the sign of $y(x)$.

When the training data set is linearly separable, one can find parameters w and b that satisfy $y(x_n) > 0$ for points having $k_n = +1$ and $y(x_n) < 0$ for points having $k_n = -1$. Hence $k_n y(x_n) > 0$ for all training data points. If there are too many solutions for these parameters that separate the classes, we need to try to find the best solution with the smallest generalization error. So, in SVM we compute the the margin which is defined as the largest distance between two class boundaries as shown in 4.2 [3, Chapter 7.1].

We need to measure the distance of the each observation from the boundary. The perpendicular distance shown by blue dotted line in Figure 4.3. The equation of the distance is given by using using Equation (4.27).

$$\frac{|y(x)|}{\|w\|} = \frac{|w^T \theta(x) + b|}{\|w\|} \tag{4.28}$$

When we proceed on the correct classified points it is means that $k_n y(x_n) > 0$ for all data points correctly classified, thus multiplying the distance equation by k_n, we find

$$\frac{k_n y(x)}{\|w\|} = \frac{k_n (w^T \theta(x) + b)}{\|w\|} \tag{4.29}$$

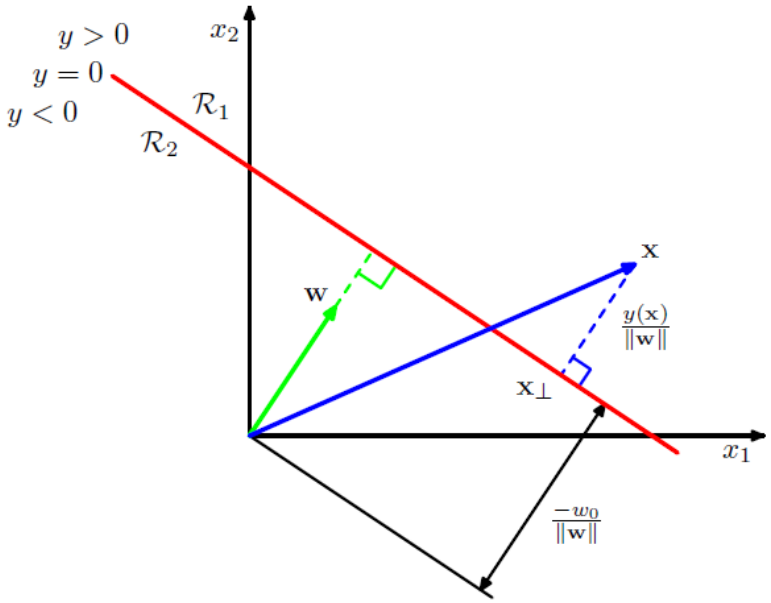


Figure 4.3 : The decision surface is shown by the red line, is perpendicular to w, and its displacement from the origin is controlled by the parameter w_0. The perpendicular distance of a point x to the decision boundary is given by $y(x)/||w||$. [3, Chapter 4.1.1.]

We can find the margin using this formulation by taking the closest point from the data set. SVM optimizes the parameters w and b in order to maximize the distance of the data points from the boundary. Thus and so, the optimal distance can be found by solving this optimization problem.

$$\arg\max_{w,b} \left\{ \frac{1}{\|w\|} \min_n [k_n(w^T \theta(x_n) + b)] \right\} \tag{4.30}$$

Since the optimization problem is complex to solve directly, we make it a simpler problem. We shall convert the optimization problem in order to easier solving the problem by rescaling w$\rightarrow \Gamma$w and b$\rightarrow \Gamma$b. And then the distance from any point x_n to the hyperplane is the same as mentioned before $\frac{k_n y(x_n)}{\|w\|}$ thanks to this scaling.

$$k_n(w^T \theta(x_n) + b) = 1 \tag{4.31}$$

The equation (4.31) holds for the closest point to the surface. For other points, this equality will be greater than one. Thus, we obtain the below constraints the all data points.

$$k_n(w^T\theta(x)+b) \geq 1, \qquad n = 1,...,N. \tag{4.32}$$

The constraint is *active* for data points that hold the inequality, while the constraint is *inactive* for the remaining points. The optimization problem that we need to solve, requires that we maximize $\|w\|^{-1}$, which is the same as minimizing $\|w\|^2$, then our optimization problem is

$$\arg\min_{w,b} \frac{1}{2}\|w\|^2 \tag{4.33}$$

subject to the given constraints by (4.32). This optimization problem is such a quadratic programming problem. Our purpose here is to minimize a quadratic function. We need to do this under the constraints we have while minimizing this problem. We use the method of Lagrange multipliers to solve the problem. The Lagrangian function is given by

$$\mathscr{L}(w,b,\dot{a}) = \frac{1}{2}\|w\|^2 - \sum_{n=1}^{N} a_n\{k_n(w^T\theta(x_n)+b)-1\} \tag{4.34}$$

where $\dot{a} = (a_1,...,a_N)^T$. These scalers $a_n \geq 0$ are the Lagrange multipliers, we have one of them, one for every constraint in (before). Following two conditions can be given on the derivatives of the $\mathscr{L}(w,b,\dot{a})$ with respect to w and b equal to zero.

$$w = \sum_{n=1}^{N} a_n k_n \theta(x_n) \tag{4.35}$$

$$0 = \sum_{n=1}^{N} a_n k_n \tag{4.36}$$

We eliminated w and b from $\mathscr{L}(w,b,\dot{a})$ using the above conditions. Then, we find the dual representation of the problem.

$$\tilde{\mathscr{L}}(\dot{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m k_n k_m k(x_n,x_m) \tag{4.37}$$

where the kernel function is $k(x,x^{'}) = \theta(x)^T\theta(x^{'})$. We need to maximize the problem under the constraints

$$a_n \geq 0, \qquad n = 1,...,N, \tag{4.38}$$

$$\sum_{n=1}^{N} a_n k_n = 0 \tag{4.39}$$

The same case in here, this is again the form of a quadratic programming problem with the constraints. When we want to classify new data points using the trained model, sign of $y(x)$ should be considered using (4.27). The $y(x)$ also can be written using the parameters $\{a_n\}$ and the substitute (4.35) into w for the kernel function.

$$y(x) = \sum_{n=1}^{N} a_n k_n k(x, x_n) + b \tag{4.40}$$

The optimization problem satisfies the Karush-Kuhn-Tucker (KKT) conditions under the constraints which are given below

$$a_n \geq 0 \tag{4.41}$$

$$k_n y(x_n) - 1 \geq 0 \tag{4.42}$$

$$a_n \{k_n y(x_n) - 1\} = 0 \tag{4.43}$$

Under these conditions, every data point satisfy either $a_n = 0$ or $k_n y(x_n) = 1$. The data points for which $a_n = 0$ have no contribution to the model, because these data points disappear in the equation (4.40). The remaining data points is the support vectors which obtain by $k_n y(x_n) = 1$, as shown in Figure 4.2 by the circles. After the model is trained using SVM, a large amount of the data points can be ignored only keeping support vectors to future points.

And then, we find the threshold parameter b considering any x_n satisfies $k_n y(x_n) = 1$ using (4.40).

$$k_n \left(\sum_{m \in S} a_m t_m k(x_n, x_m) + b \right) = 1 \tag{4.44}$$

where S is the set of indices of the support vectors. Here, first multiply the above equation through by k_n and then averaging the equations over all support vectors. Then the equation in above becomes using $k_n^2 = 1$, we find

$$b = \frac{1}{N_S} \sum_{n \in S} \left(k_n - \sum_{m \in S} a_m k_m k(x_n, x_m) \right) \tag{4.45}$$

where N_S is the total number of support vectors. The SVM minimizes an error function using regularization parameter λ

$$\sum_{n=1}^{N} E_\infty (y(x_n) k_n - 1) + \lambda \|w\|^2 \tag{4.46}$$

where

$$E_\infty(z) = \begin{cases} 0 & \text{if } z \geq 0 \\ \infty & \text{otherwise} \end{cases} \tag{4.47}$$

to in either case satisfying the constraints (4.32).

4.3.2 SVM soft margin classifier

We considered data points are linearly separable and then we used the SVM hard margin classifier even if the decision boundary nonlinear using *kernel trick*. Now consider the data is not linearly separable, in this case, the support vector machine with a hard margin does not really work well. We will change this approach with allows few training points to be misclassified with a *soft margin*.

With this approach, few points lie on the wrong side of hyperplane and we have to control it. To control the misclassified points, we use the slack variables $\xi_n \geq 0$ corresponding to each training points x_n for $n = 1,\ldots,N$. If the data points are on the correct side of the margin, these variables are given the value $\xi_n = 0$, while we set $\xi_n = |k_n - y(x_n)|$ for the remaining part of data points. This slack variable imposes a penalty for each misclassified points and the penalty function increases linearly with the slack variable dependent on the perpendicular distance from a point to the boundary. So that we can keep this approach under control even if the soft margin makes a few mistakes.

Any data point on the boundary satisfies $y(x_n) = 0$. Thus it will have $\xi_n = 1$ hence from the linear penalty function defined above. On the other hand, if the point is on the wrong side, the slack variable will have $\xi_n > 1$. The constraints given in (4.32) are then modified by the slack variables

$$k_n y(x_n) \geq 1 - \xi_n, \qquad n = 1,...,N \tag{4.48}$$

where the slack variables should satisfy $\xi_n \geq 0$.

If the corresponding slack variable satisfies $\xi_n = 0$, it says that the point is classified correctly. In this case, the point may be on the margin or lies on the correct side of the hyperplane. Data points for which $0 < \xi_n \leq 1$ are also classified correctly since

even though such points lie inside the margin, they are located on the correct side of the hyperplane. In such a case $\xi_n > 1$, the point is misclassified. (See Figure 4.4).

We now minimize the optimization problem

$$C\sum_{n=1}^{N}\xi_n + \frac{1}{2}\|w\|^2 \tag{4.49}$$

where $C > 0$ is regularization parameter that controls the trade-off between the margin and the size of slack variable penalty ξ_n.

The Lagrangian is to minimize (4.49) under these constraints (4.48) with $\xi_n \geq 0$.

$$\mathscr{L}(w,b,\dot{a}) = \frac{1}{2}\|w^2\| + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N}a_n\{k_n y(x_n) - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n \tag{4.50}$$

where the Lagrange multipliers $\{a_n \geq 0\}$ and $\{\mu_n \geq 0\}$.

The corresponding KKT conditions for the solution to be optimal is given as follows:

$$a_n \geq 0 \tag{4.51}$$

$$k_n y(x_n) - 1 + \xi_n \geq 0 \tag{4.52}$$

$$a_n(k_n y(x_n) - 1 + \xi_n) = 0 \tag{4.53}$$

$$\mu_n \geq 0 \tag{4.54}$$

$$\xi_n \geq 0 \tag{4.55}$$

$$\mu_n\xi_n = 0 \tag{4.56}$$

where $n = 1,\ldots,N$. We then optimize these parameters w, b and ξ_n using (4.27).

$$\frac{\partial\mathscr{L}}{\partial w} = 0 \implies w = \sum_{n=1}^{N}a_n k_n\theta(x_n) \tag{4.57}$$

$$\frac{\partial \mathscr{L}}{\partial b} = 0 \implies \sum_{n=1}^{N} a_n k_n = 0 \tag{4.58}$$

$$\frac{\partial \mathscr{L}}{\partial \xi_n} = 0 \implies a_n = C - \mu_n. \tag{4.59}$$

These parameters w, b and ξ_n eliminated from the Lagrangian using these results above and hence the dual Lagrangian found below:

$$\tilde{\mathscr{L}}(\dot{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m k_n k_m k(x_n, x_m) \tag{4.60}$$

which is the same formulation with the separable data set. But the constraints are slightly different. As mentioned before the Lagrangian multipliers $a_n \geq 0$. Also we find the $a_n \leq C$ using (4.59) with the $\mu_n \geq 0$. Hence we have these constraints

$$0 \leq a_n \leq C \tag{4.61}$$

$$\sum_{n=1}^{N} a_n k_n = 0 \tag{4.62}$$

for $n = 1, \ldots N$. The constraints (4.61) are called as *box constraints*. This is form of the quadratic programming problem. After substitute (4.57) into (4.27), we find again (4.40) in order to predict future data points.

The data points which have $a_n > 0$ generates the support vectors. In this case 4.53 have to satisfy this equality

$$k_n y(x_n) = 1 - \xi_n. \tag{4.63}$$

We find $\mu_n > 0$ hence from (4.59) for the case when $a_n < C$. Then the slack variables should be $\xi_n = 0$ to satisfy (4.56). This tells us the points are on the margin. When $a_n = C$, the points are inside the margin. There are two cases for the points inside the margin. If $\xi_n \leq 1$, the points correctly classsified or $\xi_n > 1$ means the points misclassified.

We mentioned above, in case of $0 < a_n < C$ have $\xi_n = 0$, and hence from $k_n y(x_n) = 1$. It follows below to the determine value of threshold parameter b

$$k_n \left(\sum_{m \in S} a_m k_m k(x_n, x_m) + b \right) = 1 \tag{4.64}$$

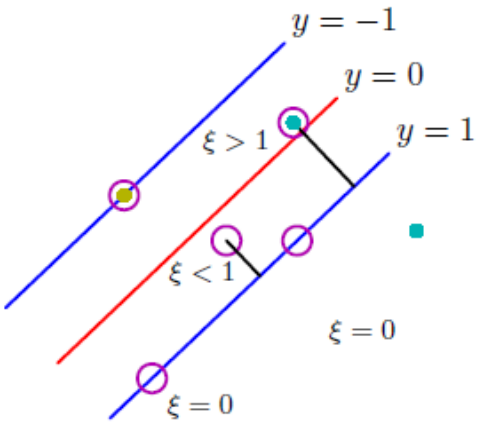


Figure 4.4 : Illustration of the slack variables $\xi_n \geq 0$. Data points with circles around them are support vectors. [3, Chapter 7.1.1]

Again the equation becomes by averaging

$$b = \frac{1}{N_M} \sum_{n \in M} \left(k_n - \sum_{m \in S} a_m k_m k(x_n, x_m) \right) \tag{4.65}$$

where M is the set of indices of points which for $0 < a_n < C$.

4.3.3 The advantages and disadvantages of using SVM

SVM is able to capable of doing classification and even regression when the data are not regularly distributed or have an unknown distribution. SVM can found a unique solution, since the optimally problem is convex. Once you have we have trained the model with SVM, the support vectors allow us to classify new point without keeping the other data points. Selection of the best support vector also controls the over-fitting problem on the model [66]. The model also provides us with the advantage of deriving the global optimum [66]. With SVM, we get good results by mapping using the kernel into higher dimensions to the increase the power of the learning even when the number of features in our data exceeds the number of data [67]. The method works well on the non-linear data set by controlling the non-linearity through the kernel trick [67]. However, the selection of the kernel may not easy, especially when there is a large amount of data, which takes too much time may be disadvantage. Although the model gives us good accuracy, it may not be easy interpret the model.

5. MATERIALS AND METHODS

5.1 The Olivetti Faces Data Set

5.1.1 Data exploration

Olivetti faces data set [4] consists of 400 face images of 40 different people. Each person has 10 different face images in the data set. The images contain different facial expressions. There are also glasses on some images of faces. All images were taken at AT&T Laboratories Cambridge with the same background in the same position. The image files are in PGM format. Each image is grayscale and contains 92×112 pixels. Each pixel has non-negative values from 0 for black to 255 for white in the original data set, but we use a version of the size 64×64. There are 400 rows and 4096 columns in the database. The rows contain images of each individual person and the columns contain the pixel of those images. There are 4096 columns here because the images are 64×64 pixels. There is also a response variable, an integer value from 0 to 39, which encodes which person these images belong to. The first ten rows belong to images of the people encoded with 0 and the second ten rows belong to the images of the people encoded with 1 and so on.

5.1.2 Method

5.1.2.1 Data preprocessing

After all, we easily fetched the data using *Scikit-learn* library [68]. We then created a dataframe from the array using *Pandas* package [69]. We then standardized the features for pixel scaling by subtracting the mean and scaling to unit variance. Figure (5.2) shows the 40 distinct faces with the target values in the data set.

5.1.2.2 Applying models

In this section, we built thirty-one different models using LDA (4.2), PCA (4.1) and SVM (4.3). Firstly, we classified the faces using LDA by projecting the data set onto

the different dimensional subspace. Secondly, we applied the SVM with a different types of kernels which are linear, radial basis, polynomial with 3 degree, and sigmoid function. As a third, we applied the SVM with choosing different kernels on the reduced data by PCA for the classification tasks. When using the PCA we transformed the data set onto a different number of principal components to interested in how to change the performance of the model. The cross-validation method split the data into three equal-size parts and then calculated the scores of the model three times with different splits instead of once time. Finally, we calculated the accuracy, precision, recall, f1-score metrics for the each model applied in this section.

5.2 Fashion-MNIST Data Set

5.2.1 Data exploration

Fashion-MNIST data set [5] consists of 70,000 fashion images that are labeled with one of 10 different classes which are t-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots. The 784 columns in the data set correspond to the 28×28 pixels of the images, while the first column is their label of the ten classes.

5.2.2 Method

5.2.2.1 Data preprocessing

We fetched the data set from github profile of Zalando Research [70]. The first column in the data refers to the corresponding label of the image. We extracted the first column from data sets for the response variable. After then we standardized the features for pixel scaling.

5.2.2.2 Applying models

We built thirty-one different models based on LDA (4.2), SVM (4.3) and SVM by combining PCA (4.1) throughout the experiment on the Fashion-MNIST data set. We applied the methods with the same approach as in the *Olivetti data set* experiment. LDA was used to classify the images by projecting the data set into different dimensional spaces. SVM performed using different kernel functions which are linear, radial basis, polynomial with 3 degree and sigmoid. PCA used for dimensionality

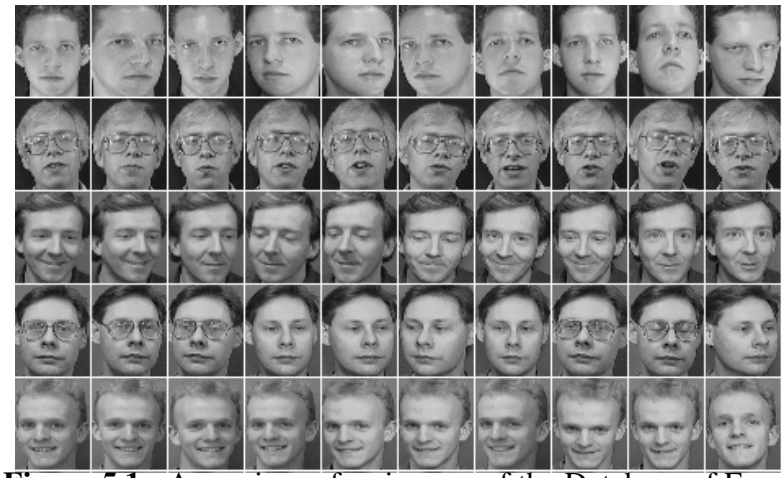


Figure 5.1 : A preview a few images of the Database of Faces

Figure 5.2 : The faces of 40 distinct people with corresponding target in the data set

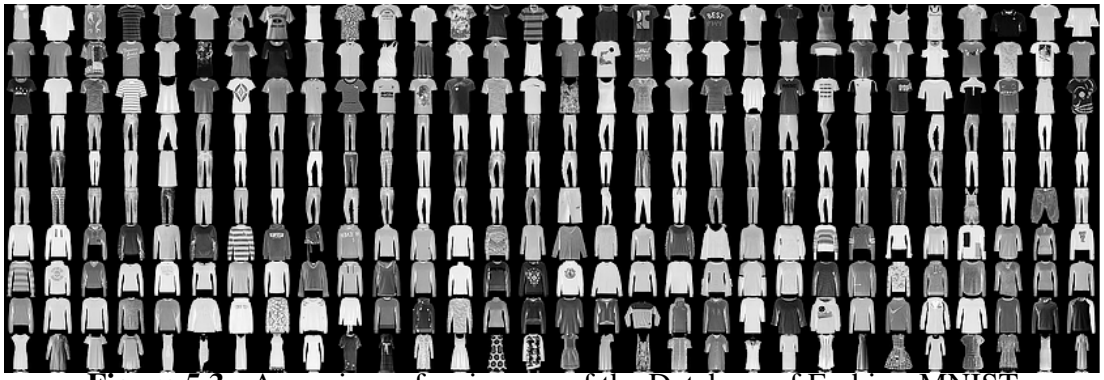


Figure 5.3 : A preview a few images of the Database of Fashion-MNIST

reduction of the data set and then SVM classified the images using different kernel functions on the reduced data set. The intuition of the combined SVM with PCA is that PCA applied to reduce the dimensions of the data set that are inputs to the SVM as in the previous experiment. We tested the performance of our models three times using 3-fold cross-validation.

5.3 MADELON Data Set

5.3.1 Data exploration

MADELON data set [6] is an artificial and highly non-linear which is used for binary classification problems. The data set was one of the NIPS2003 problems [71]. There are 32 clusters in the data set. The label of the points assigned with $+1$ or -1 randomly. Five-dimensional hypercube has these clusters on the vertices. Five dimensions of the hypercube represent 5 attributes of the data set. The data set also contains 15 linear combinations of these features which later forms into a set of 20 informative features. The samples that are based on 20 features labeled into 2 classes that go by +-1. A number of distractor feature which is called 'probes' were added. They have no predictive power. Thus, there are a number of 500 features, 20 of which are real, the remaining 480 are probes. The order of features and patterns were added randomly.

5.3.2 Method

5.3.2.1 Data preprocessing

We imported the data set by using urllib2 [72] which is extensible library for opening URLs. The data set consists of training set, validation set and test test. The number of observations of these sets is shown by Table 5.1. In this experiment, we used the training set to build models. We only standardized the features before applying the models.

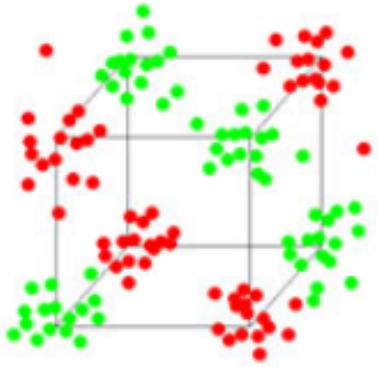


Figure 5.4 : Clusters of MADELON data set at a glance.

Table 5.1 : Number of observations of the training set, the validation set and the test set of MADELON data set.

MADELON Data Set	Positive ex.	Negative ex.	TOTAL
Training Set	1000	1000	2000
Validation Set	300	300	600
Test Set	900	900	1800
All	2200	2200	4400

5.3.2.2 Applying models

We created thirty-one different models based on the same approach with other experiments to compare the results later. Then, we tested the performance of our models 5-times using cross-validation to find average scores after 5 different holdouts.

6. RESULTS

In this chapter, we presented the results of the evaluation of different models including different parameter values on the data sets. To do this, we showed the effects of different dimensions on the performance of PCA and LDA and different kernel functions on the performance of SVMs. We calculated the differences between the performances of the models by using metric measurements such as accuracy, precision, recall, f1-score. The fit time and the score time are the other important factors to be considered in evaluating performance of model. Because in practice the time we spend in solving a problem is also very important. The time it takes to find the best version of a model should also be remembered. For this reason, we took into account the run time to compare models as well. In this way, we presented that a robust model with appropriate parameter values improves the performance of the model.

6.1 The Olivetti Faces Data Set

We showed the scores of the LDA model with different number of components applied on the data set in Table 6.1. We obtained the scores by changing number of components to choosing optimal LDA model. According our study, the LDA gives us very good scores even 2D projection on this data set. There is no difference in scores by increasing the dimension in LDA. The only thing that changes is the slight increase in the run time. Therefore it is the best choice to apply LDA model in 2 dimensions for the Olivetti faces data set.

In SVM model, we used four different kernel functions which are linear, RBF, polynomial with degree 3 and sigmoid on the data set. The objective is to find which type of kernel function that gives the best scores. As shown in Table 6.2, there is no significant difference between the linear kernel SVM and RBF kernel SVM and both have better accuracy than the other kernel functions. Nonetheless, linear kernel SVM ran faster when compared to RBF kernel SVM. The results show us that the model may give pretty bad results when the model is used by selecting the wrong kernel such as polynomial for this data set. We infer from the results that the kernel function needs to

The Olivetti Faces Data Set						
Model	Accuracy	Precision	Recall	F1-score	Fit Time(s)	Score Time(s)
2D-LDA	**0.97 (+/- 0.01)**	**0.98 (+/- 0.01)**	**0.97 (+/- 0.01)**	**0.97 (+/- 0.01)**	**0.32**	**0.02**
5D-LDA	0.97 (+/- 0.01)	0.98 (+/- 0.01)	0.97 (+/- 0.01)	0.97 (+/- 0.01)	0.42	0.03
15D-LDA	0.97 (+/- 0.01)	0.98 (+/- 0.01)	0.97 (+/- 0.01)	0.97 (+/- 0.01)	0.38	0.03

Table 6.1 : The table shows the results of the LDA model fitted the Olivetti faces data set for different dimensions. The optimal LDA model is highlighted.

The Olivetti Faces Data Set						
Model	Accuracy	Precision	Recall	F1-score	Fit Time(s)	Score Time(s)
Linear Kernel SVM	**0.94 (+/- 0.05)**	**0.96 (+/- 0.03)**	**0.94 (+/- 0.05)**	**0.93 (+/- 0.05)**	**0.61**	**0.65**
RBF Kernel SVM	0.93 (+/- 0.03)	0.95 (+/- 0.02)	0.93 (+/- 0.03)	0.93 (+/- 0.03)	0.75	0.66
Sigmoid Kernel SVM	0.87 (+/- 0.08)	0.89 (+/- 0.05)	0.87 (+/- 0.08)	0.86 (+/- 0.09)	0.54	0.70
Polynomial Kernel SVM	0.58 (+/- 0.09)	0.75 (+/- 0.13)	0.58 (+/- 0.09)	0.62 (+/- 0.11)	0.69	0.67

Table 6.2 : SVM results with difference type of kernels on the Olivetti faces data set. The optimal SVM model is highlighted.

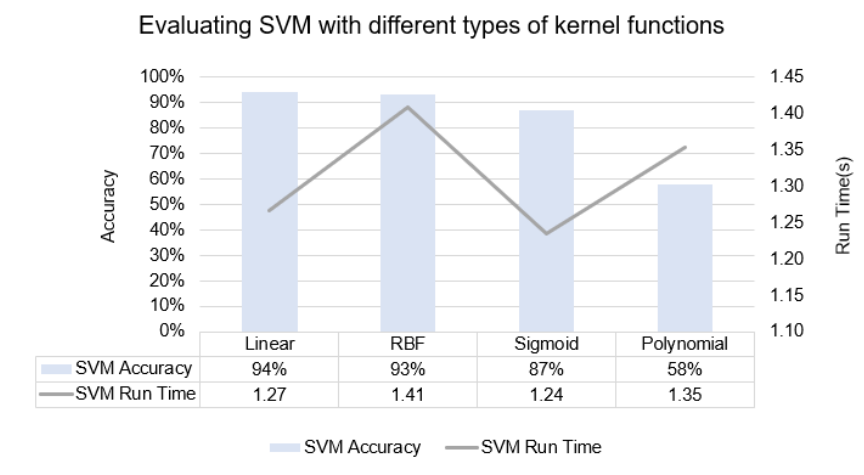


Figure 6.1 : A comparison the SVM with different types of kernel functions on the Olivetti faces data set.

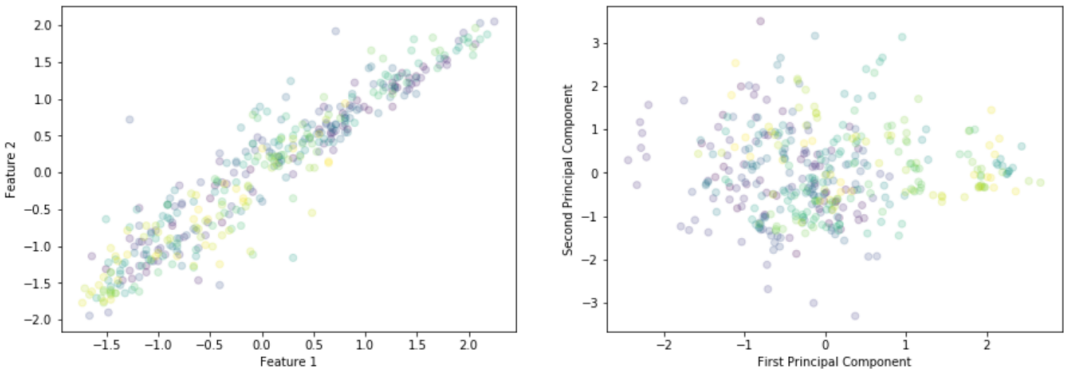


Figure 6.2 : The left figure shows the plot with two features of the original Olivetti faces data set. The right figure shows the plot of the transformed data into two-dimensional space by PCA. Thus, the data points on the diagonal have been rotated into the xy axis (i.e. principal components) that maximizes the variance.

be considered for enhancing the performance of the SVM. Figure 6.1 shows the chart of the comparison of the different types of kernel functions.

After using the SVM model, we used the SVM for face recognition on the reduced data set by PCA. Before explaining these results, we will explain and illustrate how we get results with the PCA and LDA. In Figure 6.2, right panel shows the original data points in two features, while the right panel is shows the reduced data in two dimensional space by PCA. In Figure 6.3, right panel shows the original data points in two features, while the right panel is shows the reduced data in two dimensional space by LDA. The reason why plotting the data points in two features is to try to see the relationship between the features. As we know, PCA ignores the classes whereas LDA maximizes the distance between the each class. This major difference between the two different dimensional reduction methods seen clearly when visualized the points. PCA also projects the data to the new coordinates that have the highest variances called the principal components. Thus we were able to see what the biggest variances in the face images correspond to by plotting the first and second principal components, as shown in Figure 6.4. We also plotted average face image by computing the per-feature empirical mean using 50D-PCA in Figure 6.5. The mean face showed the characteristics of the faces corresponding to the two principal components.

We applied the PCA to reduced the dimension of the data set and used the SVM for face recognition task. Thus, we reduced the run time and the high dimensional space to catch the results of the LDA. In the data set reduced to different components by PCA,

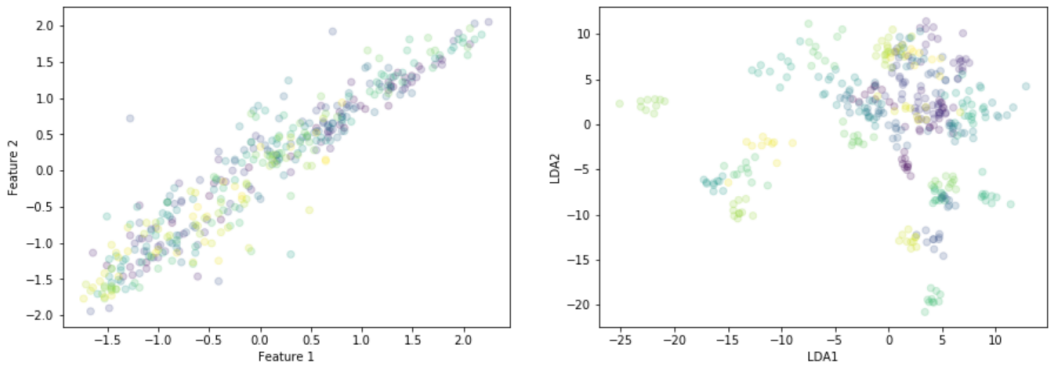


Figure 6.3 : The left figure shows the plot with two features of the original Olivetti faces data set. The right figure shows the plot of transformed data into two-dimensional space that is the directions (i.e. linear discriminants) represents the axes that maximize the separation between classes by LDA.

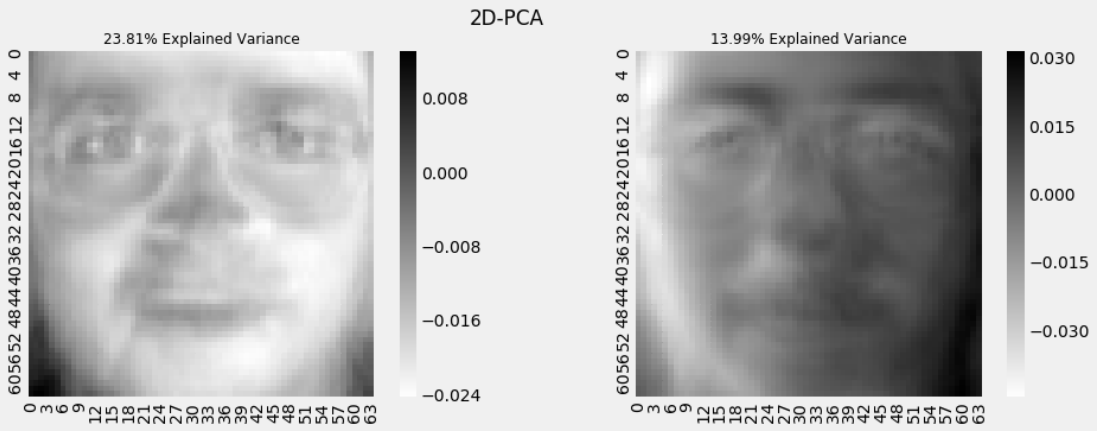


Figure 6.4 : Plot of the first and second components choosing by 2D-PCA with their explained variance. The characteristics of the faces corresponding to the two components are different from each other.

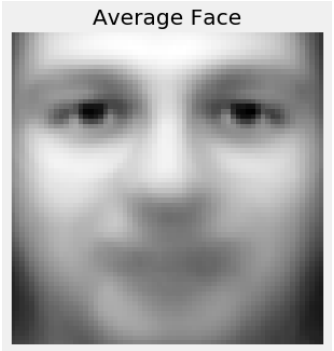


Figure 6.5 : The image shows the average face of the peoples in the Olivetti faces data set. The mean face computed by 50D-PCA.

the SVM was used with different types of kernel functions as shown in Table 6.3. The accuracy of linear kernel SVM increased from 94% to 97% by implementing 30-dimensional PCA. Moreover, the run time of the linear kernel SVM decreased from 1.27 seconds to 0.02 seconds by implementing PCA. In this way, we have saved space by reducing the number of dimensions from 4096 to 30. As shown in Table 6.4, in generally, we obtained the better results using the SVM with PCA than using only SVM. As the number of dimensions increases in PCA and the kernel function in SVM changes, the corresponding accuracy can be seen in Figure 6.6.

We have listed the best of the thirty-one different models built on the data set by comparing their scores and run times are shown in Table 6.5 and in Figure 6.7. The obtained results show that, The 2D-LDA gave better performance in accuracy even if we chose the best kernel in the SVM. 2D-LDA is also ran 3.7 times faster than linear kernel SVM that has the best kernel fitted the data set. However, when we first reduced the dimension of the data set using PCA and then classified with SVM, we got better results in run time. Linear Kernel SVM with 30D-PCA substantially faster that is 15 times than 2D-LDA. Moreover, the scores of the Linear Kernel SVM with 30D-PCA is almost as good as 2D-LDA. However, it should be noted that the 2D-LDA is better for saving space. In the process of deciding the correct model according to our results needs to be the focus on our priorities. With this approach, besides the accuracy of the model, we may choose the model that is good in terms of speed or space depends on our priorities.

6.2 Fashion-MNIST Data Set

Based on results of the LDA on the Fashion-MNIST data set in Table 6.6, we are able to use LDA model to classify this data set perfectly because both the scores are good and the run times are very short. Although our data set has $70,000$ observations, it has performed well in a very short time. Increasing the number of dimensions when using LDA did not increase the scores. As in the previous experiment, only extended runtime slightly.

The performance of SVM with four different kernel functions used is shown in Table 6.7. Figure 6.8 also shows the chart of the performance of the SVM with a different types of kernel functions. Changing kernel functions affected the

The Olivetti Faces Data Set						
Model	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Fit Time(s)**	**Score Time(s)**
Linear Kernel SVM with 2D-PCA	0.21 (+/- 0.08)	0.14 (+/- 0.07)	0.21 (+/- 0.08)	0.15 (+/- 0.07)	0.01	0.01
Linear Kernel SVM with 5D-PCA	0.64 (+/- 0.06)	0.62 (+/- 0.08)	0.64 (+/- 0.06)	0.60 (+/- 0.06)	0.01	0.01
Linear Kernel SVM with 15D-PCA	0.94 (+/- 0.06)	0.95 (+/- 0.06)	0.94 (+/- 0.06)	0.94 (+/- 0.07)	0.01	0.01
Linear Kernel SVM with 20D-PCA	0.95 (+/- 0.06)	0.95 (+/- 0.07)	0.95 (+/- 0.06)	0.94 (+/- 0.07)	0.01	0.01
Linear Kernel SVM with 30D-PCA	**0.97 (+/- 0.04)**	**0.96 (+/- 0.06)**	**0.97 (+/- 0.04)**	**0.96 (+/- 0.06)**	**0.01**	**0.01**
Linear Kernel SVM with 50D-PCA	0.97 (+/- 0.04)	0.98 (+/- 0.03)	0.97 (+/- 0.04)	0.97 (+/- 0.04)	0.02	0.01
Sigmoid Kernel SVM with 2D-PCA	0.10 (+/- 0.06)	0.05 (+/- 0.04)	0.10 (+/- 0.06)	0.06 (+/- 0.04)	0.01	0.01
Sigmoid Kernel SVM with 5D-PCA	0.48 (+/- 0.07)	0.43 (+/- 0.09)	0.48 (+/- 0.07)	0.43 (+/- 0.07)	0.01	0.01
Sigmoid Kernel SVM with 15D-PCA	0.89 (+/- 0.10)	0.89 (+/- 0.12)	0.89 (+/- 0.10)	0.88 (+/- 0.11)	0.01	0.01
Sigmoid Kernel SVM with 20D-PCA	0.92 (+/- 0.06)	0.93 (+/- 0.06)	0.92 (+/- 0.06)	0.91 (+/- 0.07)	0.01	0.01
Sigmoid Kernel SVM with 30D-PCA	**0.95 (+/- 0.05)**	**0.95 (+/- 0.06)**	**0.95 (+/- 0.05)**	**0.94 (+/- 0.06)**	**0.02**	**0.01**
Sigmoid Kernel SVM with 50D-PCA	0.97 (+/- 0.05)	0.97 (+/- 0.04)	0.97 (+/- 0.05)	0.96 (+/- 0.05)	0.02	0.01
RBF Kernel SVM with 2D-PCA	0.21 (+/- 0.09)	0.13 (+/- 0.07)	0.21 (+/- 0.09)	0.15 (+/- 0.07)	0.01	0.01
RBF Kernel SVM with 5D-PCA	0.61 (+/- 0.12)	0.59 (+/- 0.16)	0.61 (+/- 0.12)	0.57 (+/- 0.13)	0.01	0.01
RBF Kernel SVM with 15D-PCA	0.92 (+/- 0.08)	0.92 (+/- 0.09)	0.92 (+/- 0.08)	0.91 (+/- 0.09)	0.02	0.01
RBF Kernel SVM with 20D-PCA	**0.94 (+/- 0.05)**	**0.94 (+/- 0.08)**	**0.94 (+/- 0.05)**	**0.94 (+/- 0.06)**	**0.02**	**0.01**
RBF Kernel SVM with 30D-PCA	0.97 (+/- 0.05)	0.96 (+/- 0.08)	0.97 (+/- 0.05)	0.96 (+/- 0.07)	0.02	0.01
RBF Kernel SVM with 50D-PCA	0.97 (+/- 0.07)	0.98 (+/- 0.05)	0.97 (+/- 0.07)	0.97 (+/- 0.07)	0.03	0.01
Polynomial Kernel SVM with 2D-PCA	0.24 (+/- 0.10)	0.20 (+/- 0.06)	0.24 (+/- 0.10)	0.19 (+/- 0.07)	0.01	0.01
Polynomial Kernel SVM with 5D-PCA	0.50 (+/- 0.08)	0.54 (+/- 0.15)	0.50 (+/- 0.08)	0.49 (+/- 0.11)	0.01	0.01
Polynomial Kernel SVM with 15D-PCA	0.70 (+/- 0.14)	0.80 (+/- 0.17)	0.70 (+/- 0.14)	0.72 (+/- 0.15)	0.01	0.01
Polynomial Kernel SVM with 20D-PCA	**0.76 (+/- 0.15)**	**0.86 (+/- 0.08)**	**0.76 (+/- 0.15)**	**0.77 (+/- 0.12)**	**0.01**	**0.01**
Polynomial Kernel SVM with 30D-PCA	0.66 (+/- 0.10)	0.78 (+/- 0.09)	0.66 (+/- 0.10)	0.68 (+/- 0.10)	0.02	0.01
Polynomial Kernel SVM with 50D-PCA	0.56 (+/- 0.11)	0.69 (+/- 0.14)	0.56 (+/- 0.11)	0.59 (+/- 0.12)	0.02	0.01

Table 6.3 : SVM with PCA results on the Olivetti faces data set. The best performance of each kind of model has been highlighted in the table.

The Olivetti Faces Data Set												
SVM Kernel	**2D-PCA**		**5D-PCA**		**15D-PCA**		**20D-PCA**		**30D-PCA**		**50D-PCA**	
	Accuracy	**Run Time(s)**	**Accuracy**	**Run Time(s)**	**Accuracy**	**Run Time(s)**	**Accuracy**	**Run Time(s)**	**Accuracy**	**Run Time(s)**	**Accuracy**	**Run Time(s)**
Linear	0.21 (+/- 0.08)	0.02	0.64 (+/- 0.06)	0.02	0.94 (+/- 0.06)	0.02	0.95 (+/- 0.06)	0.02	0.97 (+/- 0.04)	0.02	0.97 (+/- 0.04)	0.03
RBF	0.21 (+/- 0.09)	0.02	0.61 (+/- 0.12)	0.02	0.92 (+/- 0.08)	0.03	0.94 (+/- 0.05)	0.03	0.97 (+/- 0.05)	0.03	0.97 (+/- 0.07)	0.04
Polynomial	0.24 (+/- 0.10)	0.02	0.50 (+/- 0.08)	0.02	0.70 (+/- 0.14)	0.02	0.76 (+/- 0.15)	0.02	0.66 (+/- 0.10)	0.03	0.56 (+/- 0.11)	0.03
Sigmoid	0.10 (+/- 0.06)	0.02	0.48 (+/- 0.07)	0.02	0.89 (+/- 0.10)	0.02	0.92 (+/- 0.06)	0.02	0.95 (+/- 0.05)	0.03	0.97 (+/- 0.05)	0.04

Table 6.4 : A comparison of the performance of the SVM with different types of kernel functions on the reduced Olivetti faces data set by PCA.

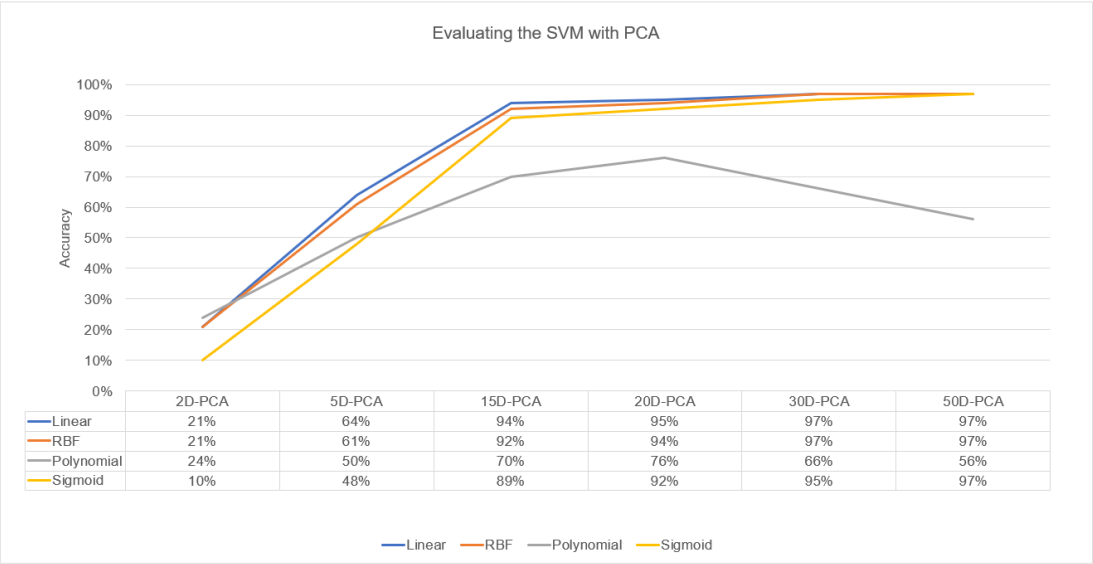


Figure 6.6 : A comparison of the accuracy of the SVM with different types of kernels on the reduced data by PCA on the Olivetti faces data set.

The Olivetti Faces Data Set						
Model	**Accuracy**	**Precision**	**Recall**	**f1-score**	**Fit Time(s)**	**Score Time(s)**
2D-LDA	0.97 (+/- 0.01)	0.98 (+/- 0.01)	0.97 (+/- 0.01)	0.97 (+/- 0.01)	0.32	0.02
Linear Kernel SVM with 30D-PCA	0.97 (+/- 0.04)	0.96 (+/- 0.06)	0.97 (+/- 0.04)	0.96 (+/- 0.06)	0.01	0.01
Sigmoid Kernel SVM with 30D-PCA	0.95 (+/- 0.05)	0.95 (+/- 0.06)	0.95 (+/- 0.05)	0.94 (+/- 0.06)	0.02	0.01
RBF Kernel SVM with 20D-PCA	0.94 (+/- 0.05)	0.94 (+/- 0.08)	0.94 (+/- 0.05)	0.94 (+/- 0.06)	0.02	0.01
Linear Kernel SVM	0.94 (+/- 0.05)	0.96 (+/- 0.03)	0.94 (+/- 0.05)	0.93 (+/- 0.05)	0.61	0.65
Polynomial Kernel SVM with 20D-PCA	0.76 (+/- 0.15)	0.86 (+/- 0.08)	0.76 (+/- 0.15)	0.77 (+/- 0.12)	0.01	0.01

Table 6.5 : Table of the models which have the best performance in their methods.

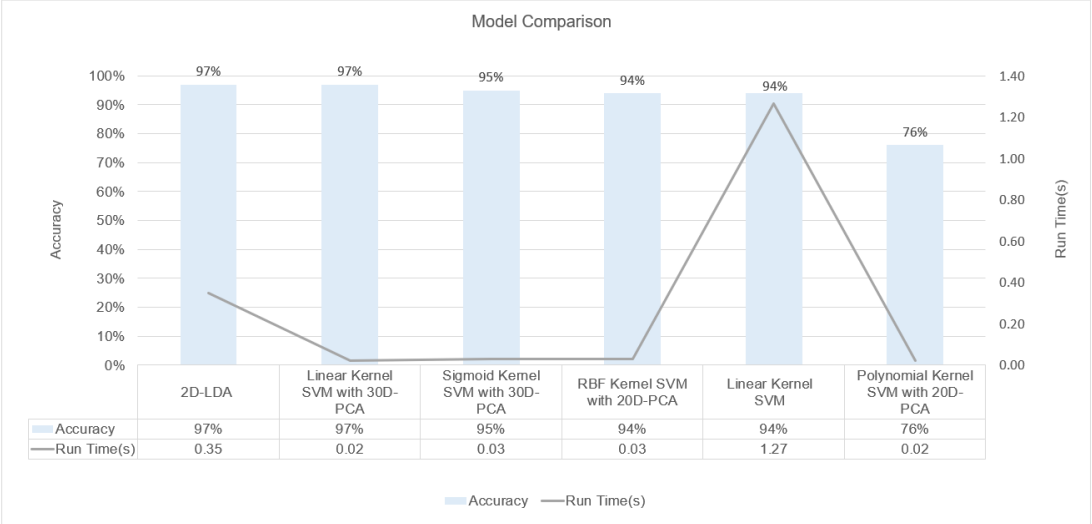


Figure 6.7 : Accuracy and run time comparison for the best models on the Olivetti faces data set.

Fashion-MNIST Data Set						
Model	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Fit Time(s)**	**Score Time(s)**
2D-LDA	**0.82 (+/- 0.00)**	**0.82 (+/- 0.00)**	**0.82 (+/- 0.00)**	**0.82 (+/- 0.00)**	**6.60**	**0.16**
5D-LDA	0.82 (+/- 0.00)	0.82 (+/- 0.00)	0.82 (+/- 0.00)	0.82 (+/- 0.00)	6.59	0.16
15D-LDA	0.82 (+/- 0.00)	0.82 (+/- 0.00)	0.82 (+/- 0.00)	0.82 (+/- 0.00)	7.03	0.16

Table 6.6 : The table shows the results of the LDA model fitted the Fashion-MNIST data set for different dimensions. The optimal LDA model is highlighted.

Fashion-MNIST Data Set						
Model	Accuracy	Precision	Recall	F1-score	Fit Time(s)	Score Time(s)
RBF Kernel SVM	**0.89 (+/- 0.00)**	**0.89 (+/- 0.00)**	**0.89 (+/- 0.00)**	**0.89 (+/- 0.00)**	**404.94**	**1452.10**
Linear Kernel SVM	0.84 (+/- 0.00)	0.84 (+/- 0.00)	0.84 (+/- 0.00)	0.84 (+/- 0.00)	750.13	1176.58
Polynomial Kernel SVM	0.88 (+/- 0.00)	0.88 (+/- 0.00)	0.88 (+/- 0.00)	0.88 (+/- 0.00)	501.70	1980.14
Sigmoid Kernel SVM	0.72 (+/- 0.01)	0.72 (+/- 0.01)	0.72 (+/- 0.01)	0.71 (+/- 0.01)	313.61	1443.42

Table 6.7 : SVM results with difference type of kernels on the Fashion-MNIST data set. The optimal SVM model is highlighted.

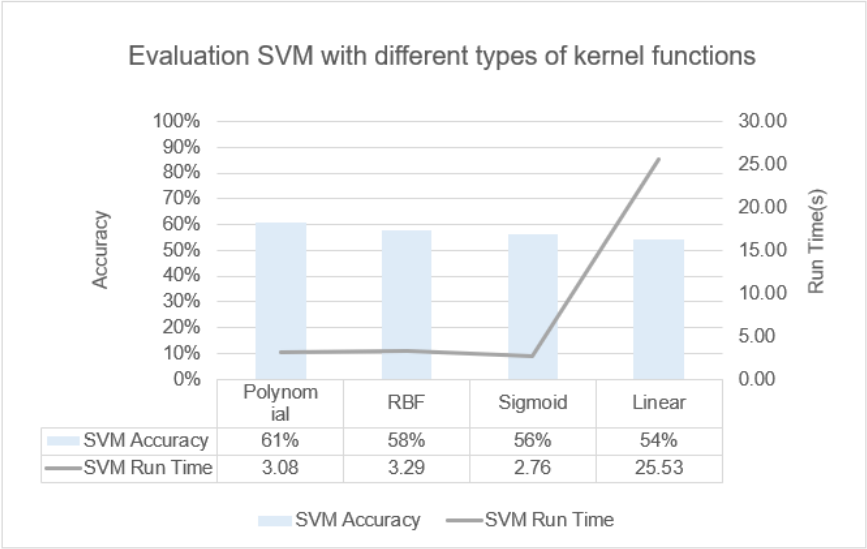


Figure 6.8 : A comparison the SVM with different types of kernel functions on the Fashion-MNIST data set.

performance of the SVM on the Fashion-MNIST data set. The performance of the RBF kernel, which is better than other kernels, still did not exceed the performance of the 2D-LDA. Because although the scores are good, the run time was 275 times the 2D-LDA.

The results obtained by using RBF kernel SVM reduced the run time from approximately thirty minutes to one minute using RBF Kernel SVM with 15D-PCA, while accuracy decreased slightly from 89% to 85% as shown in Table 6.8 and in Table 6.9. The experimental results showed that when SVM was combined with PCA, instead of using SVM alone, it saved us time to classify images, but LDA was not as good as run-time performance. Figure 6.9 shows the chart of the performance PCA combined SVM.

We would like to emphasize the difference of data points distribution between 2D-PCA and 2D-LDA by showing Figure 6.10 and Figure 6.11. We plotted these data points distributions with the same reason explained in the Olivetti faces data set. We showed

Fashion-MNIST Data Set						
Model	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Fit Time(s)**	**Score Time(s)**
RBF Kernel SVM with 2D-PCA	0.55 (+/- 0.01)	0.56 (+/- 0.01)	0.55 (+/- 0.01)	0.54 (+/- 0.01)	35.19	57.15
RBF Kernel SVM with 5D-PCA	0.75 (+/- 0.01)	0.74 (+/- 0.01)	0.75 (+/- 0.01)	0.74 (+/- 0.01)	21.19	40.89
RBF Kernel SVM with 15D-PCA	**0.85 (+/- 0.00)**	**0.85 (+/- 0.00)**	**0.85 (+/- 0.00)**	**0.85 (+/- 0.00)**	**25.04**	**39.66**
RBF Kernel SVM with 20D-PCA	0.86 (+/- 0.00)	0.86 (+/- 0.00)	0.86 (+/- 0.00)	0.86 (+/- 0.00)	28.84	41.59
RBF Kernel SVM with 30D-PCA	0.87 (+/- 0.00)	0.87 (+/- 0.00)	0.87 (+/- 0.00)	0.87 (+/- 0.00)	49.84	57.58
RBF Kernel SVM with 50D-PCA	0.88 (+/- 0.00)	0.88 (+/- 0.00)	0.88 (+/- 0.00)	0.88 (+/- 0.00)	69.74	72.29
Polynomial Kernel SVM with 2D-PCA	0.51 (+/- 0.01)	0.53 (+/- 0.01)	0.51 (+/- 0.01)	0.49 (+/- 0.00)	39.75	42.59
Polynomial Kernel SVM with 5D-PCA	0.74 (+/- 0.01)	0.74 (+/- 0.01)	0.74 (+/- 0.01)	0.74 (+/- 0.01)	19.97	30.25
Polynomial Kernel SVM with 15D-PCA	**0.84 (+/- 0.00)**	**0.84 (+/- 0.00)**	**0.84 (+/- 0.00)**	**0.84 (+/- 0.00)**	**27.14**	**30.77**
Polynomial Kernel SVM with 20D-PCA	0.85 (+/- 0.00)	0.85 (+/- 0.00)	0.85 (+/- 0.00)	0.85 (+/- 0.00)	31.50	32.66
Polynomial Kernel SVM with 30D-PCA	0.86 (+/- 0.00)	0.86 (+/- 0.00)	0.86 (+/- 0.00)	0.86 (+/- 0.00)	60.19	49.65
Polynomial Kernel SVM with 50D-PCA	0.86 (+/- 0.00)	0.87 (+/- 0.00)	0.86 (+/- 0.00)	0.87 (+/- 0.00)	960.65	67.32
Linear Kernel SVM with 2D-PCA	0.52 (+/- 0.01)	0.51 (+/- 0.01)	0.52 (+/- 0.01)	0.51 (+/- 0.01)	26.36	40.22
Linear Kernel SVM with 5D-PCA	0.72 (+/- 0.00)	0.71 (+/- 0.00)	0.72 (+/- 0.00)	0.71 (+/- 0.00)	21.21	28.37
Linear Kernel SVM with 15D-PCA	**0.81 (+/- 0.01)**	**0.80 (+/- 0.01)**	**0.81 (+/- 0.01)**	**0.80 (+/- 0.01)**	**31.47**	**29.15**
Linear Kernel SVM with 20D-PCA	0.82 (+/- 0.00)	0.82 (+/- 0.00)	0.82 (+/- 0.00)	0.82 (+/- 0.00)	39.35	32.22
Linear Kernel SVM with 30D-PCA	0.83 (+/- 0.01)	0.83 (+/- 0.01)	0.83 (+/- 0.01)	0.83 (+/- 0.01)	58.69	42.07
Linear Kernel SVM with 50D-PCA	0.84 (+/- 0.01)	0.84 (+/- 0.01)	0.84 (+/- 0.01)	0.84 (+/- 0.01)	92.16	55.13
Sigmoid Kernel SVM with 2D-PCA	0.32 (+/- 0.01)	0.33 (+/- 0.02)	0.32 (+/- 0.01)	0.31 (+/- 0.02)	47.91	67.72
Sigmoid Kernel SVM with 5D-PCA	0.48 (+/- 0.01)	0.49 (+/- 0.02)	0.48 (+/- 0.01)	0.47 (+/- 0.02)	29.58	51.37
Sigmoid Kernel SVM with 15D-PCA	0.62 (+/- 0.02)	0.62 (+/- 0.01)	0.62 (+/- 0.02)	0.61 (+/- 0.01)	31.59	52.98
Sigmoid Kernel SVM with 20D-PCA	0.64 (+/- 0.01)	0.64 (+/- 0.01)	0.64 (+/- 0.01)	0.64 (+/- 0.01)	37.98	56.79
Sigmoid Kernel SVM with 30D-PCA	**0.68 (+/- 0.00)**	**0.68 (+/- 0.00)**	**0.68 (+/- 0.00)**	**0.68 (+/- 0.00)**	**48.85**	**69.28**
Sigmoid Kernel SVM with 50D-PCA	0.70 (+/- 0.01)	0.70 (+/- 0.01)	0.70 (+/- 0.01)	0.70 (+/- 0.01)	68.60	95.43

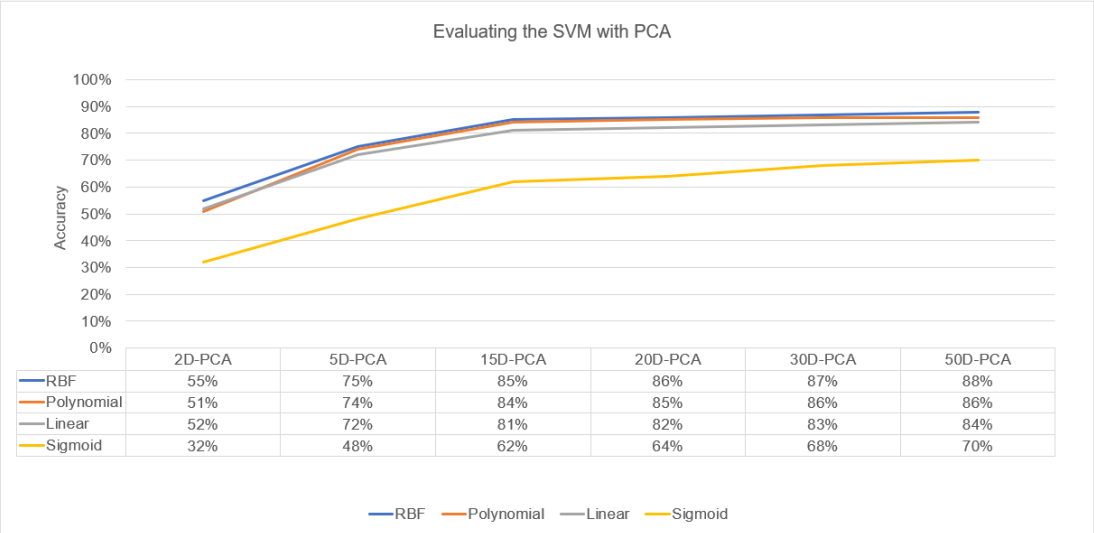
Table 6.8 : SVM with PCA results on the Fashion-MNIST data set. The best performance of each kind of model has been highlighted in the table.

Fashion-MNIST Data Set												
SVM Kernel	**2D-PCA**		**5D-PCA**		**15D-PCA**		**20D-PCA**		**30D-PCA**		**50D-PCA**	
	Accuracy	**Run Time(s)**	**Accuracy**	**Run Time(s)**	**Accuracy**	**Run Time(s)**	**Accuracy**	**Run Time(s)**	**Accuracy**	**Run Time(s)**	**Accuracy**	**Run Time(s)**
RBF	0.55 (+/- 0.01)	92.34	0.75 (+/- 0.01)	62.09	0.85 (+/- 0.00)	64.70	0.86 (+/- 0.00)	70.43	0.87 (+/- 0.00)	107.42	0.88 (+/- 0.00)	142.03
Polynomial	0.51 (+/- 0.01)	82.33	0.74 (+/- 0.01)	50.21	0.84 (+/- 0.00)	57.91	0.85 (+/- 0.00)	64.15	0.86 (+/- 0.00)	109.84	0.86 (+/- 0.00)	1027.97
Linear	0.52 (+/- 0.01)	66.58	0.72 (+/- 0.00)	49.58	0.81 (+/- 0.01)	60.62	0.82 (+/- 0.00)	71.57	0.83 (+/- 0.01)	100.75	0.84 (+/- 0.01)	147.29
Sigmoid	0.32 (+/- 0.01)	115.62	0.48 (+/- 0.01)	80.95	0.62 (+/- 0.02)	84.57	0.64 (+/- 0.01)	94.77	0.68 (+/- 0.00)	118.13	0.70 (+/- 0.01)	164.03

Table 6.9 : A comparison of the performance of the SVM with different types of kernel functions on the reduced Fashion-MNIST data set by PCA.

the biggest variances in the fashion images correspond to by plotting the first and second principal components, as shown in Figure 6.12. We also plotted average fashion image by computing the per-feature empirical mean using 50D-PCA in Figure 6.13. This average image showed the silhouettes of both top clothes and trousers.

Table 6.10 shows the set of best models has been chosen on the Fashion-MNIST data set. In Figure 6.14, we also show the accuracy and the run time comparison for the best models. The 2D-LDA performed much better on this data set than any other model. This model achieved with an accuracy of 82% in 6.76 seconds. The advantages of the 2D-LDA are that it is fast, classifies images with good accuracy and gives more space than other models. The highest accuracy was 89% achieved by the SVM with RBF kernel, but the run time was 275 times the 2D-LDA. We then applied the SVM with RBF kernel on the reduced data from 784 dimensions to 15 dimensions by PCA. PCA-SVM combination ran faster 28 times with an accuracy of 85% than only using



	2D-PCA	5D-PCA	15D-PCA	20D-PCA	30D-PCA	50D-PCA
RBF	55%	75%	85%	86%	87%	88%
Polynomial	51%	74%	84%	85%	86%	86%
Linear	52%	72%	81%	82%	83%	84%
Sigmoid	32%	48%	62%	64%	68%	70%

Figure 6.9 : A comparison of the accuracy of the SVM with different types of kernels on the reduced data by PCA on the Fashion-MNIST data set.

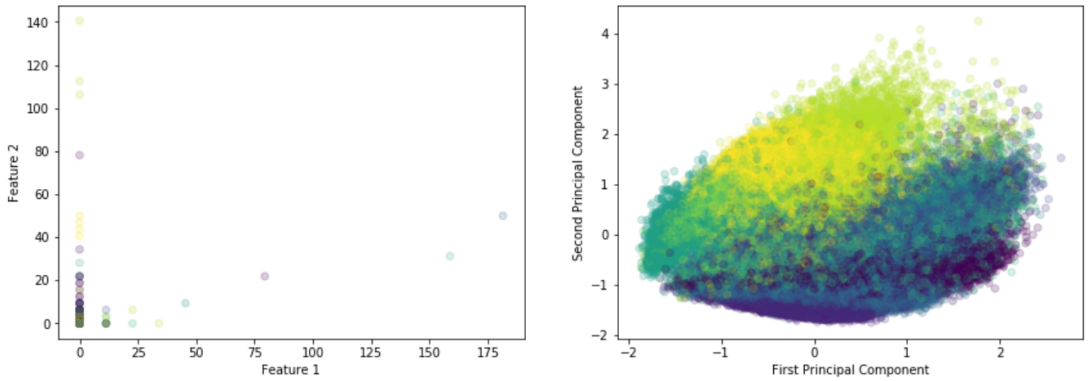


Figure 6.10 : The left figure shows the plot with two features of the original Fashion-MNIST data set. The right figure shows the plot of the transformed data into two-dimensional space by PCA. Thus, the data points on the diagonal have been rotated into the xy axis (i.e. principal components) that maximizes the variance.

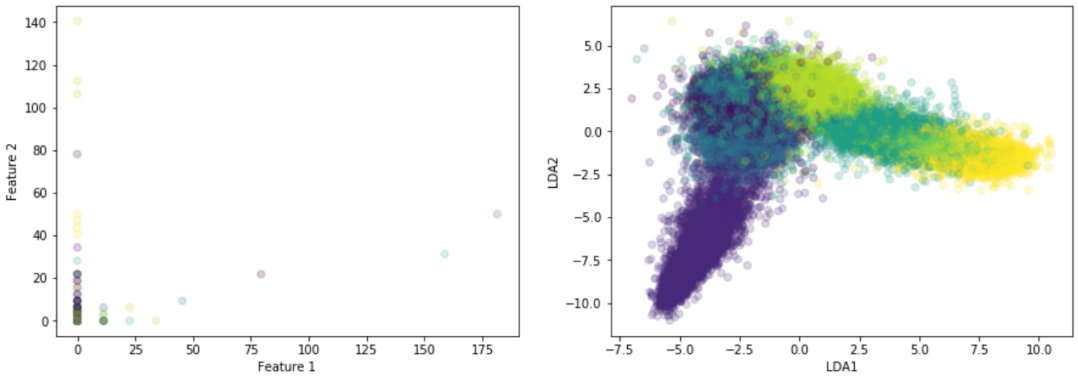


Figure 6.11 : The left figure shows the plot with two features of the original Fashion-MNIST data set. The right figure shows the plot of transformed data into two-dimensional space that is the directions represents the axes that maximize the separation between classes by LDA.

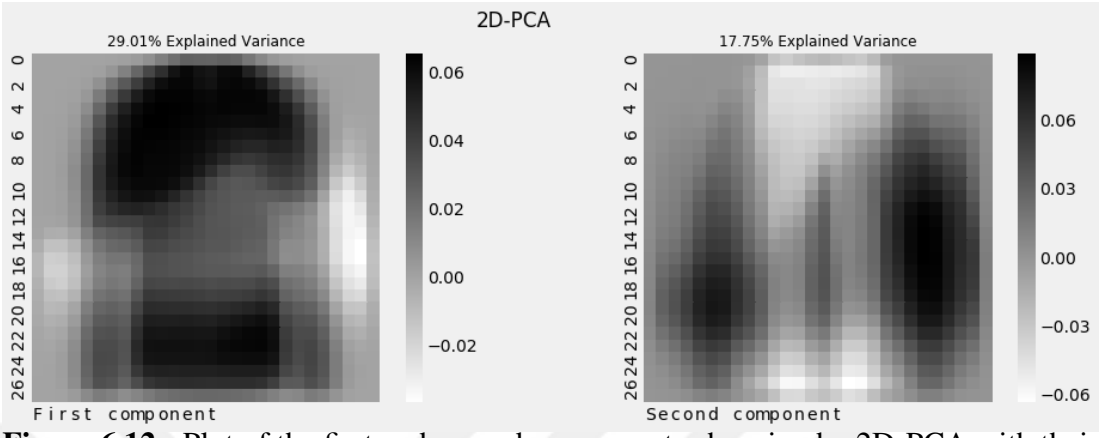


Figure 6.12 : Plot of the first and second components choosing by 2D-PCA with their explained variance. The first component looks like such a tshort/top or a shoe and the second component looks like kind of a trouser or a tshort/top.

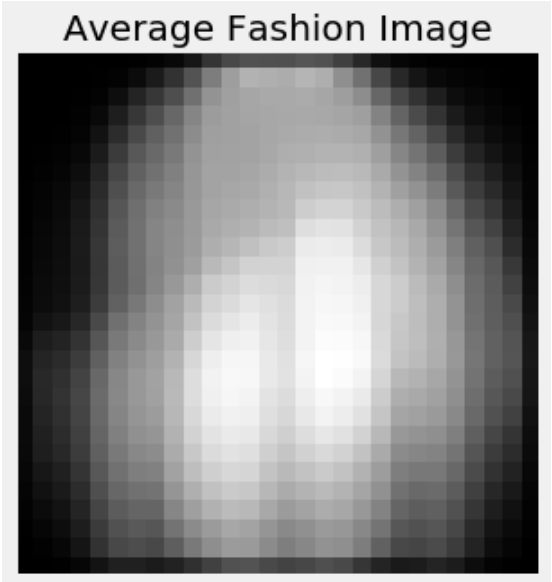


Figure 6.13 : The image shows the average fashion item of the all images in the Fashion-MNIST data set. The mean image computed by 50D-PCA.

Fashion-MNIST Data Set						
Model	Accuracy	Precision	Recall	f1-score	Fit Time(s)	Score Time(s)
2D-LDA	0.82 (+/- 0.00)	0.82 (+/- 0.00)	0.82 (+/- 0.00)	0.82 (+/- 0.00)	6.60	0.16
RBF Kernel SVM with 15D-PCA	0.85 (+/- 0.00)	0.85 (+/- 0.00)	0.85 (+/- 0.00)	0.85 (+/- 0.00)	25.04	39.66
Polynomial Kernel SVM with 15D-PCA	0.84 (+/- 0.00)	0.84 (+/- 0.00)	0.84 (+/- 0.00)	0.84 (+/- 0.00)	27.14	30.77
Linear Kernel SVM with 15D-PCA	0.81 (+/- 0.01)	0.80 (+/- 0.01)	0.81 (+/- 0.01)	0.80 (+/- 0.01)	31.47	29.15
Sigmoid Kernel SVM with 30D-PCA	0.68 (+/- 0.00)	0.68 (+/- 0.00)	0.68 (+/- 0.00)	0.68 (+/- 0.00)	48.85	69.28
RBF Kernel SVM	0.89 (+/- 0.00)	0.89 (+/- 0.00)	0.89 (+/- 0.00)	0.89 (+/- 0.00)	404.94	1452.10

Table 6.10 : Table of the models which have the best performance in their methods on the Fashion-MNIST data set.

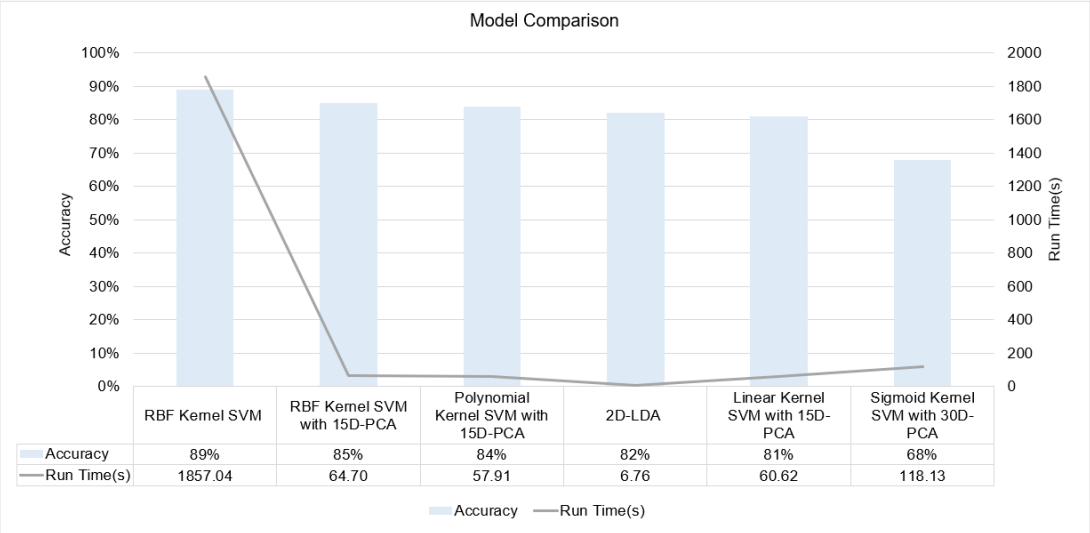


Figure 6.14 : Accuracy and run time comparison for the best models on the Fashion-MNIST data set.

SVM. Yet, the 2D-LDA still approximately 10 times faster and save approximately 7 times more space.

6.3 MADELON Data Set

According to our results, the LDA did not classify the MADELON data set which is highly non-linear and multivariate data set. Increasing the number of dimensions in the LDA did not increase the scores as shown in Table 6.11. The reason for the poor performance is that LDA works as a linear classifier and the data set is highly non-linear. Visualizing the data points in 3D-hypercube can be seen in Figure 6.15 using three features in the data set.

When it comes to the examining the SVM, we see that it works better than the LDA. However, when we choose the linear kernel, the result was still very poor. Furthermore, the linear kernel worked much slower than other kernels due to the same reason is that a highly non-linear data set. Degree 3 polynomial kernel gave very good performance by

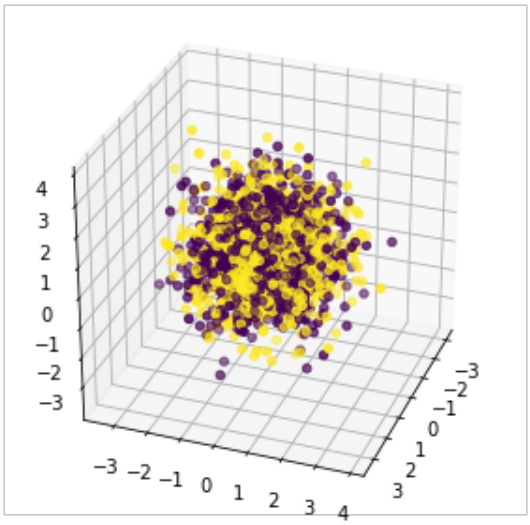


Figure 6.15 : Visualizing the data points in the 3D-hypercube.

MADELON Data Set						
Model	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Fit Time(s)**	**Score Time(s)**
2D-LDA	**0.55 (+/- 0.05)**	**0.55 (+/- 0.05)**	**0.55 (+/- 0.05)**	**0.55 (+/- 0.05)**	**0.21**	**0.00**
5D-LDA	0.55 (+/- 0.05)	0.55 (+/- 0.05)	0.55 (+/- 0.05)	0.55 (+/- 0.05)	0.24	0.01
15D-LDA	0.55 (+/- 0.05)	0.55 (+/- 0.05)	0.55 (+/- 0.05)	0.55 (+/- 0.05)	0.24	0.01

Table 6.11 : The table shows the results of the LDA model fitted the MADELON data set for different dimensions. The optimal LDA model is highlighted.

providing non-linear decision boundary with kernel trick. We came to the conclusion that the correct kernel selection is associated with the characteristics of the data. For this reason, we need to investigate the effect of the different kernels on the results of SVM. The obtained results shown by Table 6.12 and Figure 6.16.

Combined SVM and PCA was used to classify the data points as $+1$ or -1. In Table 6.13 shows the results obtained using the SVM with PCA. In Table 6.14, we summarized these results. The results show that SVM performs better on the reduced data into the 5-dimensional space by PCA. For instance, RBF kernel SVM gave an accuracy of %58 in 3,29 seconds, and RBF kernel SVM with 5D-PCA performed

MADELON Data Set						
Model	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Fit Time(s)**	**Score Time(s)**
Polynomial Kernel SVM	**0.61 (+/- 0.04)**	**0.61 (+/- 0.05)**	**0.61 (+/- 0.04)**	**0.60 (+/- 0.03)**	**1.61**	**1.48**
RBF Kernel SVM	0.58 (+/- 0.05)	0.59 (+/- 0.05)	0.58 (+/- 0.05)	0.58 (+/- 0.05)	1.69	1.60
Sigmoid Kernel SVM	0.56 (+/- 0.05)	0.56 (+/- 0.05)	0.56 (+/- 0.05)	0.56 (+/- 0.05)	1.50	1.25
Linear Kernel SVM	0.54 (+/- 0.04)	0.54 (+/- 0.04)	0.54 (+/- 0.04)	0.54 (+/- 0.04)	24.56	0.97

Table 6.12 : SVM results with difference type of kernels on the MADELON Data Set. The optimal SVM model is highlighted.

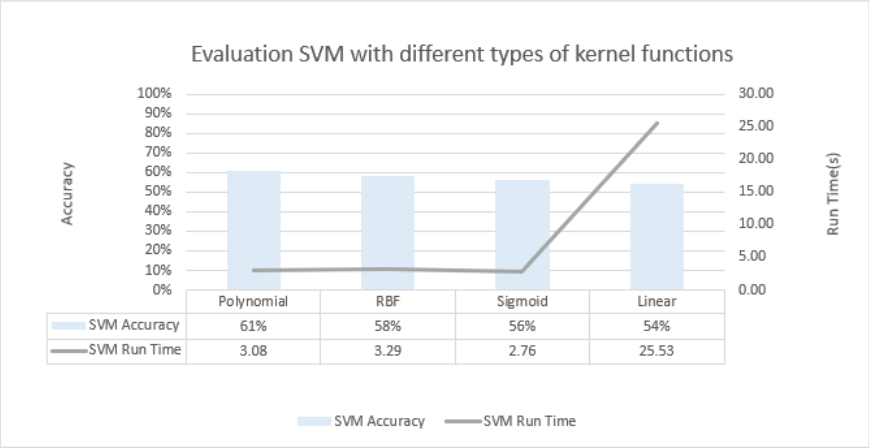


Figure 6.16 : A comparison the SVM with different types of kernel functions on the MADELON data set.

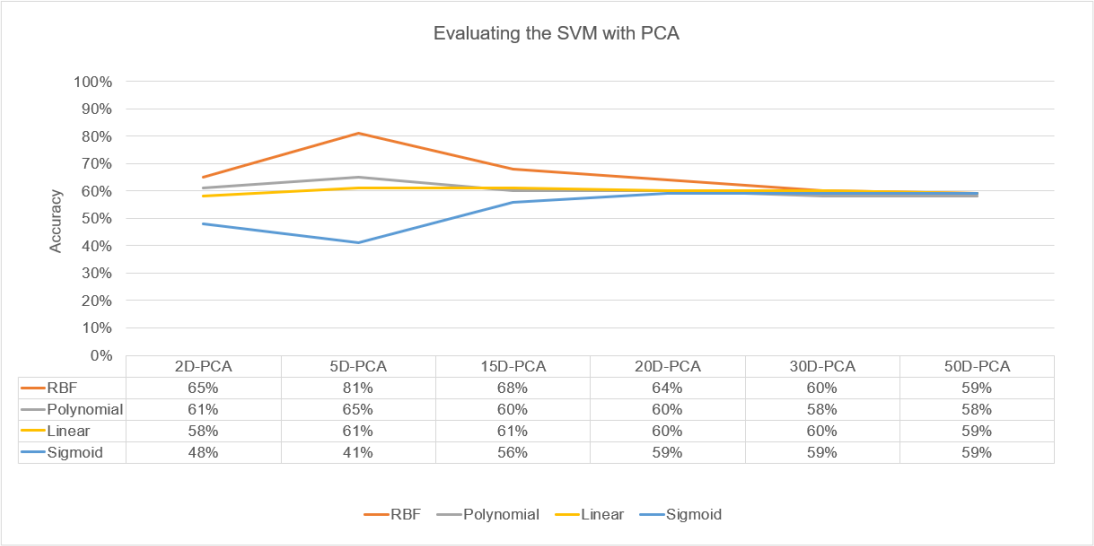


Figure 6.17 : A comparison of the accuracy of the SVM with different types of kernels on the reduced data by PCA on the MADELON data set.

better result with an accuracy of %58 in 0.08 seconds than using SVM only. The sigmoid kernel did not fit the data, because it did not give consistent results. RBF Kernel SVM with 5D-PCA gives excellent performance, while others such as Linear Kernel SVM with 5D-PCA fail to classify the data set. The performance comparison for the different types of kernels on the reduced data with different dimensions using the PCA is shown in Figure 6.17.

Table 6.15 is the list of the models having the best performance from 38 different models built on the MADELON data set. Figure 6.18 also shows the accuracy and run time comparison for the best models. From these results we see that, RBF Kernel SVM with 5D-PCA gave good results in terms of run time and speed compared to other models. The reason of the best performance in this data set, the kernel trick enabled

MADELON Data Set						
Model	Accuracy	Precision	Recall	F1-score	Fit Time(s)	Score Time(s)
RBF Kernel SVM with 2D-PCA	0.65 (+/- 0.02)	0.67 (+/- 0.02)	0.65 (+/- 0.02)	0.64 (+/- 0.03)	0.05	0.03
RBF Kernel SVM with 5D-PCA	**0.81 (+/- 0.02)**	**0.81 (+/- 0.02)**	**0.81 (+/- 0.02)**	**0.81 (+/- 0.02)**	**0.05**	**0.03**
RBF Kernel SVM with 15D-PCA	0.68 (+/- 0.04)	0.68 (+/- 0.04)	0.68 (+/- 0.04)	0.68 (+/- 0.04)	0.08	0.06
RBF Kernel SVM with 20D-PCA	0.64 (+/- 0.05)	0.64 (+/- 0.04)	0.64 (+/- 0.05)	0.64 (+/- 0.05)	0.10	0.07
RBF Kernel SVM with 30D-PCA	0.60 (+/- 0.03)	0.60 (+/- 0.03)	0.60 (+/- 0.03)	0.60 (+/- 0.03)	0.12	0.10
RBF Kernel SVM with 50D-PCA	0.59 (+/- 0.04)	0.59 (+/- 0.04)	0.59 (+/- 0.04)	0.59 (+/- 0.04)	0.18	0.15
Polynomial Kernel SVM with 2D-PCA	0.61 (+/- 0.04)	0.64 (+/- 0.05)	0.61 (+/- 0.04)	0.59 (+/- 0.04)	0.06	0.02
Polynomial Kernel SVM with 5D-PCA	**0.65 (+/- 0.03)**	**0.65 (+/- 0.03)**	**0.65 (+/- 0.03)**	**0.65 (+/- 0.03)**	**0.05**	**0.02**
Polynomial Kernel SVM with 15D-PCA	0.60 (+/- 0.05)	0.60 (+/- 0.05)	0.60 (+/- 0.05)	0.60 (+/- 0.05)	0.07	0.04
Polynomial Kernel SVM with 20D-PCA	0.60 (+/- 0.06)	0.60 (+/- 0.06	0.60 (+/- 0.06)	0.60 (+/- 0.05)	0.09	0.05
Polynomial Kernel SVM with 30D-PCA	0.58 (+/- 0.03)	0.58 (+/- 0.03)	0.58 (+/- 0.03)	0.58 (+/- 0.03)	0.11	0.07
Polynomial Kernel SVM with 50D-PCA	0.58 (+/- 0.05)	0.58 (+/- 0.05)	0.58 (+/- 0.05)	0.58 (+/- 0.05)	0.16	0.13
Linear Kernel SVM with 2D-PCA	0.58 (+/- 0.03)	0.58 (+/- 0.03)	0.58 (+/- 0.03)	0.58 (+/- 0.03)	0.04	0.02
Linear Kernel SVM with 5D-PCA	**0.61 (+/- 0.04)**	**0.61 (+/- 0.04)**	**0.61 (+/- 0.04)**	**0.61 (+/- 0.04)**	**0.05**	**0.02**
Linear Kernel SVM with 15D-PCA	0.61 (+/- 0.05)	0.61 (+/- 0.05)	0.61 (+/- 0.05)	0.61 (+/- 0.05)	0.12	0.03
Linear Kernel SVM with 20D-PCA	0.60 (+/- 0.04)	0.60 (+/- 0.04)	0.60 (+/- 0.04)	0.60 (+/- 0.04)	0.15	0.04
Linear Kernel SVM with 30D-PCA	0.60 (+/- 0.04)	0.60 (+/- 0.04)	0.60 (+/- 0.04)	0.60 (+/- 0.04)	0.22	0.06
Linear Kernel SVM with 50D-PCA	0.59 (+/- 0.03)	0.59 (+/- 0.03)	0.59 (+/- 0.03)	0.59 (+/- 0.03)	0.38	0.08
Sigmoid Kernel SVM with 2D-PCA	0.48 (+/- 0.04)	0.48 (+/- 0.04)	0.48 (+/- 0.04)	0.48 (+/- 0.04)	0.05	0.04
Sigmoid Kernel SVM with 5D-PCA	0.41 (+/- 0.03)	0.41 (+/- 0.03)	0.41 (+/- 0.03)	0.41 (+/- 0.03)	0.05	0.04
Sigmoid Kernel SVM with 15D-PCA	0.56 (+/- 0.06)	0.56 (+/- 0.06)	0.56 (+/- 0.06)	0.56 (+/- 0.06)	0.09	0.07
Sigmoid Kernel SVM with 20D-PCA	**0.59 (+/- 0.03)**	**0.59 (+/- 0.03)**	**0.59 (+/- 0.03)**	**0.59 (+/- 0.03)**	**0.11**	**0.08**
Sigmoid Kernel SVM with 30D-PCA	0.59 (+/- 0.05)	0.59 (+/- 0.05)	0.59 (+/- 0.05)	0.59 (+/- 0.04)	0.12	0.10
Sigmoid Kernel SVM with 50D-PCA	0.59 (+/- 0.05)	0.59 (+/- 0.05)	0.59 (+/- 0.05)	0.59 (+/- 0.05)	0.19	0.15

Table 6.13 : SVM with PCA results on the MADELON data set. The best performance of each kind of model has been highlighted in the table.

MADELON Data Set												
SVM Kernel	2D-PCA		5D-PCA		15D-PCA		20D-PCA		30D-PCA		50D-PCA	
	Accuracy	Run Time(s)	Accuracy	Run Time(s)	Accuracy	Run Time(s)	Accuracy	Run Time(s)	Accuracy	Run Time(s)	Accuracy	Run Time(s)
RBF	0.65 (+/- 0.02)	0.08	0.81 (+/- 0.02)	0.08	0.68 (+/- 0.04)	0.14	0.64 (+/- 0.05)	0.17	0.60 (+/- 0.03)	0.22	0.59 (+/- 0.04)	0.33
Polynomial	0.61 (+/- 0.04)	0.08	0.65 (+/- 0.03)	0.07	0.60 (+/- 0.05)	0.11	0.60 (+/- 0.06)	0.14	0.58 (+/- 0.03)	0.18	0.58 (+/- 0.05)	0.28
Linear	0.58 (+/- 0.03)	0.05	0.61 (+/- 0.04)	0.07	0.61 (+/- 0.05)	0.15	0.60 (+/- 0.04)	0.19	0.60 (+/- 0.04)	0.27	0.59 (+/- 0.03)	0.47
Sigmoid	0.48 (+/- 0.04)	0.09	0.41 (+/- 0.03)	0.09	0.56 (+/- 0.06)	0.16	0.59 (+/- 0.03)	0.19	0.59 (+/- 0.03)	0.23	0.59 (+/- 0.05)	0.34

Table 6.14 : A comparison of the performance of the SVM with different types of kernel functions on the reduced MADELON data set by PCA.

MADELON Data Set						
Model	Accuracy	Precision	Recall	f1-score	Fit Time(s)	Score Time(s)
RBF Kernel SVM with 5D-PCA	0.81 (+/- 0.02)	0.81 (+/- 0.02)	0.81 (+/- 0.02)	0.81 (+/- 0.02)	0.05	0.03
Polynomial Kernel SVM with 5D-PCA	0.65 (+/- 0.03)	0.65 (+/- 0.03)	0.65 (+/- 0.03)	0.65 (+/- 0.03)	0.05	0.02
Polynomial Kernel SVM	0.61 (+/- 0.04)	0.61 (+/- 0.05)	0.61 (+/- 0.04)	0.60 (+/- 0.03)	1.61	1.48
Linear Kernel SVM with 5D-PCA	0.61 (+/- 0.04)	0.61 (+/- 0.04)	0.61 (+/- 0.04)	0.61 (+/- 0.04)	0.05	0.02
Sigmoid Kernel SVM with 20D-PCA	0.59 (+/- 0.03)	0.59 (+/- 0.03)	0.59 (+/- 0.03)	0.59 (+/- 0.03)	0.11	0.08
2D-LDA	0.55 (+/- 0.05)	0.55 (+/- 0.05)	0.55 (+/- 0.05)	0.55 (+/- 0.05)	0.21	0.00

Table 6.15 : Table of the models which have the best performance in their methods on the MADELON data set.

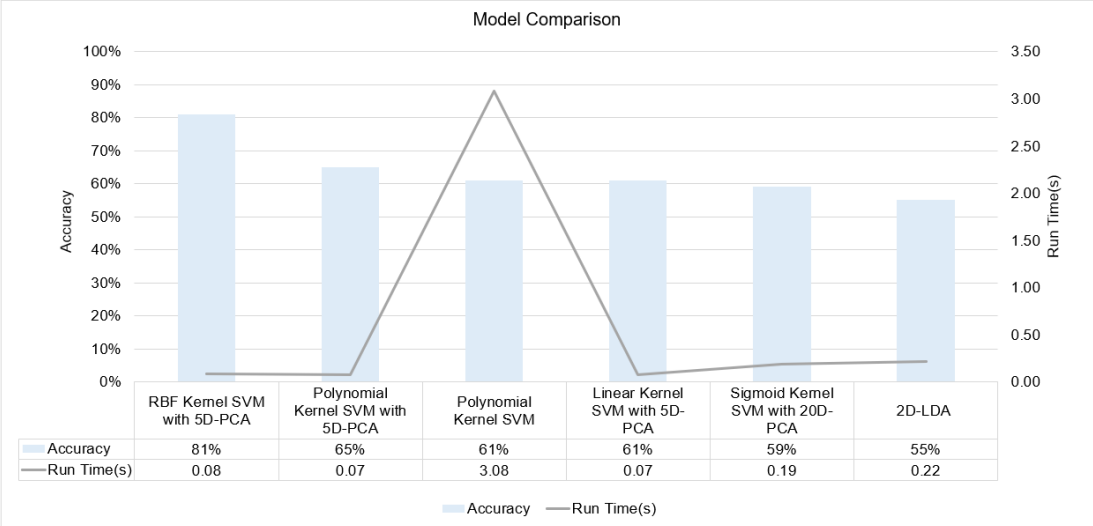


Figure 6.18 : Accuracy and run time comparison for the best models on the MADELON data set.

the ability to generate non-linear hyperplane by using the RBF kernel function on the highly non-linear data set. Another result we expected is that SVM performed better in 5-dimensional feature space since the MADELON data set consists of 5 real features. As a result, LDA as a linear classifier failed to recognize the higly non-linear data set.

7. CONCLUSION AND RECOMMENDATIONS

This thesis provided an in-depth investigation of the theoretical foundations of three main linear algebraic methods used in machine learning on three different data sets. Throughout the thesis, we observed that the performance of the models we developed vary depending on the type of data sets. We also observed that each model has different dynamics depending on the parameters one chooses.

Specifically, LDA performed very well on the Olivetti faces data set in terms of speed, scores and space, while the performance on the MADELON data set was very poor. We also found that the LDA's performance was good on the Fashion-MNIST data set, and had the same dynamics as the Olivetti faces data set in terms of its response to changing the parameters of the chosen model type. We observed that LDA's performance varied depending on the data type.

SVM was much slower on all datasets than LDA. The performance of the SVM model was quite good on the Olivetti faces data set and Fashion-MNIST data set, but did not exceed the LDA's time and space performance. One important thing that we see from the results is that the performances of the SVM models vary considerably according to the chosen kernel. The most significant example of this difference was the linear kernel vs the polynomial kernel on the Olivetti face data set. The polynomial kernel yielded 94% accuracy and while the linear kernel yielded 58%. On the MADELON data set, the linear kernel performed 8.27 times slower than the polynomial kernel. Therefore, we conclude that, the type of kernel affects both the accuracy score and the run time. We also observed that the most appropriate kernel varies from one data set to another.

One common result we observed across all data sets is that, when we used PCA and SVM together, we achieved better results in terms of accuracy scores, run times and the size reduction compared to the results we obtained when we used SVM alone. Although SVM with PCA did not catch the LDA's performance in terms of saving space, we were able to find models in SVM with PCA for which scores and time were better than LDA.

In summary, all these results showed us that the performance of the models vary according to the type of data set. It should be emphasized that, the best model may also vary depending on the combination of the methods and the parameters chosen for the methods. If we expect the model to be fast, or if we do not have enough space, we need to consider different methods and parameters for the methods we have at hand.

7.1 Future Work

This thesis contributed to the understanding of some linear algebraic methods for classification tasks, yet there are still many different research areas that future work can address. Our study can extend through the use of other type of data sets for different tasks. For instance, as future work, PCA can be applied to remote sensing data sets. This study can be also extended using other SVM kernels such as Laplace RBF kernel, rational quadratic kernel, multiquadric kernel, ANOVA radial basis kernel for classification tasks. On the other hand, different learning methods such as an artificial neural network can be applied to the same sample data, as well as these classifiers.

REFERENCES

[1] **James, G.**, **Witten, D.**, **Hastie, T. and Tibshirani, R.** (2013). *An introduction to statistical learning*, Springer.

[2] **Hastie, T.**, **Tibshirani, R. and Friedman, J.** (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*, Springer.

[3] **Bishop, C.M.** (2006). *Pattern recognition and machine learning*, Springer Science+ Business Media.

[4] **Turk, M. and Pentland, A.** (1991). Eigenfaces for Recognition, *J. Cognitive Neuroscience*, *3*(1), 71–86, *http://dx.doi.org/10.1162/jocn.1991.3.1.71*.

[5] **Xiao, H.**, **Rasul, K. and Vollgraf, R.**, (2017), Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, *cs.LG/1708. 07747*.

[6] **Guyon, I.**, **Gunn, S.**, **Ben-Hur, A. and Dror, G.** Result analysis of the NIPS 2003 feature selection challenge, *http://archive.ics.uci.edu/ml/datasets/ madelon*.

[7] **Li, C.**, **Diao, Y.**, **Ma, H. and Li, Y.** (2008). A statistical PCA method for face recognition, *2008 Second International Symposium on Intelligent Information Technology Application*, volume 3, IEEE, pp.376–380.

[8] **Sirovich, L. and Kirby, M.** (1987). Low-dimensional procedure for the characterization of human faces, *Josa a*, *4*(3), 519–524.

[9] **Kirby, M. and Sirovich, L.** (1990). Application of the Karhunen-Loeve procedure for the characterization of human faces, *IEEE Transactions on Pattern analysis and Machine intelligence*, *12*(1), 103–108.

[10] **Murase, H. and Nayar, S.K.** (1995). Visual learning and recognition of 3-D objects from appearance, *International journal of computer vision*, *14*(1), 5–24.

[11] **Murase, H.**, **Kimura, F.**, **Yoshimura, M. and Miyake, Y.** (1981). An improvement of the auto-correlation matrix in pattern matching method and its application to handprinted'HIRAGANA', *Trans. IECE*, *64*(3), 276–283.

[12] **Weng, J.** (1996). Cresceptron and SHOSLIF: Toward comprehensive visual learning, *Early visual learning*, 183–214.

[13] **Nayar, S.K.**, **Nene, S.A. and Murase, H.** (1996). Subspace methods for robot vision, *IEEE Transactions on Robotics and Automation*, *12*(5), 750–758.

[14] **Pearson, K.** (1901). LIII. On lines and planes of closest fit to systems of points in space, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11), 559–572.

[15] **Fisher, R.A. and Mackenzie, W.A.** (1923). Studies in crop variation. II. The manurial response of different potato varieties, *The Journal of Agricultural Science*, *13*(3), 311–320.

[16] **Hotelling, H.** (1933). Analysis of a complex of statistical variables into principal components., *Journal of educational psychology*, *24*(6), 417.

[17] **Golub, G. and VanLoan, C.** (1983). *Matrix computations*, Johns Hopkins University Press.

[18] **Mandel, J.** (1982). Use of the singular value decomposition in regression analysis, *The American Statistician*, *36*(1), 15–24.

[19] **Horn, R.A. and Johnson, C.R.**, (1985), Matrix Analysis: Section 2.8.

[20] **Karhunen, K.** (1947). Under Lineare Methoden in der Wahr Scheinlichkeitsrechnung, *Annales Academiae Scientiarun Fennicae Series A1: Mathematia Physica*, *137*.

[21] **Harman, H.H.** (1960). Modern factor analysis.

[22] **Weare, B.C. and Nasstrom, J.S.** (1982). Examples of extended empirical orthogonal function analyses, *Monthly Weather Review*, *110*(6), 481–485.

[23] **Chatterjee, A.** (2000). An introduction to the proper orthogonal decomposition, *Current science*, 808–817.

[24] **Gnanadesikan, R.** (2011). *Methods for statistical data analysis of multivariate observations*, volume321, John Wiley & Sons.

[25] **Mardia, K.**, **Kent, J. and Bibby, J.** (1979). *Multivariate Analysis*, Academic Press.

[26] **Johnson, R.A. and Wichern, D.W.** (1982). Applied multivariate statistical analysis Prentice Hall, *Inc.: Englewood Cliffs, NJ*.

[27] **Johffe, J.** (1986). *Principal Component Analysis*, Springer.

[28] **Belhumeur, P.N.**, **Hespanha, J.P. and Kriegman, D.J.** (1997). Eigenfaces vs. fisherfaces: Recognition using class specific linear projection, *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (7), 711–720.

[29] **Swets, D.L. and Weng, J.J.** (1996). Using discriminant eigenfeatures for image retrieval, *IEEE Transactions on pattern analysis and machine intelligence*, *18*(8), 831–836.

[30] **Dudoit, S.**, **Fridlyand, J. and Speed, T.P.** (2002). Comparison of discrimination methods for the classification of tumors using gene expression data, *Journal of the American statistical association*, *97*(457), 77–87.

[31] **Weng, J.** (1996). Cresceptron and SHOSLIF: Toward comprehensive visual learning, *Early visual learning*, 183–214.

[32] **Altman, E.I.** *et al.* (1973). Predicting railroad bankruptcies in America, *Bell Journal of Economics*, *4*(1), 184–211.

[33] **Tahmasebi, P.**, **Hezarkhani, A. and Mortazavi, M.** (2010). Application of discriminant analysis for alteration separation; sungun copper deposit, East Azerbaijan, Iran, *Australian Journal of Basic and Applied Sciences*, *6*(4), 564–576.

[34] **Preisner, O.**, **Guiomar, R.**, **Machado, J.**, **Menezes, J.C. and Lopes, J.A.** (2010). Application of Fourier transform infrared spectroscopy and chemometrics for differentiation of Salmonella enterica serovar Enteritidis phage types, *Appl. Environ. Microbiol.*, *76*(11), 3538–3544.

[35] **Fisher, R.A.** (1936). The use of multiple measurements in taxonomic problems, *Annals of eugenics*, *7*(2), 179–188.

[36] **Cortes, C. and Vapnik, V.** (1995). Support-vector networks, *Machine learning*, *20*(3), 273–297.

[37] **Vapnik, V.N.** (1995). The nature of statistical learning, *Theory*.

[38] **Pradhan, S.S.**, **Ward, W.H.**, **Hacioglu, K.**, **Martin, J.H. and Jurafsky, D.** (2004). Shallow semantic parsing using support vector machines, *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pp.233–240.

[39] **Barghout, L.**, (2015). Spatial-taxon information granules as used in iterative fuzzy-decision-making for image segmentation, Granular Computing and Decision-Making, Springer, pp.285–318.

[40] **Veropoulos, K.**, **Cristianini, N. and Campbell, C.** (1999). The application of support vector machines to medical decision support: a case study, *Advanced Course in Artificial Intelligence*, 1–6.

[41] **Fernandez, R.** (1999). Predicting time series with a local support vector regression machine, *In ACAI 99*, Citeseer.

[42] **Vapnik, V.** *et al.*, (1998), Statistical learning theory.

[43] **Boser, B.E.**, **Guyon, I.M. and Vapnik, V.N.** (1992). A training algorithm for optimal margin classifiers, *Proceedings of the fifth annual workshop on Computational learning theory*, ACM, pp.144–152.

[44] **Cortes, C. and Vapnik, V.** (1995). Support-vector networks, *Machine learning*, *20*(3), 273–297.

[45] **Veropoulos, K.**, **Cristianini, N. and Campbell, C.** (1999). The application of support vector machines to medical decision support: a case study, *Advanced Course in Artificial Intelligence*, 1–6.

[46] **Tefas, A.**, **Kotropoulos, C. and Pitas, I.** (1999). Enhancing the Performance of Elastic Graph Matching for Face Authentication by using Support Vector Machines, *IEEE Trans. on Pattern Analysis and Machine Intelligence*.

[47] **Alpaydin, E.**, (2014), Introduction to machine learning/Ethem Alpaydin.

[48] **Rish, I.** *et al.* (2001). An empirical study of the naive Bayes classifier, *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pp.41–46.

[49] **Lane, T. and Brodley, C.E.** (1997). An application of machine learning to anomaly detection, *Proceedings of the 20th National Information Systems Security Conference*, volume377, Baltimore, USA, pp.366–380.

[50] **Zhao, Q. and Bhowmick, S.S.** (2003). Association rule mining: A survey, *Nanyang Technological University, Singapore*.

[51] **Liu, B.**, **Hsu, W.**, **Ma, Y.** *et al.* (1998). Integrating classification and association rule mining., *KDD*, volume 98, pp.80–86.

[52] **Zhu, X.** (2005). Semi-supervised learning literature survey.

[53] **Cozman, F.G.**, **Cohen, I. and Cirelo, M.C.** (2003). Semi-supervised learning of mixture models, *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp.99–106.

[54] **Chen, Y.**, **Wang, G. and Dong, S.** (2003). Learning with progressive transductive Support Vector Machine, *Pattern Recognition Letters*, *24*.

[55] **Zhu, X. and Lafferty, J.** (2005). Harmonic mixtures: combining mixture models and graph-based methods for inductive and scalable semi-supervised learning.

[56] **Blum, A. and Chawla, S.** (2001). Learning from labeled and unlabeled data using graph mincuts.

[57] **Smith, L.** (2002). A tutorial on principal components analysis, *Cornell University, USA*, *52*.

[58] **Shores, T.S.** (2007). *Applied linear algebra and matrix analysis*, volume2541, Springer.

[59] **Chunming Li, Yanhua Diao, H.M.Y.L.** (2008). A Statistical PCA Method for Face Recognition, *3*, 376–380.

[60] **Lewicki, M.S. and Sejnowski, T.J.** (2000). Learning overcomplete representations, *Neural computation*, *12*.

[61] **Schölkopf, L.**, **Smola, A. and Müller, K.** (1997). Kernel Principal Component Analysis.

[62] **Le Roux, B. and Rouanet, H.** (2004). *Geometric data analysis: from correspondence analysis to structured data analysis*, Springer Science & Business Media.

[63] **Madsen, R.E.**, **Hansen, L.K. and Winther, O.** (2004). Singular value decomposition and principal component analysis, *Rasmus Elsborg Madsen,*.

[64] **S. Balakrishnama, A.G.** LINEAR DISCRIMINANT ANALYSIS - A BRIEF TUTORIAL.

[65] **Siqueira, L.F.**, **Júnior, R.F.A.**, **de Araújo, A.A.**, **Morais, C.L. and Lima, K.M.** (2017). LDA vs. QDA for FT-MIR prostate cancer tissue classification, *Chemometrics and Intelligent Laboratory Systems*, *162*, 123–129.

[66] **Chen, W.H.**, **Hsu, S.H. and Shen, H.P.** (2005). Application of SVM and ANN for intrusion detection, *Computers & Operations Research*, *32*(10), 2617–2634.

[67] **Smits, G.F. and Jordaan, E.M.** (2002). Improved SVM regression using mixtures of kernels, *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, volume 3, IEEE, pp.2785–2790.

[68] **Pedregosa, F.**, **Varoquaux, G.**, **Gramfort, A.**, **Michel, V.**, **Thirion, B.**, **Grisel, O.**, **Blondel, M.**, **Prettenhofer, P.**, **Weiss, R.**, **Dubourg, V.**, **Vanderplas, J.**, **Passos, A.**, **Cournapeau, D.**, **Brucher, M.**, **Perrot, M. and Duchesnay, E.** (2011). Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research*, *12*, 2825–2830.

[69] **McKinney, W.** (2010). Data Structures for Statistical Computing in Python, ***S. van der Walt and J. Millman***, *editors, Proceedings of the 9th Python in Science Conference*, pp.51 – 56.

[70] **Xiao, H.**, **Rasul, K. and Vollgraf, R.**, (2017), Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, *cs.LG/1708.07747*.

[71] **Guyon, I. and Elisseeff, A.** (2003). An introduction to variable and feature selection, *Journal of machine learning research*, *3*(Mar), 1157–1182.

[72] **urllib2**, Extensible library for opening URLs, *https://docs.python.org/2/library/urllib2.html*.

APPENDICES

APPENDIX A.1 : Python Codes for Olivetti Faces Data Set
APPENDIX A.2 : Python Codes for Fashion-MNIST Data Set
APPENDIX A.3 : Python Codes for MADELON Data Set

APPENDIX A.1

1.1 Olivetti Faces Data Set

1.1.1 Data preprocessing

```
# Importing libraries

from sklearn import datasets
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_validate
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

# Importing the data set

faces = datasets.fetch_olivetti_faces()
df = pd.DataFrame(data=faces.data,index=range(400))

# Showing the 40 distinct people

fig=plt.figure(figsize=(16,8))
for i in range(40):
    ax=fig.add_subplot(4,10,i+1,xticks=[i], yticks=[])
    ax.imshow(faces.images[i*10], cmap=plt.cm.bone)

# Showing the all images

fig=plt.figure(figsize=(64,64))
for i in range(400):
    ax=fig.add_subplot(25,16,i+1,xticks=[], yticks=[])
    ax.imshow(faces.images[i], cmap=plt.cm.bone)

# Feature Scaling

scl = StandardScaler()
faces_data = scl.fit_transform(faces.data)
```

1.1.2 Applying models

```
# Transformed data by LDA

lda=LinearDiscriminantAnalysis(n_components=2)
X_train_lda=lda.fit(faces_data, faces.target)
X_train=lda.transform(faces_data)

fig = plt.figure(figsize=(15,5))
fig.add_subplot(1,2,1)
plt.scatter(faces_data[:,0], faces_data[:,1], alpha=0.2, c=faces.target, cmap='viridis')
plt.xlabel('Feature␣1')
plt.ylabel('Feature␣2')
#fig.add_subplot(1,3,2)
#plt.scatter(faces_data[:,0], faces_data[:,2], alpha=0.2, c=faces.target, cmap='viridis')
fig.add_subplot(1,2,2)
plt.scatter(X_train[:,0], X_train[:,1], alpha=0.2, c=faces.target, cmap='viridis')
plt.xlabel('LDA1')
plt.ylabel('LDA2')

# Performing LDA

dimension=[2,5,15]

for i in dimension:
    lda = LinearDiscriminantAnalysis(n_components = i)
    scoring = ['accuracy', 'precision_macro', 'recall_macro',  'f1_macro']
    scores = cross_validate(lda, faces_data, faces.target, scoring = scoring, cv = 3)
    print('{}D-LDA'.format(i))
    print('Fit␣Time', scores['fit_time'].mean().round(3))
    print('Score␣Time', scores['score_time'].mean().round(3))
    print("Accuracy:␣%0.2f␣(+/−␣%0.2f)" % (scores['test_accuracy'].mean(), scores['test_accuracy'].std() * 2))
    print("Precision:␣%0.2f␣(+/−␣%0.2f)" % (scores['test_precision_macro'].mean(), scores['test_precision_macro'].std() * 2))
    print("Recall:␣%0.2f␣(+/−␣%0.2f)" % (scores['test_recall_macro'].mean(), scores['test_recall_macro'].std() * 2))
    print("f1−score:␣%0.2f␣(+/−␣%0.2f)" % (scores['test_f1_macro'].mean(), scores['test_f1_macro'].std() * 2))
    print(scores)
```

```
# Performing SVM with different kernels

models=[]
models.append(("Linear-SVM",SVC(kernel = 'linear')))
models.append(("RBF-SVM",SVC(kernel = 'rbf')))
models.append(("Polynomial-SVM",SVC(kernel = 'poly')))
models.append(("Sigmoid-SVM",SVC(kernel = 'sigmoid')))

for name, model in models:

    scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
    scores = cross_validate(model, faces_data, faces.target, scoring = scoring, cv = 3)
    print(name)
    print('Fit Time', scores['fit_time'].mean().round(3))
    print('Score Time', scores['score_time'].mean().round(3))
    print("Accuracy: %0.2f (+/- %0.2f)" % (scores['test_accuracy'].mean(), scores['test_accuracy'].std() * 2))
    print("Precision: %0.2f (+/- %0.2f)" % (scores['test_precision_macro'].mean(), scores['test_precision_macro'].std() * 2))
    print("Recall: %0.2f (+/- %0.2f)" % (scores['test_recall_macro'].mean(), scores['test_recall_macro'].std() * 2))
    print("f1-score: %0.2f (+/- %0.2f)" % (scores['test_f1_macro'].mean(), scores['test_f1_macro'].std() * 2))
    print(scores)


# Performing PCA

pca = PCA(n_components = 400, svd_solver = 'randomized', whiten = True).fit(faces_data)
X_train_pca = pca.transform(faces_data)
dimension=np.array(range(1,401))
variance=(pca.explained_variance_ratio_)
variance_cum=np.cumsum(variance)
plt.figure(figsize=(9, 5))
print(plt.plot(dimension,variance_cum))
plt.title('Percentage of variance explained by each of the selected components')
plt.xlabel('Principal components')
plt.ylabel('Explained Variance Ratio')
plt.show()


# Performing 2D-PCA

pca = PCA(n_components = 2, svd_solver = 'randomized',  whiten = True).fit(faces_data)
X_train_pca = pca.transform(faces_data)
fig = plt.figure(figsize=(15,5))
fig.add_subplot(1,2,1)
plt.scatter(faces_data[:,0], faces_data[:,1], alpha=0.2, c=faces.target, cmap='viridis')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
fig.add_subplot(1,2,2)
plt.scatter(X_train_pca[:,0], X_train_pca[:,1], alpha=0.2, c=faces.target, cmap='viridis')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')

# Visualize the result of 2D-PCA

plt.style.use('fivethirtyeight')

fig, axarr = plt.subplots(1, 2, figsize=(15, 5))

sns.heatmap(pca.components_[0, :].reshape(64, 64), ax=axarr[0], cmap='gray_r')
sns.heatmap(pca.components_[1, :].reshape(64, 64), ax=axarr[1], cmap='gray_r')
axarr[0].set_title(
    "{0:.2f}% Explained Variance".format(pca.explained_variance_ratio_[0]*100),
    fontsize=12
)
axarr[1].set_title(
    "{0:.2f}% Explained Variance".format(pca.explained_variance_ratio_[1]*100),
    fontsize=12
)
axarr[0].set_aspect('equal')
axarr[1].set_aspect('equal')

plt.suptitle('2D-PCA')

#Showing the average face by 50D-PCA

pca=PCA(n_components=50, whiten=True)
pca.fit(faces.data)
fig,ax=plt.subplots(1,1,figsize=(5,5))
ax.imshow(pca.mean_.reshape((64,64)), cmap="gray")
ax.set_xticks([])
ax.set_yticks([])
ax.set_title('Average Face')
```

APPENDIX A.2

1.2 Fashion-MNIST Data Set

1.2.1 Data preprocessing

```
# Importing libraries

import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_validate
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

# Importing the data set

df = pd.read_csv("fashion-mnist.csv")

# Data Preprocessing

x = df.iloc[:,1:]
y = df.iloc[:,0]

# Display first image

images_array= np.array(x)
sns.heatmap(images_array[0, :].reshape(28, 28) )

# Display first 3 images

plt.style.use('fivethirtyeight')
fig, axarr = plt.subplots(1, 3, figsize = (7, 3))
a=sns.heatmap(images_array[0, :].reshape(28, 28), ax = axarr[0], cmap = 'gray_r')
a.set(xticks = [])
a.set(yticks = [])
b=sns.heatmap(images_array[1, :].reshape(28, 28), ax = axarr[1], cmap = 'gray_r')
b.set(xticks = [])
b.set(yticks = [])
c=sns.heatmap(images_array[2, :].reshape(28, 28), ax = axarr[2], cmap = 'gray_r')
c.set(xticks = [])
c.set(yticks = [])
plt.suptitle('Visualize the first 3 images from the data set')

# Feature Scaling

scl = StandardScaler()
X = scl.fit_transform(x)
```

1.2.2 Applying models

```
# Transformed data by LDA

lda=LinearDiscriminantAnalysis(n_components=2)
X_train_lda=lda.fit(X, y)
X_train=lda.transform(X)

fig = plt.figure(figsize=(15,5))
fig.add_subplot(1,2,1)
plt.scatter(X[:,0], X[:,1], alpha=0.2, c=y, cmap='viridis')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
#fig.add_subplot(1,3,2)
#plt.scatter(faces_data[:,0], faces_data[:,2], alpha=0.2, c=faces.target, cmap='viridis')
fig.add_subplot(1,2,2)
plt.scatter(X_train[:,0], X_train[:,1], alpha=0.2, c=y, cmap='viridis')
plt.xlabel('LDA1')
plt.ylabel('LDA2')

# Performing LDA

dimension=[2,5,15]

for i in dimension:
    lda = LinearDiscriminantAnalysis(n_components = i)
    scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
    scores = cross_validate(lda, X, y, scoring = scoring, cv = 3)
    print('{}D-LDA'.format(i))
    print('Fit Time', scores['fit_time'].mean().round(3))
    print('Score Time', scores['score_time'].mean().round(3))
    print("Accuracy: %0.2f (+/- %0.2f)" % (scores['test_accuracy'].mean(), scores['test_accuracy'].std() * 2))
```

```
    print("Precision: %0.2f (+/- %0.2f)" % (scores['test_precision_macro'].mean(), scores['test_precision_macro'].std() * 2))
    print("Recall: %0.2f (+/- %0.2f)" % (scores['test_recall_macro'].mean(), scores['test_recall_macro'].std() * 2))
    print("f1-score: %0.2f (+/- %0.2f)" % (scores['test_f1_macro'].mean(), scores['test_f1_macro'].std() * 2))
    print(scores)


# Performing SVM with different kernels

models = []
models.append(("Linear-SVM", SVC(kernel = 'linear')))
models.append(("RBF-SVM", SVC(kernel = 'rbf')))
models.append(("Polynomial-SVM", SVC(kernel = 'poly')))
models.append(("Sigmoid-SVM", SVC(kernel = 'sigmoid')))

for name, model in models:

    scoring = ['accuracy', 'precision_macro', 'recall_macro',  'f1_macro']
    scores = cross_validate(model, X, y, scoring = scoring, cv = 3)
    print(name)
    print('Fit Time', scores['fit_time'].mean().round(3))
    print('Score Time', scores['score_time'].mean().round(3))
    print("Accuracy: %0.2f (+/- %0.2f)" % (scores['test_accuracy'].mean(), scores['test_accuracy'].std() * 2))
    print("Precision: %0.2f (+/- %0.2f)" % (scores['test_precision_macro'].mean(), scores['test_precision_macro'].std() * 2))
    print("Recall: %0.2f (+/- %0.2f)" % (scores['test_recall_macro'].mean(), scores['test_recall_macro'].std() * 2))
    print("f1-score: %0.2f (+/- %0.2f)" % (scores['test_f1_macro'].mean(), scores['test_f1_macro'].std() * 2))
    print(scores)
    print('---')

# Performing PCA

pca = PCA(n_components = 784, svd_solver = 'randomized',  whiten = True).fit(X)
X_train_pca = pca.transform(X)
dimension=np.array(range(1,785))
variance=(pca.explained_variance_ratio_)
variance_cum=np.cumsum(variance)
plt.figure(figsize=(9, 5))
plt.plot(dimension,variance_cum)
plt.title('Percentage of variance explained by each of the selected components')
plt.xlabel('Principal components')
plt.ylabel('Explained Variance Ratio')
plt.show()
print(variance.sum())

# Performing 2D-PCA

pca = PCA(n_components = 2, svd_solver = 'randomized',  whiten = True).fit(X)
X_train_pca = pca.transform(X)
fig = plt.figure(figsize=(15,5))
fig.add_subplot(1,2,1)
plt.scatter(X[:,0], X[:,1], alpha=0.2, c=y, cmap='viridis')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
fig.add_subplot(1,2,2)
plt.scatter(X_train_pca[:,0], X_train_pca[:,1], alpha=0.2, c=y, cmap='viridis')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')

# Visualize the result of 2D-PCA

plt.style.use('fivethirtyeight')

fig, axarr = plt.subplots(1, 2, figsize=(15, 5))

sns.heatmap(pca.components_[0, :].reshape(28, 28), ax=axarr[0], cmap='gray_r')
sns.heatmap(pca.components_[1, :].reshape(28, 28), ax=axarr[1], cmap='gray_r')

axarr[0].set_title(
    "{0:.2f}% Explained Variance".format(pca.explained_variance_ratio_[0]*100),
    fontsize=12
)
axarr[1].set_title(
    "{0:.2f}% Explained Variance".format(pca.explained_variance_ratio_[1]*100),
    fontsize=12
)
axarr[0].set_aspect('equal')
axarr[1].set_aspect('equal')

plt.suptitle('2D-PCA')

#Showing the average fashion item by PCA

pca=PCA(n_components=50, whiten=True)
pca.fit(x)
fig,ax=plt.subplots(1,1,figsize=(5,5))
ax.imshow(pca.mean_.reshape((28,28)), cmap="gray")
ax.set_xticks([])
ax.set_yticks([])
ax.set_title('Average Fashion Image')

# Performing SVM with different kernels after applying PCA

dimension=[2,5,15,20,30,50]
models=[]
models.append(("Linear",SVC(kernel = 'linear')))
models.append(("RBF",SVC(kernel = 'rbf')))
models.append(("Polynomial",SVC(kernel = 'poly')))
models.append(("Sigmoid",SVC(kernel = 'sigmoid')))

for i in dimension:
    pca = PCA(n_components = i, svd_solver = 'randomized',  whiten = True).fit(X)
    X_train_pca = pca.transform(X)
```

```
for name, model in models:
    scoring = ['accuracy', 'precision_macro', 'recall_macro',  'f1_macro']
    scores = cross_validate(model, X_train_pca, y, scoring = scoring, cv = 5)
    print('Performing SVM with {} kernel after applying {}D PCA'.format(name,i))
    print('Fit Time', scores['fit_time'].mean().round(3))
    print('Score Time', scores['score_time'].mean().round(3))
    print("Accuracy: %0.2f (+/- %0.2f)" % (scores['test_accuracy'].mean(), scores['test_accuracy'].std() * 2))
    print("Precision: %0.2f (+/- %0.2f)" % (scores['test_precision_macro'].mean(), scores['test_precision_macro'].std() * 2))
    print("Recall: %0.2f (+/- %0.2f)" % (scores['test_recall_macro'].mean(), scores['test_recall_macro'].std() * 2))
    print("f1-score: %0.2f (+/- %0.2f)" % (scores['test_f1_macro'].mean(), scores['test_f1_macro'].std() * 2))
    print(scores)
    print('---')
```

APPENDIX A.3

1.3 MADELON Data Set

1.3.1 Data preprocessing

```
# Importing libraries

import urllib.request as urllib2
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_validate
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import axes3d

# Importing the data set

train_data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/madelon/MADELON/madelon_train.data'
val_data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/madelon/MADELON/madelon_valid.data'
train_resp_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/madelon/MADELON/madelon_train.labels'
val_resp_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/madelon/madelon_valid.labels'
test_data_url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/madelon/MADELON/madelon_test.data'
x_train = np.loadtxt(urllib2.urlopen(train_data_url))
y_train = np.loadtxt(urllib2.urlopen(train_resp_url))
x_val = np.loadtxt(urllib2.urlopen(val_data_url))
y_val = np.loadtxt(urllib2.urlopen(val_resp_url))

# Feature Scaling

scl = StandardScaler()
X_train = scl.fit_transform(x_train)

# Visualize Hypercube in 3D

fig = plt.figure(figsize = (5,5))
ax = fig.add_subplot(111,projection = '3d')
X_1 = X_train[:,0]
X_2 = X_train[:,1]
X_3 = X_train[:,2]
ax.scatter(X_1, X_2, X_3, c = y_train)
cmaps = 'magma'
ax.view_init(30,20)
```

1.3.2 Applying models

```
# Performing LDA

dimension=[2,5,15]
for i in dimension:
    lda = LinearDiscriminantAnalysis(n_components = i)
    scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
    scores = cross_validate(lda, X_train, y_train, scoring = scoring, cv = 5)
    print('{}D-LDA'.format(i))
    print('Fit Time', scores['fit_time'].mean().round(3))
    print('Score Time', scores['score_time'].mean().round(3))
    print("Accuracy: %0.2f (+/- %0.2f)" % (scores['test_accuracy'].mean(), scores['test_accuracy'].std() * 2))
    print("Precision: %0.2f (+/- %0.2f)" % (scores['test_precision_macro'].mean(), scores['test_precision_macro'].std() * 2))
    print("Recall: %0.2f (+/- %0.2f)" % (scores['test_recall_macro'].mean(), scores['test_recall_macro'].std() * 2))
    print("f1-score: %0.2f (+/- %0.2f)" % (scores['test_f1_macro'].mean(), scores['test_f1_macro'].std() * 2))
    print(scores)


# Performing SVM with different kernels

models=[]
models.append(("Linear-SVM", SVC(kernel = 'linear')))
models.append(("RBF-SVM", SVC(kernel = 'rbf')))
models.append(("Polynomial-SVM", SVC(kernel = 'poly')))
models.append(("Sigmoid-SVM", SVC(kernel = 'sigmoid')))

for name, model in models:

    scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
    scores = cross_validate(model, X_train, y_train, scoring = scoring, cv = 5)
    print(name)
    print('Fit Time', scores['fit_time'].mean().round(3))
    print('Score Time', scores['score_time'].mean().round(3))
    print("Accuracy: %0.2f (+/- %0.2f)" % (scores['test_accuracy'].mean(), scores['test_accuracy'].std() * 2))
    print("Precision: %0.2f (+/- %0.2f)" % (scores['test_precision_macro'].mean(), scores['test_precision_macro'].std() * 2))
    print("Recall: %0.2f (+/- %0.2f)" % (scores['test_recall_macro'].mean(), scores['test_recall_macro'].std() * 2))
```

```
    print("f1-score: %0.2f (+/- %0.2f)" % (scores['test_f1_macro'].mean(), scores['test_f1_macro'].std() * 2))
    print(scores)


# Performing PCA

pca = PCA(n_components = 500, svd_solver = 'randomized',  whiten = True).fit(X_train)
X_train_pca = pca.transform(X_train)
dimension=np.array(range(1,501))
variance=(pca.explained_variance_ratio_)
variance_cum=np.cumsum(variance)
plt.figure(figsize=(9, 5))
plt.plot(dimension,variance_cum)
plt.title('Percentage of variance explained by each of the selected components')
plt.xlabel('Principal components')
plt.ylabel('Explained Variance Ratio')
plt.show()

# Visualize data points on 3D-Hypercube

fig = plt.figure(figsize = (5,5))
ax = fig.add_subplot(111,projection = '3d')

X_1 = X_train[:,0]
X_2 = X_train[:,1]
X_3 = X_train[:,2]
ax.scatter(X_1, X_2, X_3, c = y_train)
cmaps = 'magma'
ax.view_init(30,20)

pca = PCA(n_components = 3, svd_solver = 'randomized',  whiten = True).fit(X_train)
X_train_pca = pca.transform(X_train)

from mpl_toolkits.mplot3d import axes3d
fig = plt.figure(figsize = (5,5))
ax = fig.add_subplot(111,projection = '3d')

X_1 = X_train_pca[:,0]
X_2 = X_train_pca[:,1]
X_3 = X_train_pca[:,2]
ax.scatter(X_1, X_2, X_3, c = y_train)
cmaps = 'magma'
ax.view_init(30,20)

# Performing 2D-PCA

pca = PCA(n_components = 2, svd_solver = 'randomized',  whiten = True).fit(X_train)
X_train_pca = pca.transform(X_train)
fig = plt.figure(figsize=(15,5))
fig.add_subplot(1,2,1)
plt.scatter(X_train[:,0], X_train[:,1], alpha=0.2, c=y_train, cmap='viridis')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
fig.add_subplot(1,2,2)
plt.scatter(X_train_pca[:,0], X_train_pca[:,1], alpha=0.2, c=y_train, cmap='viridis')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')

# Performing SVM with different kernels after applying PCA

dimension=[2,5,15,20,30,50]
models=[]
models.append(("Linear", SVC(kernel = 'linear')))
models.append(("RBF", SVC(kernel = 'rbf')))
models.append(("Polynomial", SVC(kernel = 'poly')))
models.append(("Sigmoid", SVC(kernel = 'sigmoid')))

for i in dimension:
    pca = PCA(n_components = i, svd_solver = 'randomized',  whiten = True).fit(X_train)
    X_train_pca = pca.transform(X_train)
    for name, model in models:
        scoring = ['accuracy', 'precision_macro', 'recall_macro',  'f1_macro']
        scores = cross_validate(model, X_train_pca, y_train, scoring = scoring, cv = 5)
        print('Performing SVM with {} kernel after applying {}D-PCA'.format(name,i))
        print('Fit Time', scores['fit_time'].mean().round(3))
        print('Score Time', scores['score_time'].mean().round(3))
        print("Accuracy: %0.2f (+/- %0.2f)" % (scores['test_accuracy'].mean(), scores['test_accuracy'].std() * 2))
        print("Precision: %0.2f (+/- %0.2f)" % (scores['test_precision_macro'].mean(), scores['test_precision_macro'].std() * 2))
        print("Recall: %0.2f (+/- %0.2f)" % (scores['test_recall_macro'].mean(), scores['test_recall_macro'].std() * 2))
        print("f1-score: %0.2f (+/- %0.2f)" % (scores['test_f1_macro'].mean(), scores['test_f1_macro'].std() * 2))
        print(scores)
```

CURRICULUM VITAE

Name Surname: Elif ALTUNOK

Place and Date of Birth: Zile, 27 July 1992

E-Mail: altunokelif@gmail.com

EDUCATION:

- **B.Sc.:** 2013, Mimar Sinan Fine Arts University, Faculty of Science and Letters, Mathematics, GPA: 3,32

PROFESSIONAL EXPERIENCE:

- 2018-2019 Istanbul Technical University - Scientific Researcher at Tubitak project
- 2013-2017 Shell Upstream Turkey B.V. - Financial Reporting Analyst

PUBLICATIONS:

- Külekci O., Öztürk Y., Altunok, E., Altınığne C. Y., 2019. Enumerative Data Compression with Non-Uniquely Decodable Codes. *Working paper, Cornell University, arXiv.org*