Linear versus Nonlinear Dimensionality reduction

Dimensionality reduction as preproccesing of image data for use in image classification

Daniel Runge Petersen, Gustav Svante Graversen, Lars Emanuel Hansen, Raymond Kacso, Sebastian Aaholm

Computer Science, cs-22-dat-5-05, 2022-12

Semester Project

Composed and typeset by the authors using the LaTeX Document Preparation System, based on the AAU report template by Daniel Runge Petersen [1].

AALBORG UNIVERSITY
STUDENT REPORT

Electronics and IT
Aalborg University
http://www.aau.dk

Title:
Linear versus Nonlinear Dimensionality Reduction

Theme:
Theoretical data analysis and modeling

Project Period:
Fall Semester 2022

Project Group:
cs-22-dat-5-05

Participant(s):
Daniel Runge Petersen
Gustav Svante Graversen
Lars Emanuel Hansen
Raymond Kacso
Sebastian Aaholm

Supervisor(s):
Alexander Leguizamon Robayo

Copies: 1

Page Numbers: 89

Date of Completion:
December 20, 2022

Abstract:

In recent years, dimensionality reduction has become an increasingly important topic in Machine Learning (ML). Previous research has shown that nonlinear dimensionality reduction methods might not outperform linear methods. In this paper, we use Modified National Institute of Standards and Technology (MNIST) to compare the dimensionality reduction methods and evaluate the results based on an Support Vector Machine (SVM) model. The evaluation is based on time, accuracy, f1 score, precision, and recall. The results show that linear methods are faster than nonlinear and achieve similar accuracy levels. In contrast, nonlinear methods are less sensitive to fluctuations in accuracy score when the number of dimensions reduced becomes smaller and smaller. With the current findings, nonlinear methods do not outperform linear methods, though the methods' peculiarities from this project serve as a foundation for future work.

Contents

Preface

This report is written by group cs-22-dat-5-05, 5th semester at AAU. The theme of the report is "Theoretical data analysis and modeling", and is completed at the end of the fall semester of 2022.

The report has been completed with help from our supervisor, Alexander Leguizamon Robayo.

Aalborg University, December 20, 2022

Daniel Runge Petersen
<username@student.aau.dk>

Gustav Svante Graversen
<username@student.aau.dk>

Lars Emanuel Hansen
<usernamestudent.aau.dk>

Raymond Kacso
<username@student.aau.dk>

Sebastian Aaholm
<username@student.aau.dk>

Chapter 1

Introduction

Dimensionality reduction is an important topic in the field of ML. The goal of dimensionality reduction is to reduce the number of features in a dataset while retaining as much relevant information as possible. This can be useful for improving the performance of ML algorithms, as well as for visualizing and understanding the structure of a dataset [2]. In this report, we focus on the comparison of two different dimensionality reduction techniques: linear and nonlinear methods.

Linear methods for dimensionality reduction, such as Principal Component Analysis (PCA), are widely used and have been well studied. These methods are computationally efficient and often perform well on a variety of tasks [3]. However, nonlinear methods, such as Kernel Principal Component Analysis (kPCA) are also relevant due to their ability to capture more complex structures in a dataset [2]. In this report, we aim to compare the relative effectiveness of these two types of dimensionality reduction methods.

We evaluate the performance of linear and nonlinear dimensionality reduction techniques on the MNIST dataset, a popular benchmark for image classification and recognition tasks [4]. We use a SVM model to classify the images in the reduced datasets and compare the performance of the model under different settings. Our results provide insights into the relative effectiveness of linear and nonlinear methods for dimensionality reduction in the context of Computer Vision (CV).

Overall, this report is of interest to researchers and practitioners in the field of ML and CV. Our findings may have implications for the use of dimensionality reduction techniques in image classification and recognition tasks, and can be used for future research in this area. By comparing linear and nonlinear methods, we aim to shed light on the relative strengths and limitations of these techniques and provide guidance for their practical use.

1.1 Motivation

High-dimensional data (i.e. data that requires more than three dimensions to be represented) can often be difficult to work with. Not only is it difficult to interpret and visualize but also can require a high use of computational resources. For these (and many more) reasons, it is important to study dimensionality reduction methods. These methods are usually used in exploratory data analysis and for visualization purposes.

The most usual methods of dimensionality reduction are **linear methods**. These methods might assume that the features in the original data are independent and they can produce reduced data by a linear combination of the original data. These assumptions might not apply to all datasets. In fact, there are cases in which linear methods do not capture important features of a dataset. For these cases one can use **nonlinear methods**. These methods can be used for more general cases while preserving important information from data.

Chapter 2

Problem analysis

This chapter describes the motivation for the project, culminating in a problem statement.

ML is a field of study within Artificial Intelligence (AI) concerned with learning from data, where learning means that data is analyzed and something is gathered from it. ML could be methods to solve a puzzle or patterns to recognize a number from an image. ML is a complex and growing field used in many areas. One of the reasons for the increasing interest in ML is the increasing amount of available data. The data collected worldwide is increasing at an incredible rate, which seems to continue in the future [5]. Because ML models train on data, the more data available, the better the models can be [6].

ML can be described as the discipline of teaching computers how to complete tasks where no perfect algorithm is possible. This also covers when there are many possible good ways to achieve something; for this, all the acceptable methods are labeled as acceptable ways to succeed. These answers help to "train" the computer to improve on some basic algorithm [7].

Inside a field as complex as ML, many challenges exist; one such challenge is working with high-dimensional data. This is due to the inherent properties of many dimensions, making it challenging to interpret and computationally expensive. Higher dimensions are also associated with the "curse of dimensionality". Richard E. Bellman coined this term in a paper about dynamic programming [8]. According to John A. Lee:

> the curse of dimensionality also refers to the fact that in the absence of simplifying assumptions, the number of data samples required to estimate a function of several variables to a given accuracy ... on a given domain grows exponentially with the number of dimensions [9].

These difficulties associated with data in many dimensions make the study of dimensionality reduction highly relevant. The goal of dimensionality reduction is to reduce the number of dimensions in a dataset while retaining as much information as possible. Dimensionality reduction can improve interpretability on data and, if used in a ML pipeline, make the pipeline faster. Dimensionality reduction is made by extracting features from the data, which trains a model. The extracted features help to predict new results on new data. This section is a general overview of the dimensionality reduction process and will be discussed in more detail in the next chapter. It will be explained what it is and some of its uses in the following paragraphs.

2.1 Machine learning

A pipeline is good for comparing and contrasting linear and nonlinear methods through the lens of a model efficiently. In this section, there will be a brief Introduction to pipelines, their use, and how to introduce dimensionality reduction to them.

Figure 2.1 illustrates a general ML pipeline. A ML pipeline consists of four main steps: data, Feature Engineering (FE), ML model training, and model evaluation. The first step is the data step, which is the data collection. The FE step transforms the data into a more suitable form for the ML model. The ML model training step trains the ML model on the data. The model evaluation step is used to evaluate the performance of the ML model. The ML model can then be used to make predictions on new data [10].

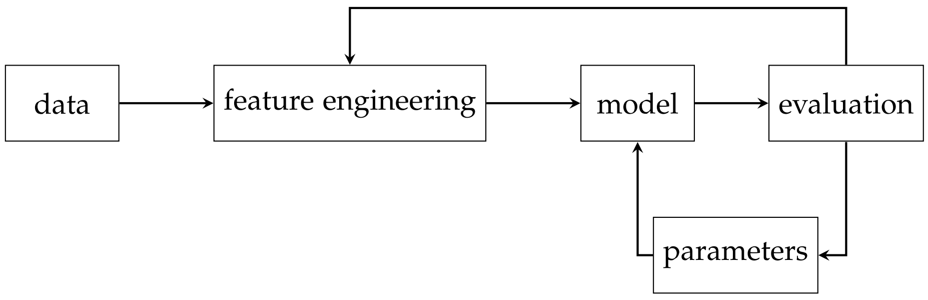


Figure 2.1: Simplified machine learning pipeline

The FE step is where dimensionality reduction is relevant - this step reduces the dimensions to increase the model's performance.

2.2 Dimensionality reduction

Dimensionality reduction is reducing the number of features in the data. This is done by removing features that are irrelevant to the task at hand or by combining features. The goal of dimensionality reduction is to reduce the number of features in the data while still retaining the essential information [2].

The advantages of dimensionality reduction are that it can reduce the amount of data that needs to be processed, making the model faster or require fewer resources. It can also make the model more interpretable because there are fewer features to look at. Dimensionality reduction can also improve the model's performance because it can remove noise from the data and make the model more robust [2].

Time is a significant factor when discussing the ML model, as the time complexity of the model is a significant factor in the overall time complexity of the system.

> Dimensionality reduction techniques can tremendously reduce the time complexity of training phase of ML algorithms hence reducing the burden of the machine learning algorithms [11].

Therefore, when reducing features in a dataset, it is crucial to find the features that are more relevant to the output of the model [12]. There are many ways to reduce the dimensionality of a dataset; the different types of dimensionality reduction will be further discussed in Chapter 3.

Interpretability is another significant factor when discussing the ML model; interpretability is the ability to understand the model's decisions. The interpretability of the model is a significant factor in the overall understanding of the system. This project will not focus on interpretability, but it is still a significant factor when discussing the ML model.

2.2.1 Applications of dimensionality reduction methods

Dimensionality reduction has many applications, and different methods can prove to be more suitable for different applications. An example of the applications of dimensionality reduction can be seen in [13], which discusses the use of dimensionality reduction in recommender systems. Using dimensionality reduction methods helps the system's scalability by reducing the amount of data run on the system and improving the quality of the recommendations.

There are many dimensionality reduction methods, and more are still being researched today [2]. Moreover, there are many different uses for these methods. An example of this could be to improve heuristic models for explaining the data from surveys better, through better visualizations as it helps to improve understanding [2].

Additionally, dimensionality reduction can be used in applications such as image compression, visualization of high-dimensional data, and feature extraction. It shows that it has many different applications and is a handy tool [13].

Many more examples exist, but the main takeaway is that dimensionality reduction is a powerful tool in many different fields and a vital part of the ML model. The following section will categorize dimensionality reduction into two general categories.

2.3 Linear versus nonlinear methods

This section will explore dimensionality reduction further. Specifically, this project will distinguish between linear and nonlinear methods. According to John A. Lee [9], several distinctions can be made for dimensionality reduction methods. This project will only focus on one distinction from [9], linear and nonlinear because it is a very straightforward way of classifying dimensionality reduction.

It is essential to distinguish between methods to classify them and to understand their differences. This will help to understand the advantages and disadvantages of each method. This will also help to understand which method is the most suitable for a specific problem [9]. The following section will discuss the differences between linear and nonlinear methods.

2.3.1 Results and differences of linear and nonlinear methods

As outlined earlier, dimensionality reduction methods can be used to remove redundancy from data, which can improve the performance of a ML model. However, the methods can also be used for other purposes, such as visualization and feature engineering [9].

A linear method assumes linear independence of the features. Linear independence means that the features are independent of each other; this is a strong assumption, which is only sometimes true [14]. A nonlinear method does not assume linear independence of the features, which means that the features are not independent; this is a weaker assumption, which is often true [15]. That means that a nonlinear method is often more robust than a linear method. However, a nonlinear method requires more parameters, which can require more data in a model [9].

Examples of how linear and nonlinear dimensionality reduction methods can be used can be seen in [16, 17], where the methods have been tested on artificial and real-world datasets. As an example, it has shown that artificial datasets, such as the swiss-roll, show that linear methods have a more challenging time finding the intrinsic dimensionality of the data than nonlinear methods [17].

According to Laurens [16], who compares the performance of linear and nonlinear dimensionality reduction methods, there is a tendency for real-world data to be nonlinear. The linear methods should have a disadvantage because they cannot capture the intrinsic dimensionality of the nonlinear data and nonlinear methods. However, the research paper states that nonlinear methods "are often not capable of outperforming traditional linear techniques" [16].

In this project, the focus will be on the impact of linear and nonlinear methods on the performance of a ML model implemented in this project. The differences between the methods used in this project will be presented in section 3.5.

2.4 Problem statement

This project aims to explore the impact of dimensionality reduction, comparing linear and nonlinear dimensionality reduction techniques. This will be done in regards to the performance of a machine learning model, for the problem of image classification and recognition, of the MNIST dataset.

Chapter 3

Theoretical considerations

This chapter will discuss the theory behind ML and FE. It will briefly introduce a sketch of the steps in how the theory will be implemented, then describe the theory behind ML and FE, and then go into more detail about the different methods used in this project on the thoughts behind why choosing them.

3.1 Images

In this project, the MNIST dataset has been chosen because it is commonly used in image recognition tasks, specifically for recognizing handwritten digits. This dataset is well-documented, which allows for comparing the results to similar projects. It is small in size, making it easy to work with and not requiring a lot of computational power. MNIST has 60000 training samples and 10000 test samples [4].

It was considered to use the Iris and Canadian Institute For Advanced Research (CIFAR-10) datasets, but the Iris dataset may have too few data samples [18], and the CIFAR-10 dataset could be too complex for this project [19]. The MNIST dataset consists of images, each composed of pixels. In a grayscale image, each pixel has a value between 0 and 255, with 0 representing white and 255 representing black [4].

This project aims to demonstrate dimensionality reduction by representing each pixel as a feature in the image. As the size of the images increases, this quickly becomes a large number of features.

3.2 Machine Learning

Understanding the basics of machine learning is important to determine the appropriate dimensionality reduction methods to compare for the project. This section will describe the basics of machine learning by describing a simplified model of a ML model pipeline.

3.2.1 Machine learning pipeline

Figure 2.1 shows the simplified and generalized steps in the pipeline of a machine-learning model. The arrows represent the flow ML through the pipeline. The model is trained on the training set and then evaluated on the validation set. The model then updates with the new information, and the process repeats.

This loop is called the training loop. The training loop repeats until the model converges or until the model is no longer improving. The model gets evaluated on the test set, and the evaluation gets used to determine the model's performance. The reason for having a pipeline for a ML model states as follows:

> A machine learning pipeline (or system) is a technical infrastructure used to manage and automate ML processes in the organization. The pipeline logic and the number of tools it consists of vary depending on the ML needs. But, in any case, the pipeline would provide data engineers with means of managing data for training, orchestrating models, and managing them on production. [10]

For this simplified example, the machine learning pipeline shown in this report has four main steps: data collection, feature engineering, model training, and model Evaluation. This model is not entirely identical to the model shown in [10], as this is a simplified model with slight modifications to fit this project's scope. The four steps will get described in the following subsections.

Data collection

The `data` box in Figure 2.1 represents the collection of data to be used in training the machine learning model [10]. There are many different types of data, but some of the most commonly used, as stated by [20], are Numerical Data, Categorical Data, Time Series Data, and Text Data. Often the data is in some type or format which is not directly usable by the machine learning model. For this, the `feature engineering` step is used.

Feature engineering

`Feature engineering` represents the step where the data gets transformed through for example dimensionality reduction. In [10], `Data preparation` and `feature engineering` are both in a step called `prepare data`. For this model, `feature engineering` covers those steps.

`feature engineering` is also the step where data is preprocessed [10]. Preprocessing is done to ensure quality data [21], and feature engineering is done, among other reasons, to improve the quality of the results, and to improve the performance of the machine learning model, by reducing the burden on the machine learning algorithms [22].

Model

`model` is where the process of training the model with the data takes place. Model training splits the data into a training set and a validation set. An example of model training gets seen in further detail in the model shown in [10], at the `Split data` step. As stated in 3.8, it is important to split the data in this manner, as it helps to reduce the risk of overfitting.

Evaluation

`evaluation` represents the step where the model trained on the training set, gets evaluated on the validation set. The evaluation compares the model's predictions with the actual values. The model's predictions get evaluated on the validation set, and this step can repeat multiple times [10]. Using accuracy, precision, recall, and F1 score metrics helps provide information regarding the performance of the model [23].

Parameters

`Parameters` represents the step where hyperparameters of the machine learning model get set and tuned. The algorithm sets some parameters through training, then there are hyperparameters, which are parameters given to the algorithms to train the model [24]. This step is purely for hyperparameters.

As will be explained in section 3.7, there are multiple ways to tune hyperparameters. The main reason this tuning is important is that it helps improve the model's performance and reduce the risk of overfitting [25].

3.3 Classification

When using a model, it is only sometimes possible to predict the correct value of a variable. For example, if one wants to predict a house's price, one can not predict the exact price, but may be able to predict if the price is high or low. This is called classification.

In ML, classification is when a quantitative answer is not required; a qualitative answer is satisfying. The goal is to classify the input into one of a set of categories set in the dataset. As long as the dataset supports supervised learning, classification can be applied to it. This is done by training a model on a labeled dataset and then using the model to predict the category of new, unlabeled data [3].

On the other hand when it is possible to predict the value of a variable it is in machine learning called regression. Regression is a type of supervised learning where the goal is to predict a continuous numerical value, such as the price of a house. In contrast to classification, where the output is a discrete category, regression models predict a numeric value based on the input data. Regression can be applied to a dataset as long as the dataset supports supervised learning and contains continuous numerical values [3].

To train a regression model, a labeled dataset is used, and then the trained model is used to make predictions on new, unlabeled data. This allows for the prediction of the value of a continuous variable based on the patterns learned from the training data [3].

The training data consists of input and class label pairs. The input can be a single value, such as numbers or strings, or vectors of values, such as a list of numbers or strings. The input can also be a matrix of values, such as a grayscale or color image. The class label is a discrete value, such as a number or a string. In this project, a class label is a number representing the digit in the image [3].

The model is trained to minimize the error between the predicted and actual class labels. The error is calculated by comparing the predicted class label with the actual class label [3].

The model is trained by finding the best parameters for the model, such as weights in a Neural Network (NN) or parameters in models, such as SVM [3]. These parameters help the model predict the category of new data. Hyperparameter optimization is finding the best parameters; this is discussed further in section 3.7.

One must consider the data and the problem when deciding which model to use. The model must be able to handle the data and the problem. In this project, the ML model is used for image classification, as the MNIST dataset is used.

Several ML models have been used for image classification with MNIST in general; the group has looked into are [4, 26–28]. These models can be used to classify images into one of ten categories, representing the digits 0-9, as this is the project's focus.

Support Vector Machine

SVM is a supervised ML model that can be used for classification or regression problems. It is a linear model for classification and regression problems. It is a binary classifier, meaning it can only classify two classes. It is a linear classifier, meaning it makes a decision based on a linear function of the input features [3].

SVM classifies data by finding a mapping (hyperplane) that separates the classes in data [29]. SVM is also known as a large-margin classifier, which means that it relies on finding "a maximum-margin hyperplane to separate classes" [29]. As the name implies, SVM uses support vectors, which are the 'vectors' closest to the function defining the mapping, and alterations on those data points will influence the hyperplane.

An exciting property of SVM is that it can have, among other parameters, a kernel function, which can be tuned whether the data is linearly separable or not [29].

For the project's purposes, SVM is a promising model since

> SVMs have been shown to perform well in a variety of settings and are often considered one of the best "out of the box" classifiers. [3]

This is because SVM is a linear model, which means it is fast to train and predict. This is important for the project, as the model needs to classify the images quickly.

The model also has a high accuracy, which is vital for the project, as the model needs to classify the images correctly. Though it is not as accurate as other models, such as NN, it is still a good model for the project [4].

On the downside of SVM: "Though it is elegant and simple, we will see that this classifier, unfortunately, cannot be applied to most data sets, since it requires that the classes be separable by a linear boundary" [3]. It means there can be some errors in the ML model, as it is only sometimes possible to separate the classes in MNIST by a linear boundary; this is something the group has considered choosing the model.

This project is not mainly concerned with the choice of ML model but rather with the choice of dimensionality reduction methods. Therefore, SVM is chosen as the ML model as it has already been used with MNIST without dimensionality reduction [4].

3.3.1 Multi-class classification

In classification, there are two types of classification, binary classification, and multi-class classification. Binary classification is the classification of two classes, while multi-class classification is the classification of more than two classes.

The MNIST dataset used in this project presents a multi-class classification problem, as the images can represent any of the ten digits. The SVM model, however, is a binary classification model and thus has to be adapted to the multi-class classification problem. there will explained two approaches to this problem: One-versus-One (OvO) and One-versus-All (OvA) [3].

One-vs-One

OvO is a method where the model trains on all possible combinations of two classes. For example, if there are five classes, the model is trained on ten different models, one for each combination of two classes; this makes it computationally expensive as it has to go through every combination. Then the model is evaluated against all other models, and the class with the highest score is picked as the predicted class [3].

One-vs-All

In machine learning, the OvA (also known as one-vs-rest) method is a strategy for learning multiclass classifiers. In this approach, a separate binary classifier is trained for each class, with the samples of that class labeled as positive and all other samples labeled as negative. During inference, the classifier that produces the highest confidence score is chosen as the prediction for a given sample. It can be seen as a generalization of the OvO method, which trains a binary classifier for every pair of classes [3].

One-vs-One vs One-vs-All

OvO is chosen as the method for multi-class classification. This is because in scikit-learn (sklearn) library, OvO is the default approach for multi-class classification using the SVM model.

3.4 Preprocessing

In this section, the theory of preprocessing and what effects it has on the ML model is discussed.

3.4.1 Definition of preprocessing

The definition of preprocessing or data preparation varies depending on the source. The explanation that will be used in this report is given as follows:

> Data preparation comprises those techniques concerned with analyzing raw data so as to yield quality data, mainly including data collecting, data integration, data transformation, data cleaning, data reduction, and data discretization. [21]

From this, it can be gathered that preprocessing is a broad term that can be divided into several subcategories. Not all subcategories will be relevant to this project; therefore, the following sections will only discuss appropriate subcategories of preprocessing.

3.4.2 Reasons for preprocessing

As stated in the definition, preprocessing is used, among other reasons, to yield quality data. The importance of this stems from the fact that real-world data is only sometimes clean or complete. Meaning that there can be a lot of noise, which is data containing errors or outliers. This noise can be removed or reduced by preprocessing, which creates a more accurate, higher quality, and smaller dataset to gather information. This results in a reduced amount of data that the model is trained on, but the data it is trained on should be more accurate, which should then train the model more accurately and efficiently [21].

Data collected may be in a form or shape that is not compatible with the process that is needed to work with it. Therefore, to use the data, it may need to be transformed into a form that fits with the process. Transformation of data can be many things, as [30] mentions normalization as a part of the transformation, to scale the data so that it fits into a new range, Subsection 3.4.3 will give a more detailed explanation of normalization.

3.4.3 Steps of preprocessing

The amount of preprocessing depends on what is needed and what is available. In [30], these steps are shown; they start by cleaning data, transforming it, reducing it, and balancing it. This section will explain the theory of the steps used in this report.

Normalization

Normalization, also called scaling, is the process of scaling the data. This is done by changing the range of the data; for example, if the data is in the field of 0-100, it can be scaled to be in the range of 0-1. This is done to make the data more suitable for the model and to make it easier to compare the data. Scaling is also a standard practice in most ML problems. There are many ways to scale the data; for example, there is a min-max scaler [12].

Min-max scaler The min-max scaler is a method that transforms the data to be in the range between 0-1 by subtracting the minimum value and dividing by the scope of the data. The formula for this is:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.1}$$

Where x is the original value, x_{scaled} is the scaled value, x_{min} is the minimum value in the data, in the case of this project, this will be 0, and x_{max} is the maximum value in the data, in this project's case, this would be 255.

The min-max scaler is a simple way to scale the data, but it is not robust to outliers. If there are outliers in the data, the min-max scaler will scale the data to be in the range of 0-1, but the outliers will be scaled very close to 0 or 1. While other data will be clustered together, this can be a problem if the outliers are essential to the model. The min-max scaler is also sensitive to the presence of zeros in the data. If there are zeros in the data, the zeros will remain zero, and this can be a problem if the zeros are essential to the model, however for this project, this will not be a concern, as this does not affect the model [12].

Variance scaling Variance scaling is a more complex way to scale data. The variance scaling is a method that transforms the data to have a mean of 0 and a variance of 1, by subtracting the mean and dividing by the standard deviation. The formula for the variance scaling is:

$$x_{scaled} = \frac{x - mean(x)}{\sqrt{var(x)}} \tag{3.2}$$

Where x is the original value, x_{scaled} is the scaled value, $mean(x)$ is the mean of the data, and $var(x)$ is the variance of the data [12]. This makes it a more robust scaling than min-max scaling.

3.4.4 Choice of Preprocessing

Following the theory of preprocessing from 3.4, this section will cover the decisions made for preprocessing the data and how there were taken.

As stated in section 3.4.3, the amount of preprocessing needed varies depending on what is needed for the model. For this project, it was deemed sufficient to reshape the data and normalize it. A detailed description of each decision can be found in the following sections.

Reshape

This subsection describes the decisions made regarding reshaping the data loaded from the MNIST dataset. This subsection only describes the decisions for the training data as an example, but similar decisions were also made for the test data.

The data, when loaded in at first, is shaped as a long array of 47.040.000 integers, ranging from 0-255, representing the values of the pixels in all 60.000 images, and then another array of 60.000 integers representing what number a given image is, between 0-9. The images are in a 28x28 matrix, meaning that the first 784 integers represent the first image, and the successive 784 integers represent the second image.

Additionally, the function used from sklearn did not accept the data in this format because the function `fit` expects: "X: array-like of shape (n_samples, n_features)" [31], which means that the function expects the data to be in array of samples(images), and an array of features(pixels). Because of the input `fit` expects, it was necessary to reshape the data into a 60.000x784 matrix. From this, there was a clear distinction between each image, and the data could easily be used in the functions given by sklearn.

Normalization

In subsubsection 3.4.3, two methods of normalizing data are described: min-max scaler and variance scaler. From these two methods, both are chosen. The combination of normalization is called StandardScaler, and is a commonly used method of normalizing data, it was deemed a good choice. Variance scaler also ensures that it is comparable to prevent bias in the model [32]. Both methods would have been good choices for normalization, variance scaler was chosen, but either would be acceptable for this project.

3.5 Dimensionality reduction

In general, dimensionality reduction transforms data with many features (dimensions) into a new representation of the data with fewer features while preserving as much relevant information as possible. Dimensionality reduction is finds a transformation of the data that maps the data to some lower dimensional space, while keeping the mean distances between the data points [16].

Dimensionality reduction can be divided into two categories: linear and nonlinear methods, described in Section 2.3. This section presents some of the standard methods from both categories.

3.5.1 Linear methods

The linear methods covered are: PCA, Factor Analysis (FA), and Linear Discriminant Analysis (LDA).

Principal Components Analysis

PCA is a linear method used to reduce a dataset's dimensionality by projecting it onto a lower dimensional subspace, retaining as much of the variance as possible. The best projection is found through the directions of maximum variance in the data based on the covariance matrix and then projecting the data onto those directions. The directions of maximum variance are principal components, and the projection results from the PCA [16].

Factor Analysis

FA is very similar to PCA, and PCA may be called a type of FA. However, FA is based on the assumption that the data is generated by a linear combination of a few latent variables called factors and that the observed variables are a linear combination of the latent variables plus some noise [33].

The goal of FA is to find the factors that explain the most covariance in the data. Intuitively, related items have stronger mathematical correlations and can thus be grouped with less loss of information. Once the correlations are determined, a rotation is performed to make the factors easier to interpret FA [33].

Linear Discriminant Analysis

LDA is quite similar to PCA since they both try to project data into a hyperplane. Still, instead, LDA tries to maximize class separability, making it easier to contrast classes. LDA finds the hyperplane with the most separation between the classes and keeps the data points with the same class as close together as possible. This is a three-step process. The first step is to determine the separation between different classes (between-class variance) as the distance between the means of each class. Secondly, the distance between the mean and samples of each class (within-class variance) is calculated. The third step is creating a lower dimensional representation that maximizes the between-class variance and minimizes the within-class variance [34].

3.5.2 Non-linear methods

Sometimes high-dimensional data may contain non-linear relationships, which linear methods cannot capture. It has been shown that PCA fails to find significant components in the swiss-roll dataset [17]. In such cases, non-linear methods are used. The non-linear methods covered are kPCA and Isometric Feature Mapping (ISOMAP).

Kernel Principal Component Analysis

kPCA is an extension of PCA, and it projects the data onto a higher dimensional plane, where PCA is performed. The first step is to construct the kernel matrix from the data. The kernel matrix is a matrix of the dot products of the data points, the kernel matrix is used like the covariance matrix in PCA [35].

In the kernel matrix the data is in a higher dimensional space, where the data is linear separable. By using the kernel function, kPCA exposes the kernel trick, which is: instead of calculating the data into a higher dimensional space, it calculates the inner product between the datapoints, which is computationally cheap to calculate instead of calculating every datapoint into a higher dimensional space [35].

The kernel matrix is then centered by subtracting the mean of each row and column. This matrix is also called the Gram matrix. The Gram matrix is then used to find eigenvectors and eigenvalues. By using the kernel trick, kPCA can compute eigenvalue decomposition, as in PCA. The eigenvectors are then used to project the data onto a higher dimensional space, where PCA is performed. The eigenvectors with the highest eigenvalues are the principal components, and the projection is the data projected onto the principal components [35].

Isometric Feature Mapping

ISOMAP is another non-linear method that is a special case of Classical Multi Dimensional Scaling (cMDS). It is assumed that ISOMAP is suited to discover manifolds of arbitrary dimensionality and that it guarantees an approximation of the true structure of the manifold [17].

The first step in ISOMAP is to map the data points in a graph. The graph is constructed by connecting each point to its nearest neighbors. The user sets the number of nearest neighbors. The nearest neighbors are found by using the Euclidean distance. The Euclidean distance is the linear distance between two points in a Euclidean space [36].

In the second step, ISOMAP finds the Geodesic distance between each point [36]. The Geodesic distance is the shortest distance between two points on the surface of a sphere. The Geodesic distance is calculated by finding the shortest path between two points on a graph. The graph is constructed by connecting each point to its nearest neighbors. The shortest path can be found by using the Dijkstra algorithm [37].

The final step in ISOMAP is finding the data's Multi Dimensional Scaling (MDS) projection. The MDS projection is found by minimizing the stress function. The stress function is the sum of the squared differences between the distances in the original data and the distances in the projected data [37], so the stress function finds the slightest change in the data. The minimum of this stress function will be the best reproduction of the data in lower dimensional space according to the ISOMAP algorithm.

3.5.3 Choice of dimensionality reduction

Some dimensionality reduction methods were presented in the Section 3.5. Due to the project's limited scope, not all of the dimensionality reduction methods will be chosen in the implementation. The implementation of the methods will be based on sklearn's implementation of the methods. The hyperparameters of the respective methods will also be presented.

Linear methods

The reason for choosing PCA is because it is a popular dimensionality reduction method [16] and because it tries to find a linear embedding that retains as much information as possible from the original data, which is done by choosing a k components.

A difference between PCA and FA is that FA, as described in 3.5.1, further assumes that the linear combinations have some noise. The hyperparameters for the PCA method are the number of components and whether the components will be whitened or not. According scikit-learn, whitening "can sometime improve the predictive accuracy of the downstream estimators" [38].

LDA was also chosen because of its ability to project the data by maximizing the separation between classes. Furthermore, the number of dimensions that would be reduced on MNIST would correspond to the number of $dimensions - 1$ [38, 39]. Such a property proves helpful because it simplifies the classification task and may also improve the performance of the machine learning model.

The fact that the number of dimensions reduced for MNIST will be maximally about the same as the number of classes might provide a good starting point for PCA and LDA to be compared. The comparison can be based on how much information they can retain by reducing the data to nine dimensions.

Nonlinear methods

kPCA was chosen because of its ability to project the nonlinear data onto a hyperplane where PCA can be used. That fact leads to the possibility of using kPCA with the hyperparameters of PCA, and kernels. Additionally, some kernels will have a kernel coefficient, which can be thought of sensitivity of the kernel. Another reason for choosing kPCA is that it uses PCA, so the differences between linear PCA and nonlinear PCA could be compared.

ISOMAP is another nonlinear dimensionality reduction method that takes another approach to reduce the dimensions of nonlinear data. Instead of using a kernel, ISOMAP constructs a graph of the data and then applies cMDS to the data. ISOMAP also tries to preserve the distances between the data points as much as possible [16]. Therefore, the hyperparameters used for ISOMAP are the number of components and the number of neighbors used to construct the graph.

The group has also worked with other nonlinear methods, such as T-SNE. However, it was ultimately not chosen because it is a method that is mostly used for data visualization [40].

This section has presented the dimensionality reduction methods that will be used in the implementation. The methods are the following: PCA, LDA, ISOMAP, kPCA, where the number of components is the hyperparameters for the methods.

3.6 Examples of methods

This section will present examples of how the linear and nonlinear methods behave on linear and nonlinear data. It should be noted again that the methods have different purposes. As a quick reminder, PCA tries to maximize the variance in the embedded data, and so does kPCA. In contrast, LDA tries to maximize the separability of classes in the data, and ISOMAP tries to preserve the distances from the high-dimensional data when projecting it onto a lower space. Therefore their results will not be a one-to-one comparison. All of the methods used in this section are implemented from sklearn [39].

3.6.1 Linear data example

As linear data the Iris dataset is used, which contains three different species of Iris flowers, with 50 examples each. Each class has four dimensions: the length and width of the sepals and petals [41]. The dataset is a simple and well-known dataset, which has been used in many machine learning papers [41] and should give intuition as to how the methods behave on the dataset.

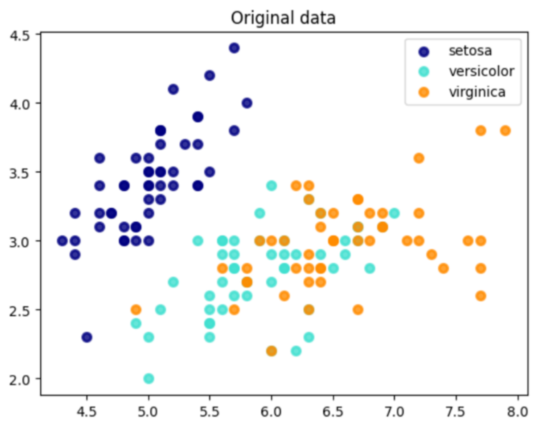
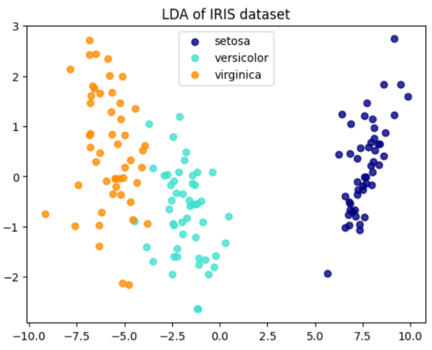


Figure 3.1: Linear data as Iris dataset

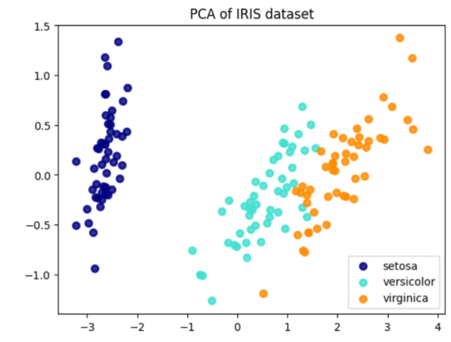
Figure 3.1 represents the iris flowers on a 2D plot. According to [41], only one class is separable, which can further be solidified by the fact that the species Setosa is the only visually linearly separable class.

Linear methods

In Figure 3.2a, LDA has also successfully separated the Setosa species from the other classes in the dataset. In contrast with PCA, LDA has managed to separate somewhat better as the distinction between those classes can be made more accessible. An advantage that LDA poses is that LDA is a supervised method, which means that LDA knows the labels of the data and may be better suited for classifying the data.



(a) LDA on iris



(b) PCA on iris

Figure 3.2: both linear methods on iris

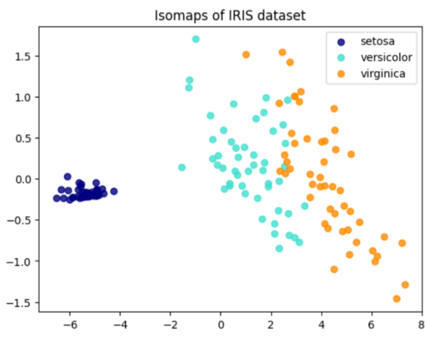
In figure 3.2b, one can see that PCA has successfully separated the Setosa species from Versicolor and Virginica in the dataset. The other species are not separated on the coordinates [1.20, -0.20]. Having only Setosa separated from Versicolor and Virginica in the dataset is not concerning, as the test on the Iris dataset is to separate Setosa from the other classes by linear separability.

Nonlinear methods

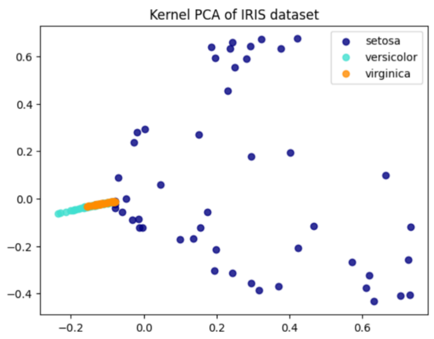
In Figure 3.3a, ISOMAP has tried to separate the Setosa and map the data on the first two dimensions. ISOMAP, as opposed to the aforementioned linear methods, has found little diversity on the second projection, as the values range from approximative 0.0 to -0.25. Such a range is short in comparison with the linear methods.

ISOMAP clusters the Setosa species, but that may not be helpful if the data needs to be projected on two dimensions. If, for example, a point is in the upper left corner, it is impossible to tell if it is a Setosa or a Versicolor. ISOMAP is nevertheless a nonlinear method, and it is expected that it does not reduce the data as efficiently as the linear methods.

In figure 3.3b, kPCA has tried to separate the Setosa. In the figure, it can be seen that kPCA has clustered the other two species on the first projection, but there cannot be any clear separation between them, especially on the second projection. kPCA has also mapped the data points for Setosa with a high degree of variance, but it did not separate the Setosa from the other two species. Likewise, ISOMAP and kPCA were not supposed to separate the data and the linear methods. However,



(a) Isomap on iris



(b) KernelPCA on iris

Figure 3.3: both nonlinear methods on iris

it is interesting that kPCA has yet to separate the Setosa species from the rest of the data.

3.6.2 Nonlinear data example

As nonlinear data, two classes of circles are constructed, an inner- and an outer circle, which will look like in Figure 3.4.

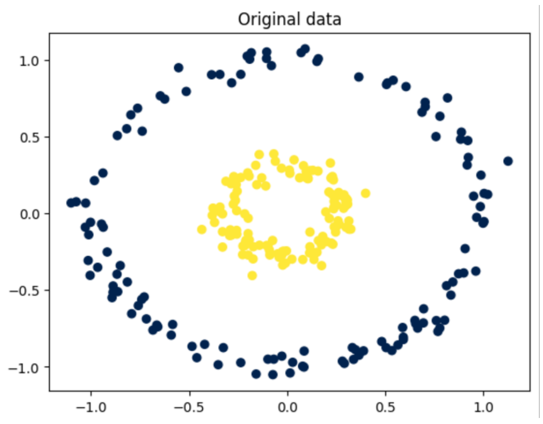


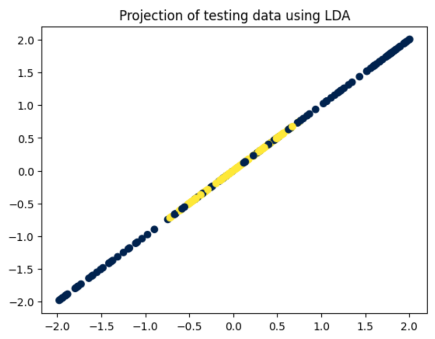
Figure 3.4: Nonlinear data as two circles

Linear methods

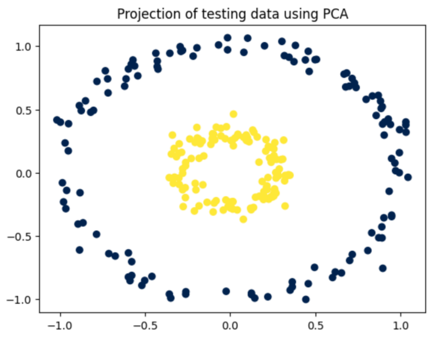
In Figure 3.5a, LDA has reduced the data's dimensions to one dimension, but the two classes are clustered, which means that LDA could not separate the two classes. In Figure 3.5b, it can be seen that PCA's transformation could have managed to map the nonlinear data.

Nonlinear methods

In Figure 3.6a, ISOMAP has separated the data points from the circles. Based on the first projection ISOMAP is capable of separating the data. On the second projection,



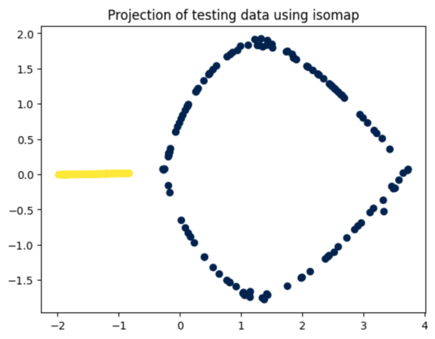
(a) LDA on circles



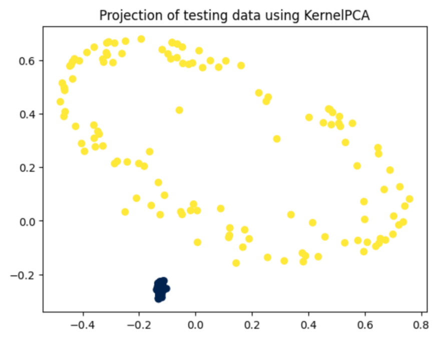
(b) PCA on circles

Figure 3.5: both linear methods on circles

the method can capture more information about the outer circle, thus approximating the original data. The outer circle has a higher variance than the inner circle. The second projection, ISOMAP, revealed little information about the inner circle.



(a) Isomap on circles



(b) KernelPCA on circles

Figure 3.6: both nonlinear methods on circles

In Figure 3.6b, kPCA has also separated the data points from the circles. kPCA does an excellent job of maximizing the variance for the inner circle and separating the data points from each other. As can be seen, kPCA needs at least two dimensions to better differentiate between the two classes, unlike ISOMAP, which only needs one dimension.

Until now, we have presented some simple examples of the methods. More often than not, the number of reduced dimensions will not be two, as there will be much more relevant information in the other dimensions.

3.7 Hyperparameter optimization

This section introduces the concept of hyperparameter optimization and the importance of this process in machine learning.

3.7.1 Hyperparameters

Hyperparameters are the parameters set, for algorithms, before training the model, which does not get learned from the data. The values of hyperparameters can have a significant impact on the performance of the model. How hyperparameters differ from model parameters is that those model parameters get learned from the data during model training [42].

Hyperparameters exist for both FE and for ML models, examples of these could be SVM and PCA. Here PCA should at least have the hyperparameter which corresponds to the amount of dimensions it should reduce down to. SVM would have hyperparameters like which kind of kernel it should use [42].

3.7.2 Methods of optimization for hyperparameters

Usually, when selecting hyperparameters for a given algorithm, users can resort to default values based on the algorithm's documentation, read literature for recommendations, or try different values and see which one works best. However, this approach could be more efficient, as it is time-consuming and requires a lot of manual work [42].

Instead, the process of finding optimal hyperparameters can be given to the computer [43], given a set of configurations. Hyperparameter optimization is complex because it is unknown which hyperparameters will significantly affect the model, which hyperparameters will interact with each other, and how their interactions will change the model's performance. According to Marc Claesen [44], the number of hyperparameters that have a significant impact may be small, but that does not mean that the number of meaningful combinations may be small too.

Many different techniques can get used to optimize the hyperparameters automatically [43]. An example of such a technique is grid search. This technique allows the user to define "a set of finite values for each hyperparameter, and grid search evaluates the Cartesian product of these sets" [43].

Another example of a method is random search, a technique similar to grid search. However, instead of evaluating all the combinations of hyperparameters, it randomly evaluates hyperparameters, given a limited amount of time. Both methods have limitations; grid search is inefficient when the number of hyperparameters is significant due to the curse of dimensionality, which means that for algorithms that require large amounts of hyperparameters, it is not feasible to use this technique [45].

Random search can prove helpful, but there is no guarantee that it will find the optimal hyperparameters, given its limited time to evaluate them. Knowing the optimal time limit for random search is tricky, as it depends on the number of hyperparameters, the number of possible values for each hyperparameter, and the number of times the hyperparameters are evaluated [45].

From this, it can be concluded that no single method is optimal for all cases. The optimal method depends on many variables, such as the number of hyperparameters and possible values for each hyperparameter. Therefore, it is essential to evaluate the different methods and find the one that works best for the given problem. Of course, many other techniques could be discussed. Nevertheless, these two methods should sufficiently demonstrate the strengths and weaknesses of different techniques.

3.7.3 Choice of hyperparameter tuning

Choosing parameters is necessary when working with models and with FE. In this pipeline, in particular, there are many parameters to tune because it has both model and FE. Tuning the parameters would take very long due to not only having to pick different model parameters and try each of these configurations with different FE, which increases the time exponentially.

The pipeline implements hyperparameter tuning, specifically using grid search to solve this. This hyperparameter tuning using grid search makes the tuning process more effective and avoids missing exemplary configurations due to human error.

3.8 Cross-validation

This section will describe cross-validation, why it is used in general and why it is used in this project.

A common problem in machine learning is that the test set is supposed to be used only in the final evaluation. This problem makes it hard to, for example, hyperparameter tune or chooses the correct model for the data. On the other hand, if the test data is used to tune or choose a model, this often leads to overfitting [46].

Overfitting can be described as when the model is very good at predicting the test data but will not do well with new data, not from the test set. This section will introduce one technique that will reduce the chances of overfitting the model: cross-validation.

Cross-validation is a technique that splits the training data into two sets: new training data set and a validation data set comprised of training data. By partitioning the training data into two sets, the model can learn from the training data and evaluate on the validation data [47].

The model can be tuned with the validation data and evaluated on the test data after the model is optimized. Such an approach is practical because the model gets evaluated based on data it has never seen, which shows how good the model is at predicting data it has not yet seen [47].

Cross-validation is particularly useful with hyperparameter tuning because it allows picking the correct parameters without using the test set. Cross-validation, more practically can be described as

> The basic form of cross-validation is the basic form of k-fold cross-validation. (. . .) In k-fold cross-validation, the data is first partitioned into k equally (or nearly equally) sized segments or folds. Subsequently, k iterations of training and validation are performed such that within each iteration a different fold of the data is held-out for validation while the remaining $k - 1$ folds are used for learning. [48]

There are also exists other forms of cross-validation, which are special cases of k-fold. As described before, shuffling the data might be necessary, and there is a version of k-fold called Stratified k-fold, where the data gets split into equal sets in regards to the total of each class for each round. Alternatively, instead of stratifying the data, one can use the leave-one-out cross-validation, where for each fold, all of the data except one sample is used for training [48].

Stratifying or shuffling the data may be necessary so that each fold has a balanced distribution of classes. There is a risk that the folds contain imbalanced samples for each class, which may affect the overall model performance because the model is biased towards the majority class [48].

In order to avoid overfitting, one can use k-fold cross-validation or derivatives of it, but k-fold implies that a specific number must be chosen. The choice of k could be made through trial and error. However, the train and data samples need to be large enough to be statistically representative of the data set [48].

3.8.1 Choice of cross-validation

Cross-validation is a part of the pipeline and the project in general because it not only allows hyperparameter tuning without the test set but also because it allows for exploration of data while keeping the test set clean [46]. Specifically stratified k fold cross validation is chosen.

3.9 Evaluation and metrics

A pipeline is generally evaluated based on quantitative metrics in machine learning and data science. These metrics are in classification general based on the confusion matrix. An example of a confusion matrix can be seen in Figure 3.7.

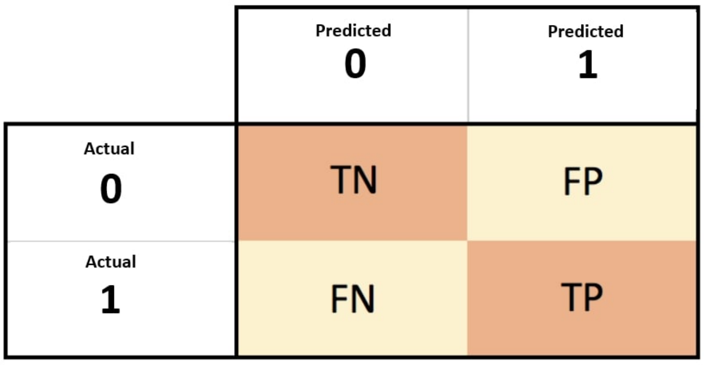


Figure 3.7: Confusion matrix [49]

Figure 3.7 has the values: `TN`, `FP`, `FN`, and `TP`, which stand for True Negative, False Positive, False Negative, and True Positive, respectively. A confusion matrix represents all an ML model's guesses, with the number of guesses placed at the corresponding label where the correctly predicted elements are on the diagonal, in this case `TN` and `TP` [46].

3.9.1 Metrics

There are many different metrics, but there are generally four basic ones. These are accuracy, precision, recall, and F1. Outside these, there are more esoteric metrics such as the Mattheus correlation coefficient, Cohen's kappa, and more [50]. The following section will describe each of the four basic metrics.

Accuracy The most straightforward and simple of the metrics is accuracy, as it simply describes the percentage of correct guesses out of all guesses. Accuracy works well in cases where the sizes of the different classes are similar but struggles in cases where one class is much larger than another. In a case where one class is much larger than others, it is possible to achieve very high accuracy by only guessing the larger class, as the larger classes will have a higher weight when compared to the smaller classes [50]. The formula for accuracy can be seen in Equation 3.3.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

Precision In the case where one class is smaller when compared to the other classes, the precision metric is better suited than accuracy. Precision is an excellent metric to use when the cost of false positives is high, as it measures how many of the positive guesses were positive. Precision can also measure how well the model can be trusted when predicting a class [50]. The formula for precision can be seen in Equation 3.4.

$$Precision = \frac{TP}{TP + FP} \tag{3.4}$$

Recall The third metric is recall, which measures the model's ability to find all the positive samples. In the case of this project, positive samples refer to correctly guessing numbers from the MNIST dataset. [50]. The formula for recall can be seen in Equation 3.5.

$$Recall = \frac{TP}{TP + FN} \tag{3.5}$$

F1 The last basic metric is F1. F1 is commonly used when both recall and precision are essential; F1 can be considered a weighted average. The formula for F1, is shown in [50]. The formula for f1 can be seen in Equation 3.6.

$$F1 - Score = \frac{2}{precision^{-1} + recall^{-1}} = 2 \cdot (\frac{precision \cdot recall}{precision + recall}) \tag{3.6}$$

These metrics generally get used in a pipeline for hyperparameter tuning and general evaluation of the model's performance. For this project, hyperparameter tuning will only use a single metric to focus on to maximize it. In evaluation, the metrics generally explain what the model does well and does not do well [46].

3.9.2 Choices of evaluation metrics

As mentioned in subsection 3.9.1, many different metrics can be used to evaluate a model. The basic metrics used in this project were accuracy, precision, recall, and F1. These metrics each have their strengths and weaknesses. Some of these strengths and weaknesses of the metrics were described in subsection 3.9.1.

Another metric used to measure the performance of the models was the time it took to train them. Depending on the circumstances, the time it takes to train a model could be just as important as the accuracy of the model, depending on the loss of accuracy. Such as, if a model takes 10 minutes to train but only loses 1% accuracy, it might be worth using that model compared to another model that takes several hours or days.

For the pipeline of this project, the metric used to evaluate the models was f1-score. The reason is that accuracy would not be a great indicator on its own, since the training data contains more examples of some digits than others. F1-score is a good metric to use in this case, since it is a weighted average of precision and recall. This means that it is not as affected by the imbalance of the training data as the other metrics.

3.10 Summary of theoretical considerations

In this chapter the theory and theoretical choices have been presented with respect to the elements in the pipeline. This section will summarize these choices.

As data, MNIST will be used. The preprocessing on MNIST will consist of reshaping the data and normalizing it with min-max and variance scaling. The features that will be extracted from the data will be extracted by the dimensionality methods PCA, LDA, kPCA, and ISOMAP. The reduced data will be used to train a SVM model. The SVM model will be evaluated with the metric f1 score. The SVM model will be tuned with the cross-validation method grid search. The implementation will be based on sklearn.

Chapter 4

Implementation

This chapter covers the implementation details for the project. The implementation is created in python 3.8.10 and uses the scikit-learn (sklearn) library for the machine learning algorithms.

The chapter starts with an overview of the pipeline used for the project based on the considerations from chapter 3; this is followed by the essential implementation details and the machine learning algorithms, from pre-processing to hyperparameter tuning.

4.1 Pipeline overview

The development for this project is an extension of the general pipeline from Figure 2.1 extended with the considerations from chapter 3.

Figure 4.1 shows an overview of the developed pipeline, which consists of five segments: The dataset, the pre-processing, the dimensionality reduction, the machine learning loop, and the output.

The first segment, the dataset, covers downloading the dataset, loading it into the program, and augmenting the original dataset to create a new one.

The second segment, pre-processing, covers reshaping the data into a form usable with the sklearn library and scaling the data in preparation for the next segment.

The third segment, and the focus of this project, is dimensionality reduction. This segment consists of the four dimensionality reduction algorithms: PCA, LDA, ISOMAP, and kPCA, as well as no dimensionality reduction.

The fourth segment is the tuning loop, where grid search with cross-validation determines the best hyperparameters for the SVM based on the f1-score. Because the project focuses on dimensionality reduction, this loop includes the number of components for dimensionality as a hyperparameter as well.

Finally, the fifth segment is the output. Once the best hyperparameters are determined, the SVM is trained on the whole dataset and tested on the test set. The results of the grid search - scores, parameters, and scoring time - and the results of the final SVM - confusion matrix and classification report - are saved.

4.2 Pipeline implementation

This section presents the project's implementation details per the descriptions from section 4.1.

4.2.1 Data and pre-processing

The MNIST dataset is downloaded from `http://yann.lecun.com/exdb/mnist/`[4] and loaded into the program using the idx2numpy library. It was later discovered that the sklearn library has a built-in function for downloading the MNIST dataset, but this was not discovered until after the dataset was downloaded manually.

The data is comprised of four parts. The training images (`X`), training labels (`y`), test images (`X_test`), and test labels (`y_test`).

```
load_mnist() # helper function for downloading the MNIST dataset
X = idx2numpy.convert_from_file(
'src/mnist_data/train_file_image').reshape(60000, 784)
y = idx2numpy.convert_from_file('src/mnist_data/train_file_label')
X_test = idx2numpy.convert_from_file(
'src/mnist_data/test_file_image').reshape(10000, 784)
y_test = idx2numpy.convert_from_file('src/mnist_data/test_file_label')
```

Listing 4.1: Data segment of pipeline details.

Each image is reshaped from a 28x28 matrix to a 784x1 vector, which is the input

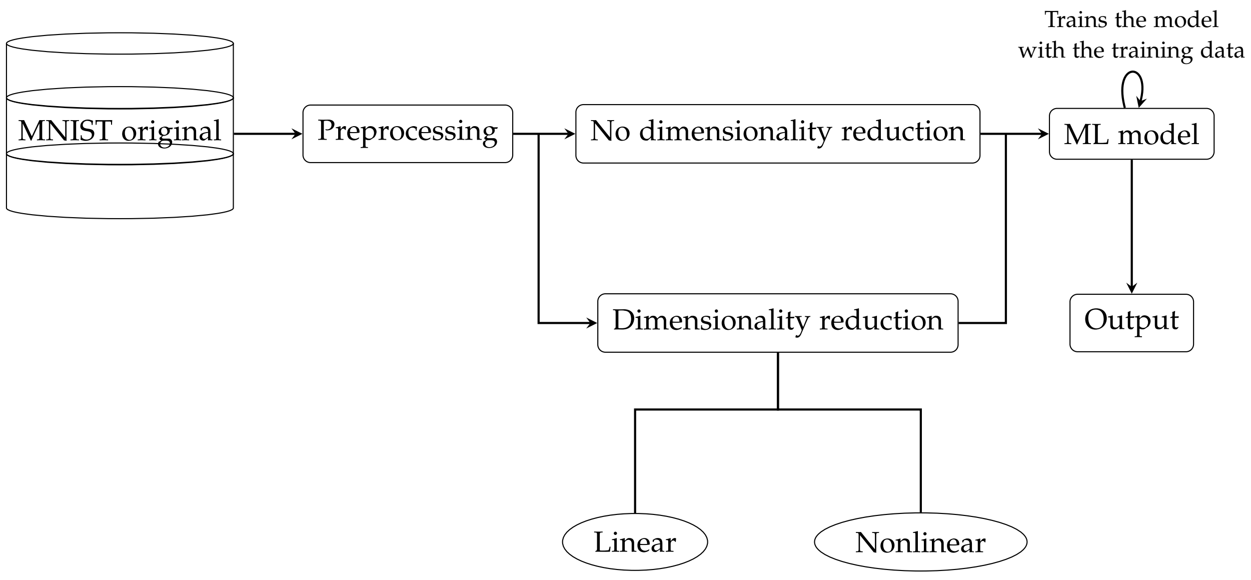


Figure 4.1: Overview of the pipeline used for the project.

that sklearn expects. If the pipeline was followed strictly, this reshaping should be done in the pre-processing segment, but it was done here for convenience.

4.2.2 Tuning loop

The last part of the pre-processing segment and the remaining segments of the pipeline are implemented together, where each dimensionality reduction method has its own function as per section 4.1. There are five functions in total, one for each dimensionality reduction method and one for no dimensionality reduction. The baseline SVM function is shown in Listing 4.2 with comments in the code highlighting the differences between the functions.

```python
def baseline_svm_results(X, y, X_test, y_test, hyperparameters,
 ↪  methodname="baseline_svm"):
    baseline_model_pipeline = Pipeline(steps=[
        ("scaler", StandardScaler()),
        # ("dimensionality_reduction",
         ↪  PCA()/LDA()/ISOMAP()/KernelPCA),
        ("classifier", SVC(kernel="linear",
         ↪  decision_function_shape="ovo", random_state=42))])
    search = GridSearchCV(baseline_model_pipeline,
                          hyperparameters,
                          cv=5,
                          scoring="f1_macro",
                          verbose=10,
                          n_jobs=-1) # -1 uses all available cores. The
                           ↪  nonlinear methods are limited to 1 core
    search.fit(X, y)
    y_pred = search.best_estimator_.predict(X_test)
    save_results(methodname, # Helper function for saving results
         search.cv_results_, # Save csv with gridsearch results
         ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred)), #
          ↪  Save confusion matrix
         classification_report(y_test, y_pred, output_dict=True)) #
          ↪  Save classification report
```

Listing 4.2: The baseline implementation of the tuning loop. The hyperparameters are passed as a dictionary. The `methodname` parameter is used for naming the output files.

The functions use the sklearn library's `Pipeline` class to create a pipeline of the pre-processing, dimensionality reduction, and classifier steps. As per section 4.1, the pre-processing step is a standard scaler. The dimensionality step depends on the function and is either PCA, LDA, ISOMAP, or kPCA. The classifier step is a support vector machine with a linear kernel and one-versus-one decision function shape and a random state of 42. The random state is used to ensure that the results are reproducible.

The classifier could be built using different sklearn functions as well, which could have improved performance. Using the `LinearSVC()` and `OneVsOneClassifier()` functions might have been more correct that the current implementation, however the way it is implemented now would allow us to use a different kernel for the SVM defined through the hyperparameter dictionary.

The `GridSearchCV` class is used to perform the hyperparameter tuning. The `GridSearchCV` class takes a pipeline as input and a dictionary of hyperparameters. Additionally the implementation sets the cross-validation number, a scoring function, a verbosity level, and the number of cores to use.

cv is the cross-validation number, in which grid search performs non-shuffled stratified k-fold cross-validation with 5 folds. The scoring function is the f1 macro score, which is the harmonic mean of the precision and recall. The verbosity level is set to 10, which means that the progress of the grid search is printed to the console in high detail. The number of cores is set to -1, which means that all available cores are used. The nonlinear methods are limited to 1 core, because of computational limits - the nonlinear methods use a lot more memory than the linear methods, which causes the program to crash if too many cores are used.

The functions are similar in structure, but they use a different number of cores and are called with a different set of hyperparameters.

Tuning hyperparameters

The following hyperparameters are used for the tuning loop:

```python
c_logspace = np.logspace(-3, 0, 4)
gamma_logspace = np.logspace(-3, 0, 4)
svm_hyperparameters = {
        "classifier__C": c_logspace, }
pca_hyperparameters = {"pca__n_components": [9, 16, 25, 36, 49],}
lda_hyperparameters = {"lda__n_components": [5, 6, 7, 8, 9],}
isomap_hyperparameters = {
    "isomap__n_components": [4, 9, 36, 49],
    "isomap__n_neighbors": [4, 5],}
kernel_pca_hyperparameters = {
    "kernel_pca__n_components": [36, 49],
    "kernel_pca__gamma": gamma_logspace,
    "kernel_pca__kernel": ["rbf", "sigmoid"]}
```

Listing 4.3: Hyperparameters used in the tuning loops.

The dictionaries are combined into single dictionaries for use in the results functions. For example, the SVM hyperparameters are combined with PCA hyperparameters to create a full hyperparameter dictionary for that method.

With regards to the hyperparameters that are shown in 4.3, the reason for choosing the values is explained in the remainder of this section.

SVM has one hyperparameter, the regularization parameter C, which penalizes the model when a sample is misclassified. The default value is 1, and higher values for C will improve the model's ability to classify the samples correctly [39]. The group chose to use a logarithmic space between 0.001 and 1 to see how much the regularization parameter affects the results.

LDA can only be used with up to nine components, as compared to the other methods. ISOMAP has the number of neighbors as hyperparameter, which is the number of points for the k-nearest neighbors used in ISOMAP. In the default implementation the number five is used [39].

The gamma hyperparameter for kPCA is a constant which is used to calculate the kernel function. The group chose values under 1 because, according to sklearn, a recommended value is 1 divided by the number of features [39]. The default kernel, which is presented, is the Radial Basis Function (RBF) kernel. Another different kernel is the sigmoid kernel.

Chapter 5

Results

This chapter presents the results of the project. The results are shown in the form of experiments. All experiments are based on the problem statement from section 2.4. The experiments are presented in the following order:

1. **Experiment 1:** How does the choice of dimensionality reduction methods impact the errors, runtime and F1 score of the pipeline, when used with the best hyperparameters?
2. **Experiment 2:** How many dimensions can a method reduce before a significant loss of accuracy occurs?
3. **Experiment 3:** What impact do kernels and hyperparameters have on the dimensionality reduction methods?
4. **Experiment 4:** How does accuracy and training time change when using different amounts of data?

Rewrite this standard thing

The second experiment will be done on a subset of the entire dataset. With 15.000 samples in the training set and the usual 10.000 samples in the test set. Instead of the standard 60.000 samples in the training set and 10.000 in the test set. The smaller sample size is used due to memory constraints regarding the nonlinear methods, as they need more memory than was available.

5.1 Experiment 1

The problem statement in section 2.4 proposes the investigation of the impact of dimensionality reduction on a chosen dataset, specifically comparing linear and nonlinear methods. This experiment will compare the selected dimensionality reduction methods in their optimal configurations on the chosen MNIST dataset. The objective is to determine the optimal configuration for each method, allowing for a fair comparison based on the same number of samples. The results of this experiment will provide insight into the impact of dimensionality reduction on machine learning models when applied in tandem.

5.1.1 Rules and overview of the experiment

The dimensionality reduction methods used in the experiment were SVM, PCA, LDA, kPCA, and ISOMAP. The baseline SVM ML model was also used without any dimensionality reduction. Each method was first cross-validated, finding the best hyperparameters for 15000 samples. Every method was tested with the same number of components. Besides LDA, it can only use up to 9 components. Then some of the methods were cross-validated again with 60000 samples to find how the methods performed with more data; the same components were still used. The methods tested on 60000 samples were the baseline SVM, PCA, and LDA, as the computer used in the experiment could not handle kPCA and ISOMAP with 60000 samples of data. The configurations used for each are shown in Table 5.1.

method	components	C	parameter	parameter
SVM-15	784	C = 0.01		
SVM-60	784	C = 0.01		
LDA-15	9	C = 0.1		
LDA-60	9	C = 1.0		
PCA-15	49	C = 0.01		
PCA-60	49	C = 0.1		
KPCA-15	49	C = 1.0	Gamma = 0.01	Sigmoid
ISOMAP-15	49	C = 0.001	neighbours = 5	

Table 5.1: Best configuration for each method used for experiment-1, method-15 and method-60, means the method with 15 and 60 thousand samples.

Every test in experiment 1 was made on the same computer, namely pc-2. See Table A.1 for the specific specs for the computer used in the experiment.

5.1.2 Results

Below is shown the results for the methods. The results are in the form of classification reports, which show the precision, recall, f1-score, support for each class, and the total accuracy for each method. The methods will be compared in accuracy, f1-score, and time fitting the data.

SVM with 15000 samples

Table 5.2 shows the accuracy for SVM without any dimensionality reduction with 15000 samples. The accuracy is 93.54%, and it takes 37 seconds to train the model. Figure 5.1 shows SVM is best at recognizing zeros and one's in pictures, as the model has the f1-score in these classes, with the scores 96.54% and 97.70%. The model has some trouble recognizing fives, eights, and threes, as these are the lowest scoring in the f1-score for all the classes, with five being 89.61%, eight being 91.48%, and three being 91.45%. With an average f1-score of 93.43%, the baseline SVM with 15000 samples is a good model.

	precision	recall	f1-score	support
0	94.7886	98.3673	96.5448	980
1	96.3979	99.0308	97.6967	1135
2	92.2488	93.4109	92.8262	1032
3	90.8203	92.0792	91.4454	1010
4	93.4739	94.8065	94.1355	982
5	90.7940	88.4529	89.6082	892
6	95.6705	94.5720	95.1181	958
7	94.4828	93.2879	93.8815	1028
8	93.1842	89.8357	91.4794	974
9	92.8717	90.3865	91.6123	1009
accuracy			93.5400	10000
macro avg	93.4733	93.4230	93.4348	10000
weighted avg	93.5263	93.5400	93.5199	10000

Table 5.2: Classification report for baseline_svm_15000

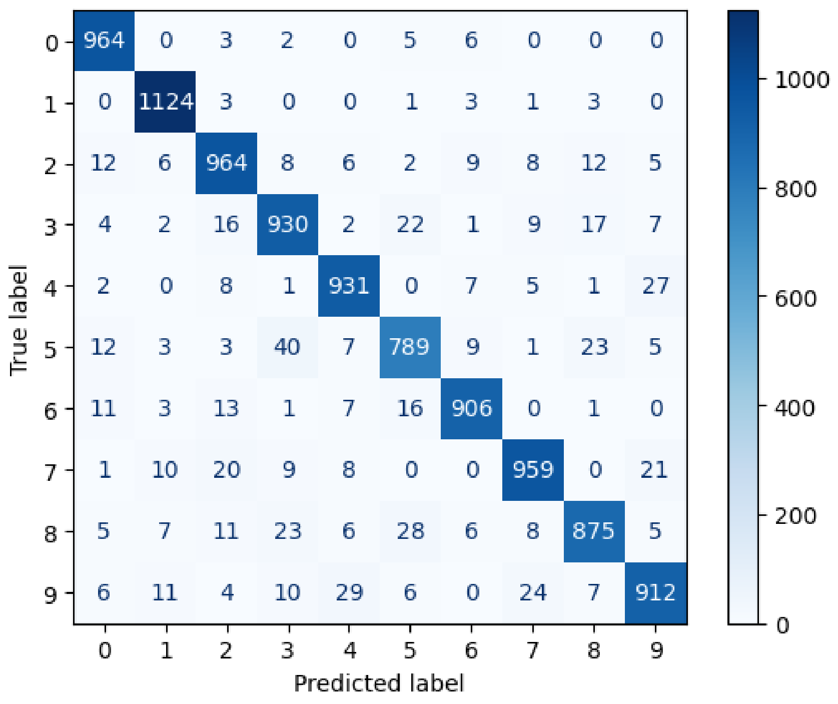


Figure 5.1: Confusion matrix for baseline SVM with 15000 samples.

SVM with 60000 samples

Table 5.3 shows the accuracy for SVM without any dimensionality reduction with 60000 samples. The accuracy is 94.56% for 60000 samples, and it takes 378 seconds to train the model, which is 6 minutes and 16 seconds. The SVM model is best

	precision	recall	f1-score	support
0	96.0278	98.6735	97.3327	980
1	97.3958	98.8546	98.1198	1135
2	93.6538	94.3798	94.0154	1032
3	91.6587	94.6535	93.1320	1010
4	93.8124	95.7230	94.7581	982
5	92.4138	90.1345	91.2599	892
6	96.2105	95.4071	95.8071	958
7	95.6262	93.5798	94.5919	1028
8	93.4668	91.0678	92.2517	974
9	94.8012	92.1705	93.4673	1009
accuracy			94.5600	10000
macro avg	94.5067	94.4644	94.4736	10000
weighted avg	94.5599	94.5600	94.5481	10000

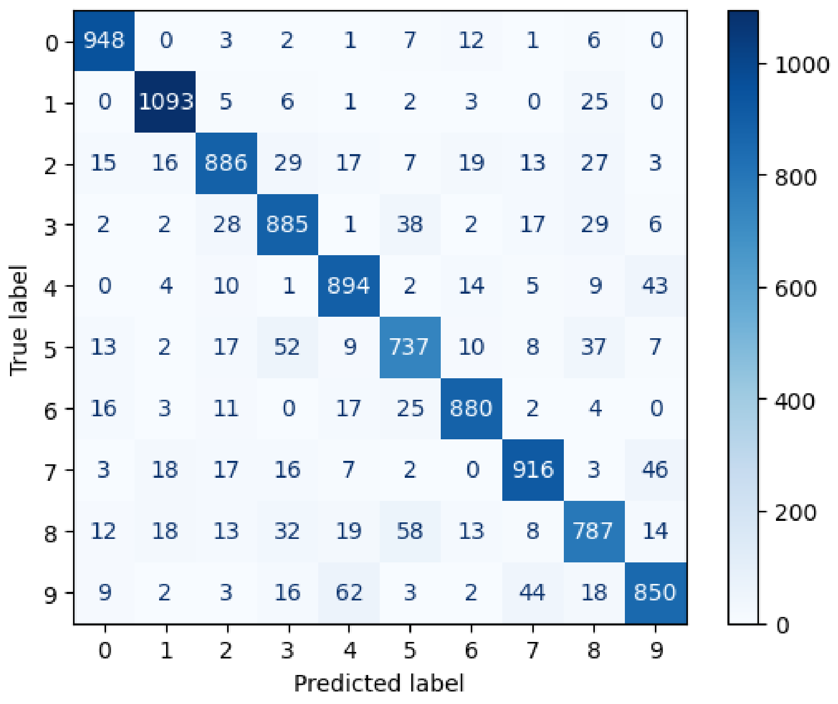
Table 5.3: Classification report for baseline_svm_60000

at recognizing zeros and ones in pictures, as the model has the f1-score in these classes, with scores of 97.33% and 98.12%. The model has some trouble recognizing fives, eights, and threes, as these are the lowest scoring in the f1-score for all the classes, with five being 91.26%, eight being 92.35%, and three being 93.13%. With an average f1-score of 94.47%, the baseline SVM with 15000 samples is a good model that uses a significant amount of time.

LDA with 15000 samples

	precision	recall	f1-score	support
0	93.1238	96.7347	94.8949	980
1	94.3869	96.2996	95.3336	1135
2	89.2246	85.8527	87.5062	1032
3	85.1781	87.6238	86.3836	1010
4	86.9650	91.0387	88.9552	982
5	83.6549	82.6233	83.1359	892
6	92.1466	91.8580	92.0021	958
7	90.3353	89.1051	89.7160	1028
8	83.2804	80.8008	82.0219	974
9	87.7193	84.2418	85.9454	1009
accuracy			88.7600	10000
macro avg	88.6015	88.6178	88.5895	10000
weighted avg	88.7285	88.7600	88.7240	10000

Table 5.4: Classification report for lda_svm_15000



	0	1	2	3	4	5	6	7	8	9
0	948	0	3	2	1	7	12	1	6	0
1	0	1093	5	6	1	2	3	0	25	0
2	15	16	886	29	17	7	19	13	27	3
3	2	2	28	885	1	38	2	17	29	6
4	0	4	10	1	894	2	14	5	9	43
5	13	2	17	52	9	737	10	8	37	7
6	16	3	11	0	17	25	880	2	4	0
7	3	18	17	16	7	2	0	916	3	46
8	12	18	13	32	19	58	13	8	787	14
9	9	2	3	16	62	3	2	44	18	850

Figure 5.2: Confusion matrix for LDA with 15000 samples.

Table 5.4 shows the accuracy for SVM with LDA as dimensionality reduction with 15000 samples. The accuracy is 88.76% for 15000 samples, and it takes 7 seconds to train the model. Figure 5.2 SVM model is best at recognizing zeros and ones in pictures, as the model has the f1-score in these classes, with scores of 94.89% and 95.33%. The model has some trouble recognizing fives, eights, and nines, as these are the lowest scoring in the f1-score for all the classes, with five being 83.14%, eight being 82.02%, and three being 85.38%. With an average f1-score of 88.59%, the baseline SVMwith 15000 samples is a worse model but is much faster than simply using SVM.

LDA with 60000 samples

Table 5.5 shows the accuracy for SVM with LDA as dimensionality reduction with 60000 samples. The accuracy is 89.33% for 60000 samples, and it takes 58 seconds to train the model. The SVM model is best at recognizing zeros and ones in pictures, as the model has the f1-score in these classes, with scores of 94.78% and 95.59%. The model has some trouble recognizing fives, eights, and threes, as these are the lowest scoring in the f1-score for all the classes, with five being 83.86%, eight being 83.31%, and three being 86.40%. SVM using PCA as the dimensionality reduction method has an average f1-score of 89.16%.

	precision	recall	f1-score	support
0	93.1953	96.4286	94.7844	980
1	94.7232	96.4758	95.5914	1135
2	90.0398	87.5969	88.8016	1032
3	85.9804	86.8317	86.4039	1010
4	88.0929	92.6680	90.3226	982
5	84.4318	83.2960	83.8600	892
6	91.0387	93.3194	92.1649	958
7	90.7093	88.3268	89.5022	1028
8	84.9520	81.7248	83.3072	974
9	88.4892	85.3320	86.8819	1009
accuracy			89.3300	10000
macro avg	89.1653	89.2000	89.1620	10000
weighted avg	89.2917	89.3300	89.2903	10000

Table 5.5: Classification report for lda_svm_60000

	precision	recall	f1-score	support
0	94.4773	97.7551	96.0883	980
1	96.7938	98.4141	97.5972	1135
2	90.4306	91.5698	90.9966	1032
3	89.7335	90.0000	89.8665	1010
4	91.6914	94.3992	93.0256	982
5	88.9143	87.2197	88.0589	892
6	95.1832	94.8852	95.0340	958
7	92.3002	92.1206	92.2103	1028
8	90.4963	87.9877	89.2244	974
9	92.7083	88.2061	90.4012	1009
accuracy			92.3700	10000
macro avg	92.2729	92.2558	92.2503	10000
weighted avg	92.3513	92.3700	92.3467	10000

Table 5.6: Classification report for pca_svm_15000

PCA with 15000 samples

Table 5.6 shows the accuracy for SVM with PCA as dimensionality reduction with 15000 samples. The accuracy is 92.37% for 15000 samples, and it takes 10 seconds to train the model. Figure 5.3 SVM model is best at recognizing zeros and ones in pictures, as the model has the f1-score in these classes, with scores of 96.08% and 97.60%. The model has some trouble recognizing fives, eights, and threes, as these are the lowest scoring in the f1-score for all the classes, with five being 88.06%, eight being 89.22%, and three being 89.87%. SVM using PCA as the dimensionality

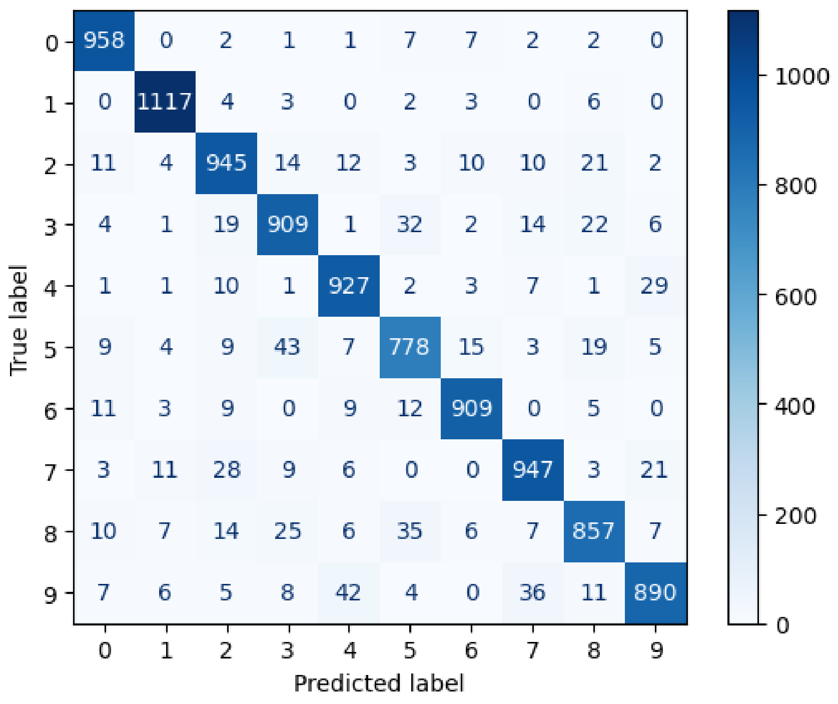


Figure 5.3: Confusion matrix for PCA with 15000 samples.

reduction method has an average f1-score of 92.25%.

PCA with 60000 samples

	precision	recall	f1-score	support
0	94.8768	98.2653	96.5414	980
1	96.7993	98.5903	97.6866	1135
2	92.3301	92.1512	92.2405	1032
3	88.9952	92.0792	90.5109	1010
4	92.3153	95.4175	93.8408	982
5	90.4157	87.7803	89.0785	892
6	95.7336	96.0334	95.8833	958
7	94.3620	92.8016	93.5753	1028
8	92.2340	89.0144	90.5956	974
9	93.7565	89.2963	91.4721	1009
accuracy			93.2500	10000
macro avg	93.1819	93.1429	93.1425	10000
weighted avg	93.2474	93.2500	93.2290	10000

Table 5.7: Classification report for pca_svm_60000

Table 5.7 shows the accuracy for SVM with PCA as dimensionality reduction with 60000 samples. The accuracy is 93.25% for 60000 samples, and it takes 97 seconds to train the model, which is 1 minute and 37 seconds. The SVM model is best at recognizing zeros and ones in pictures, as the model has the f1-score in these classes, with scores of 96.54% and 97.69%. The model has some trouble recognizing fives, eights, and threes, as these are the lowest scoring in the f1-score for all the classes, with five being 89.07%, eight being 90.60%, and three being 90.51%. SVM using PCA as the dimensionality reduction method has an average f1-score of 93.14%.

kPCA with 15000 samples

	precision	recall	f1-score	support
0	94.3137	98.1633	96.2000	980
1	96.8858	98.6784	97.7739	1135
2	91.1005	92.2481	91.6707	1032
3	89.3175	89.4059	89.3617	1010
4	91.5842	94.1955	92.8715	982
5	88.1609	85.9865	87.0602	892
6	93.7824	94.4676	94.1238	958
7	92.9342	92.1206	92.5256	1028
8	92.7039	88.7064	90.6611	974
9	91.6667	88.3053	89.9546	1009
accuracy			92.3600	10000
macro avg	92.2450	92.2278	92.2203	10000
weighted avg	92.3360	92.3600	92.3321	10000

Table 5.8: Classification report for kernel_pca_svm_15000

Table 5.8 shows the accuracy for SVM with kPCA as dimensionality reduction with 15000 samples. The accuracy is 92.36% for 15000 samples, and it takes 92 seconds to train the model, which is 1 minute and 32 seconds. Figure 5.4 SVM model is best at recognizing zeros and ones in pictures, as the model has the f1-score in these classes, with scores of 96.20% and 97.77%. The model has trouble recognizing fives, nines, and threes, as these are the lowest scores in the f1-score for all the classes, with five being 87.06%, nine being 89.95%, and three being 89.36%. SVM using kPCA as the dimensionality reduction method has an average f1-score of 92.22%.

ISOMAP with 15000 samples

Table 5.9 shows the accuracy for SVM with ISOMAP as dimensionality reduction with 15000 samples. The accuracy is 90.61% for 15000 samples, and it takes 165 seconds to train the model, which is 2 minutes and 45 seconds. Figure 5.5 SVM

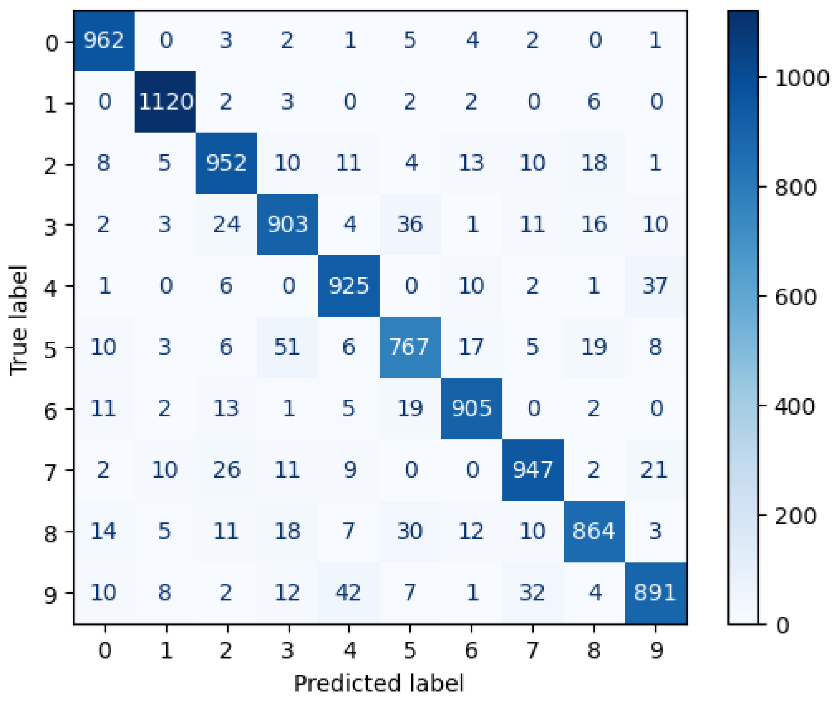
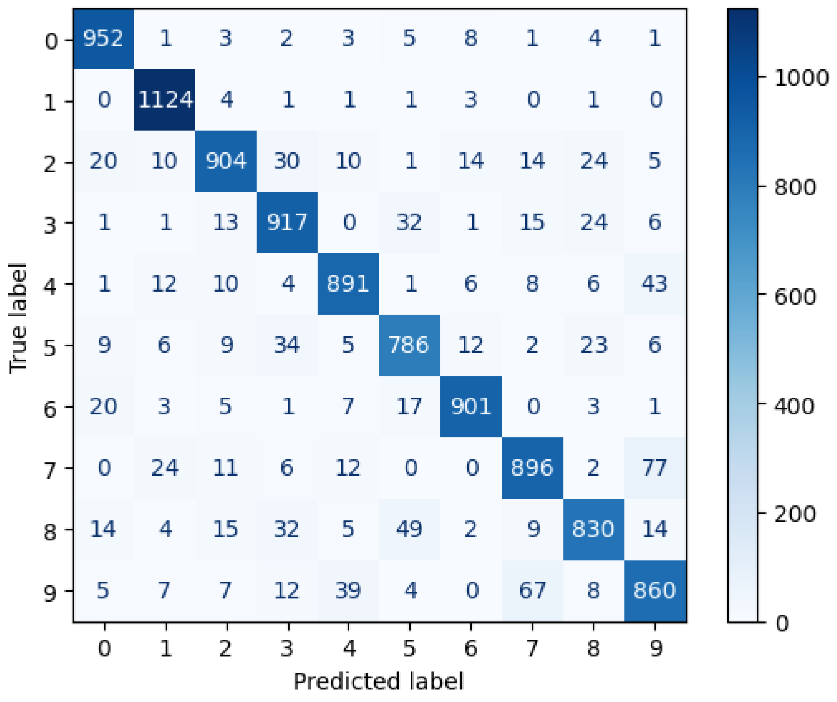


Figure 5.4: Confusion matrix for kernel PCA with 15000 samples.

	precision	recall	f1-score	support
0	93.1507	97.1429	95.1049	980
1	94.2953	99.0308	96.6051	1135
2	92.1509	87.5969	89.8162	1032
3	88.2579	90.7921	89.5071	1010
4	91.5725	90.7332	91.1509	982
5	87.7232	88.1166	87.9195	892
6	95.1426	94.0501	94.5932	958
7	88.5375	87.1595	87.8431	1028
8	89.7297	85.2156	87.4144	974
9	84.8963	85.2329	85.0643	1009
accuracy			90.6100	10000
macro avg	90.5457	90.5071	90.5019	10000
weighted avg	90.5947	90.6100	90.5771	10000

Table 5.9: Classification report for isomap_svm_15000

model is best at recognizing zeros and ones in pictures, as the model has the f1-score in these classes, with scores of 95.10% and 96.61%. The model has trouble recognizing sevens, eights, and nines, as these are the lowest scoring in the f1-score for all the classes, with seven being 87.84%, eight being 87.41%, and nine being 85.06%. SVM using ISOMAP as the dimensionality reduction method has an average



	0	1	2	3	4	5	6	7	8	9
0	952	1	3	2	3	5	8	1	4	1
1	0	1124	4	1	1	1	3	0	1	0
2	20	10	904	30	10	1	14	14	24	5
3	1	1	13	917	0	32	1	15	24	6
4	1	12	10	4	891	1	6	8	6	43
5	9	6	9	34	5	786	12	2	23	6
6	20	3	5	1	7	17	901	0	3	1
7	0	24	11	6	12	0	0	896	2	77
8	14	4	15	32	5	49	2	9	830	14
9	5	7	7	12	39	4	0	67	8	860

Figure 5.5: Confusion matrix for ISOMAP with 15000 samples.

f1-score of 90.50%.

5.1.3 Discussion of experiment 1

Comparing the results from experiment 1 is done in two comparisons; the first comparison is between the accuracy and time for the different dimensionality reduction methods. The second comparison is between the f1-scores for the different dimensionality reduction methods and what numbers the models are worst at recognizing.

Accuracy and time

All experiments done is displayed in Table 5.10, it shows the accuracy for all the different models and the time taken.

In Figure 5.6 show the comparison between the methods with 15000 samples. The baseline SVM, on 15000 samples, has an accuracy of 93.54%, and it takes 37 seconds to train the model. The baseline SVM can also be suitable to compare the other models, as it is an excellent model to hold as the baseline.

The accuracy increases by 1% when using 45000 more samples, which is a slight difference. One thing to note is that it takes 37 seconds to train the model on 15000 samples and 378 seconds to train the model on 60000 samples. This means that the time it takes to train the model grows 921,62% by using 45000 more examples of data. This is a big difference in time but a slight difference in accuracy.

Reduction method	Accuracy	Time
SVM-15	93.54%	37 seconds
LDA-15	88.76%	7 seconds
PCA-15	92.37%	10 seconds
kPCA15	92.36%	92 seconds
ISOMAP-15	90.61%	165 seconds
SVM-60	94.54%	378 seconds
LDA-60	89.33%	58 seconds
PCA-60	93.25%	97 seconds

Table 5.10: Accuracy and time for the different dimensionality reduction methods, method-15 and method-60, means the method with 15 and 60 thousand samples.

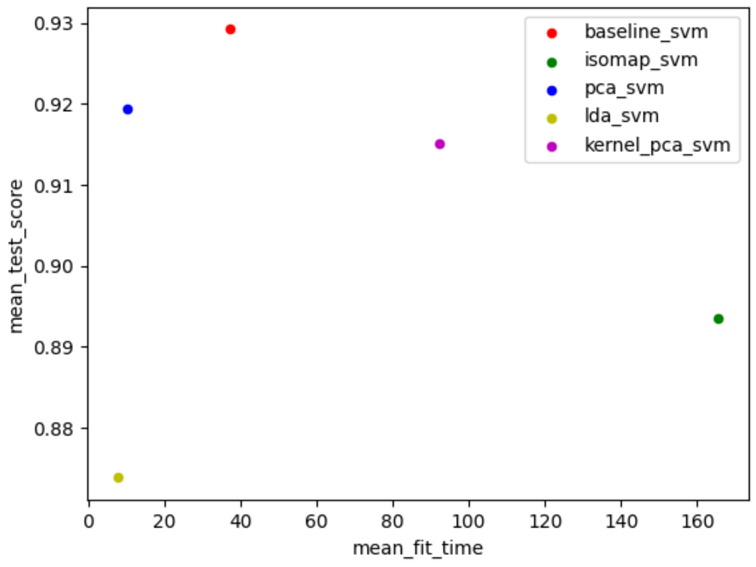


Figure 5.6: graph of time(in seconds) and accuracy(in percentage) for different dimensionality reduction methods

When using LDA on 15000 samples, the accuracy falls to 88.76%, but it only takes 7 seconds to train the model. LDA has a swift training time but low accuracy, but one thing to note, is that the maximum number of dimensions LDA can reduce to is 9 in this context. Therefore, the scores for LDA are worse than any dimensionality reduction method used. As the other methods use 49 dimensions to reduce the data, which is much higher than 9, it makes sense that the accuracy is higher than LDA. LDA only takes 7 seconds to train the model, so it is an excellent method to use if one wants a fast model with lower accuracy by comparing the other methods with more dimensions.

When using 60000 samples, the accuracy is 89.33%, and it takes 58 seconds to train the model. This means that the time it takes to train the model grows 728,57%

by using 45000 more examples of data. This is a big difference in time but a slight difference in accuracy; however, LDA is the fastest method, with lower accuracy than other methods. By using LDA, the model is faster but comes with the cost of lower accuracy.

When using PCA on 15000 samples, the accuracy is 92.37%, and it takes 10 seconds to train the model, which is faster than the baseline SVM alone but still slower than LDA with the same amount of samples. When using 60000 samples, the accuracy is 93.25%, and it takes 97 seconds to train the model.

This means that the time it takes to train the model grows by 870% by using 45000 more data samples. PCA has a lower accuracy than the baseline SVM. It is still faster than SVM alone but slower than LDA, making the model a good choice if one wants a faster model with slightly lower accuracy than the baseline SVM.

When using kPCA on 15000 samples, the accuracy is 92.36%, and it takes 92 seconds to train the model. kPCA is slower than the baseline SVM and linear methods; this makes sense since nonlinear dimensionality methods can be heavier to compute. Nevertheless, kPCA scores the highest accuracy of all the dimensionality reduction methods used when using 15000 samples. If the computational cost is not an issue, kPCA is the best dimensionality reduction.

When using ISOMAP on 15000 samples, the accuracy is 90.61%, and it takes 165 seconds to train the model. ISOMAP is slower than all the other dimensionality reduction methods and the baseline SVM. This makes sense since nonlinear dimensionality methods can be heavier to compute.

However, ISOMAP scores the second lowest accuracy of all the dimensionality reduction methods used when using 15000 samples. So there are better dimensionality reduction methods than ISOMAP if one wants a fast model with high accuracy.

To conclude from this experiment, the baseline SVM is the best model to use if one wants a fast model with high accuracy. If one wants a faster model with lower accuracy, LDA is the best model to use. If one wants a model with high accuracy but slower than the baseline SVM, kPCA is the best model. If one wants a model with high accuracy but slower than the baseline SVM, ISOMAP is the best model. PCA is the best model if one wants a model with high accuracy and faster than the baseline SVM.

F1-scores

One thing to note is that the f1-score is the harmonic mean of precision and recall, which means that the f1-score is the average of the precision and recall. The f1-score is an excellent metric to use when the classes need to be balanced, as it is the average of precision and recall.

All experiments are displayed in Table 5.11, which shows the f1-score for the different models and the three worst classes for f1-scores.

In all experiments done, all of the models have a high f1-score, which means that the models are good at recognizing the numbers in the pictures. all models are

Reduction method	f1-score	worst score	2. worst score	3. worst score
SVM-15	93.43%	5	3	8
SVM-60	94.47%	5	8	3
LDA-15	88.59%	8	5	9
LDA-60	89.16%	8	5	3
PCA-15	92.25%	5	8	3
PCA-60	93.14%	5	3	8
kPCA-15	92.22%	5	3	9
ISOMAP-15	90.50%	9	8	7

Table 5.11: F1-scores for the different dimensionality reduction methods, method-15 and method-60, means the method with 15 and 60 thousand samples.

best at recognizing zeros and ones in pictures. and good at recognizing four and sixes. When it comes to what the different models are bad at distinguishing, it is different for all.

The most common number that the models are bad at recognizing is threes, fives, and eights, in a different order depending on the dimensionality reduction method used as these are the worst in SVM with both 15000 and 60000 samples, LDA with 60000 samples, PCA with 15000 samples and 60000 samples.

ISOMAP is the only model that could be better at recognizing threes or fives, as it is terrible at sevens, eights, and nines. This can impact if one wants to use the model to find threes or fives; ISOMAP can be considered, as it is good at recognizing these numbers.

LDA with 15000 samples is interesting, as it needs to improve recognizing fives, eights, and nines. This is interesting as LDA with 60000 samples needs to improve recognizing threes, fives, and eights. This means LDA gets better at recognizing nines. This can be because more nines come into the dataset using 60000 samples, which makes LDA better at recognizing nines.

In the occasions where there is more than one experiment done to the same method, the worst f1-scores for a class change, as seen in PCA, where at 15000 samples, the eights have the second worst f1-score, but at 60000 samples, the eights have the third worst f1-score. So the worst f1-scores for a class can change depending on the number of samples used.

To conclude this experiment, the models are good at recognizing the numbers in the pictures, but they are not perfect. The models are best at recognizing zeros and ones in pictures and recognizing four and sixes. When it comes to what the different models are bad at distinguishing, it is different for all.

The most common number that the models are bad at recognizing is threes, fives, and eights, in a different order depending on the dimensionality reduction method used as these are the worst in SVM with both 15000 and 60000 samples, LDA with 60000 samples, PCA with 15000 samples and 60000 samples.

ISOMAP is the only model that is not bad at recognizing threes or fives, as it is terrible at sevens, eights, and nines. This can have an impact if one wants to use the model to find threes or fives, ISOMAP can be considered, as it is not bad at recognizing these numbers, but if one wants to use the model to find eights, then ISOMAP is not a good choice.

5.2 Experiment 2

This section will describe the second experiment of the project. The second experiment covers the necessary dimensions before reaching a threshold of 1%, 5%, and 10% loss of accuracy. The experiment will only be done on 15.000 samples, instead of the entire dataset of 60.000 samples, due to issues regarding memory usage. The experiment will focus on when different dimensionality reduction methods drop in accuracy due to too few dimensions and compare them to each other to display the robustness of each of the methods.

5.2.1 Rules and overview of the experiment

This section will cover the rules of the second experiment and how the experiment results will be evaluated.

Every test in the second experiment is run on the same computer, pc-1. See Table A.1 for the specific specs for the computer used in the experiment. It was first tried to run on another pc with less memory, but it was found that the nonlinear methods would take too long to run, and therefore it was run on pc-1.

For this experiment, the number of components will vary from around 2/5 to 50, and this range was chosen as it was believed to have a sufficient amount of components to show a general trend. For the nonlinear methods, 5 components, instead of 2, were chosen as the lowest amount, as the nonlinear methods gave errors when using fewer components than 5.

LDA is an exception, as the maximum number of components is the number of classes -1, which is 9 for the MNIST dataset, which means that the range of components for LDA will be from 2-9.

The values used to evaluate this experiment are `mean_test_score` based on `param_pca__n_components` to evaluate the model's accuracy with the number of components used.

For each experiment, the data will be analyzed to see how many components can be removed before the accuracy drops below a certain threshold. For the experiment's sake, the thresholds will be based on the best accuracy score for each method. The thresholds will be a 1% loss in accuracy, a 5% loss in accuracy, and a 10% loss in accuracy. If a method has the best accuracy score of 96%, the thresholds will be 96 - 1% = 95.05, 96 - 5% = 91.20, and 96 - 10% = 86.40. These thresholds were chosen as they are believed to be a good balance between the number of components that can be removed and the amount of accuracy lost, which could be acceptable for some use cases.

5.2.2 Results

This section will cover the results gathered from running the second experiment. Each of the dimensionality reduction methods will be presented using scatter plots. The scatter plots will show the number of components used along the x-axis and the model's accuracy along the y-axis. Each component has multiple dots, so the accuracy varies slightly depending on the hyperparameters used, but the general trend is the same. The results will then be compared and evaluated based on the experiment's rules.

The main focus of the evaluation will be on when the accuracy starts to drop noticeably and how many components are needed to have good accuracy still. The scatter plots are used to represent the results visually, and the specific values of the accuracy will also be discussed. These values are taken from the csv files generated from running the second experiment.

PCA

PCA is a linear dimensionality reduction method; the scatter plot representing this method can be seen in Figure 5.7.

As one would expect, the model's accuracy increases as the number of components increases. However, around 20 components, the accuracy starts to drop, the accuracy especially has a noticeable drop between 10 and 20 components, and the accuracy has a drastic drop for each component removed below ten components. This is expected as the lower the number of components the model has to work with; the more information is lost.

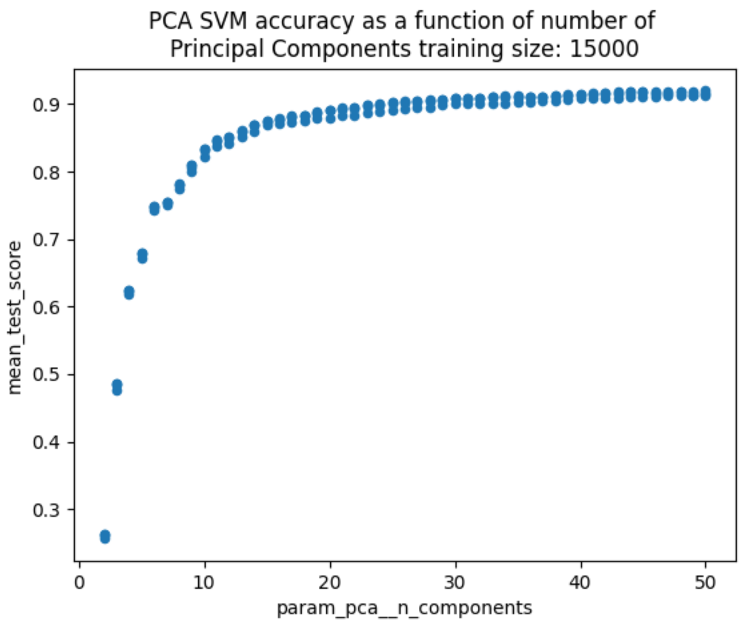


Figure 5.7: Accuracy of the SVM model with PCA as dimensionality reduction method, with the number of components used.

For PCA, the highest accuracy value is 91.99% with 50 components and the value 0.01 for the C hyperparameter in SVM. The thresholds for PCA are: 91.99 - 1% = 91.07, 91.99 - 5% = 87.39, and 91.99 - 10% = 82.79. The results of the experiment for PCA will be compared to these thresholds.

By sorting the data by `mean_test_score` and going through the values, given the best-case scenario with the best hyperparameters found. The first value that drops below the threshold of 91.07% is with an accuracy of 90.57% at 29 components, which means that the model's accuracy only increases by 1% with the last 21 components, which is close to half the total amount of components. The next threshold of 87.39% accuracy is found at 86.86% with 14 components. By removing an additional seven components, the accuracy dropped from a 1% loss to a 5% loss. The final threshold of 82.79% accuracy is found at 80.70% with nine components. By removing only five components, the accuracy dropped from a 5% loss to a 10% loss.

Linear discriminant analysis

The scatter plot representing LDA method can be seen in Figure 5.8. LDA reduces the dimension to the number of classes −1, which is why the total number of components used is nine since MNIST has ten different numbers. The general trend is still valid for discussion in the second experiment.

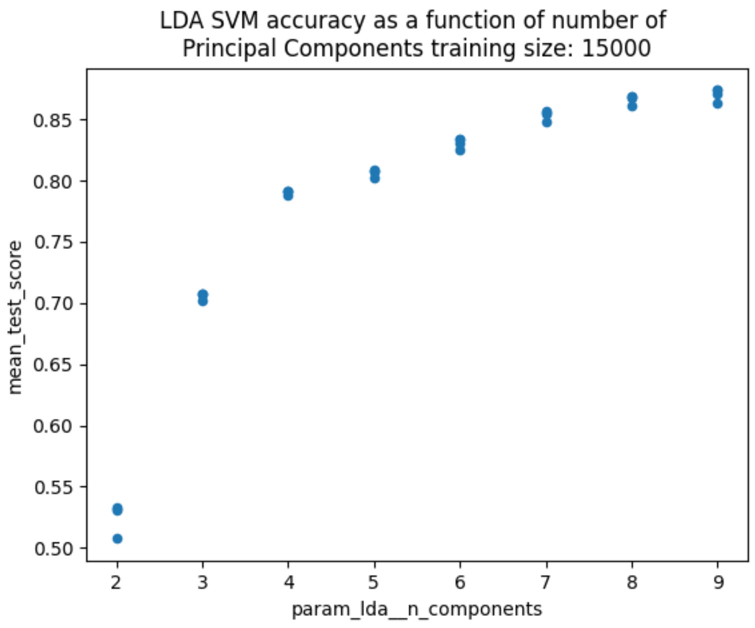


Figure 5.8: Accuracy of the SVM model with LDA as dimensionality reduction method, with the number of components used.

For LDA, the highest accuracy score is at 87.38% with nine components and the value 0.1 for the C hyperparameter in SVM. The thresholds for LDA are: 87.38 - 1% = 86.50, 87.38 - 5% = 83.01, and 87.38 - 10% = 78.64. The experiment's results for LDA will be compared to these thresholds.

Repeating the method of looking through the data gathered, the first value that drops below the first threshold of 86.50%, with the 0.1 hyperparameter, is at 85.61% with seven components.

The next threshold of 83.01% accuracy is found at 80.83% with five components. By removing two components, the accuracy dropped from a 1% loss to a 5% loss, which is not many components when compared to PCA, but for LDA that only has nine total components, a large percentage of the components can be removed with only a 5% accuracy loss.

The final threshold of 78.64% accuracy is found at 70.75% with only three components. They are showing a significant drop from the threshold since the first few components impact the accuracy score significantly, and the closest value to the threshold is 8% away.

kPCA

kPCA is a nonlinear dimensionality reduction method, and the scatter plot representing this method can be seen in Figure 5.9.

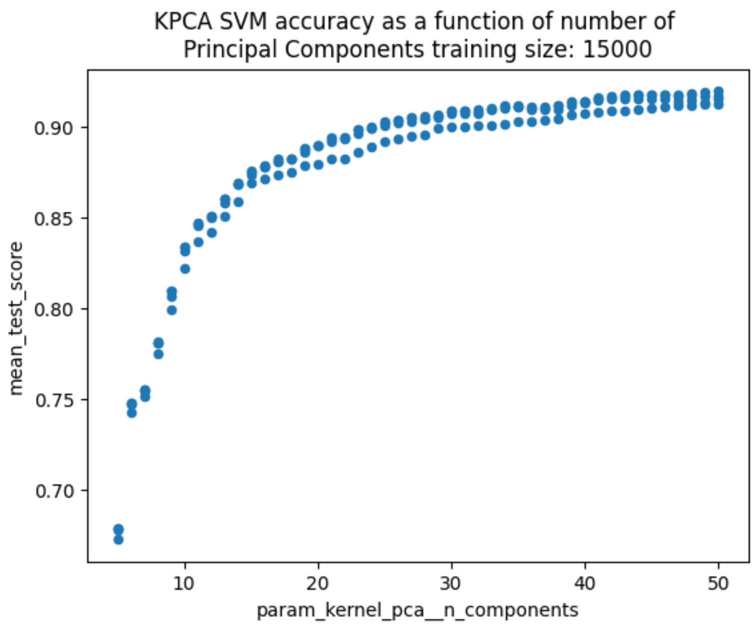


Figure 5.9: Accuracy of the SVM model with kPCA as dimensionality reduction method, with the number of components used.

Figure 5.9 is very similar to the other methods, and almost identical with Figure 5.7, the only difference is that kPCA uses at minimum 5 components, where PCA uses at the lowest 2 components.

kPCA has a top accuracy score of 91.99% with the value 0.01 for the C hyperparameter in SVM, meaning that the thresholds for kPCA are: 91.99 - 1% = 91.07, 91.9 - 5% = 87.39, and 91.9 - 10% = 82.79. The experiment's results for kPCA will be compared to these thresholds.

Similar to linear methods, the data is sorted by its accuracy score to find the first value where the accuracy drops below a threshold, that uses the same hyperparameter. The first threshold of 91.07% accuracy is found at 91.04% with 34 components. The next threshold of 87.39% accuracy is found at 86.86% with 14 components. The final threshold of 82.79% accuracy is found at 80.70% with nine components.

ISOMAP

ISOMAP is the final method of the second experiment, which is nonlinear, and the scatter plot representing this method can be seen in Figure 5.10.

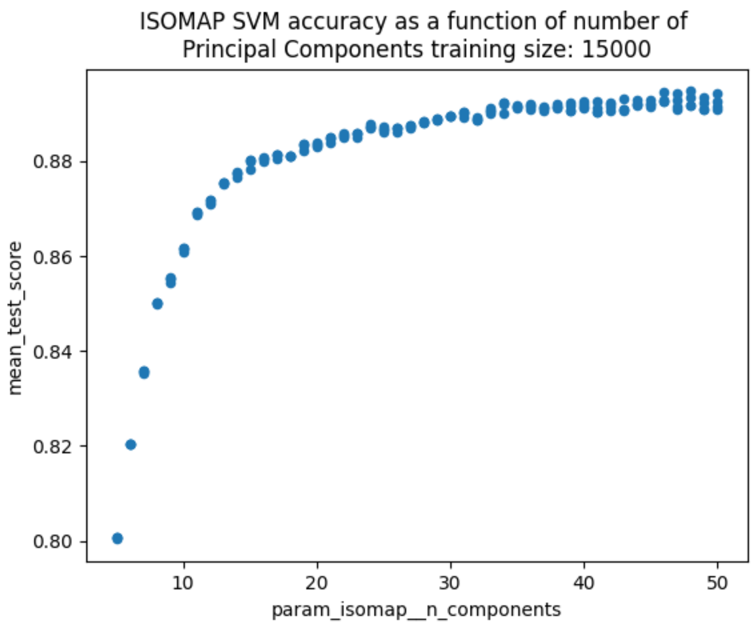


Figure 5.10: Accuracy of the SVM model with ISOMAP as dimensionality reduction method, with the number of components used.

Figure 5.10 is similar to the other methods as the accuracy drops drastically as the number of components decreases, but interestingly even the lowest accuracy for ISOMAP is still relatively high. The span from best to worst accuracy is much smaller than the other methods. The highest accuracy score for ISOMAP is 89.4% with 48 components and the value 0.001 for the C hyperparameter in SVM, and the lowest accuracy score is 80% with five components. The best and worst accuracy scores are closer together than the other methods, with only a 9% difference.

isomap has a top accuracy score of 89.47%, with 48 components, and the value 0.001 for the C hyperparameter in SVM. meaning that the thresholds for ISOMAP are: 89.47 - 1% = 88.57, 89.47 - 5% = 84.99, and 89.47 - 10% = 80.52.

The first threshold of 88.57% accuracy is found at 88.49% with 22 components, the next threshold of 84.99% accuracy is found at 83.54% with seven components, and the final threshold of 79.4% accuracy is not found in the data, as the lowest accuracy score is 80.8% with five components. To still be able to compare the results of ISOMAP with the other methods, the lowest accuracy score will be used as the threshold, but it should be made clear that the actual threshold is not found in the data.

5.2.3 Discussion of experiment 2

This section will discuss the results of the second experiment and compare the results of the different methods, first by discussing the results of the linear methods, then the nonlinear methods, and finally, comparing the results of all methods.

For each section, a table will display the different percentages of components remaining before the accuracy drops below a threshold. The table will also show the number of remaining components to show the difference between the methods.

Linear methods

By comparing the two linear methods used for the second experiment, it is essential to know that the number of components is very different between PCA LDA, which means that an exact comparison between the two methods is not always clear. Instead of using the number of components as a comparison, the percentage of remaining components will be used to try and make a fair comparison. A table showing the differences between the two methods can be seen in Table 5.12.

Thresholds	PCA % 50	LDA % 9
1%	58% (29)	77.7% (7)
5%	28% (14)	55.5% (5)
10%	18% (9)	33.3% (3)

Table 5.12: Percentage of the components remaining after the threshold is reached, with the corresponding number of components in paratheses.

Table 5.12 shows, for each linear method, the amount of components left after each of the thresholds is reached. So for PCA, 58% of the total number of components remain before reaching the first threshold of 1%, whereas, for LDA, there is 77.7%.

From Table 5.12, it can be seen that before reaching the first threshold of losing 1% accuracy, PCA can cut off 42% of its components, while LDA can can only cut off 22.3%.

By comparing the two linear methods, it can be concluded that LDA is more prone to losing accuracy when removing components than PCA. The accuracy loss is likely because LDA has so few components to work with in the first place. It could be argued that the results are not entirely fair since PCA has so many more components, but scaling the values with percentage should show a general trend.

nonlinear methods

Unlike the linear methods, the nonlinear methods have a similar number of components, which should give a more fair comparison. A table showing when the respective thresholds were reached for each of the nonlinear methods can be seen in Table 5.13.

Thresholds	KPCA % 50	ISOMAP % 50
1%	68% (34)	44% (22)
5%	28% (14)	14% (7)
10%	18% (9)	1% (5)*

Table 5.13: Percentage of the components remaining after the threshold is reached, with the corresponding number of components in paratheses.

Figure 1 shows that kPCA and ISOMAP can afford to cut off a significant amount of components before reaching the first threshold of 1%. kPCA can cut off 32% of its components, while ISOMAP can cut off 56% of its components. Another interesting note is that ISOMAP never reaches the 10% accuracy loss threshold, and the closest accuracy score is used instead. Although it is not entirely accurate, the percentage from the actual threshold is <1%, so for the sake of the comparison, it will be assumed that the threshold was reached.

Generally, it can be concluded that both kPCA and ISOMAP can afford to cut off a significant amount of components before losing accuracy. It is interesting to note that ISOMAP can cut off more components than kPCA before losing accuracy. However, kPCA reaches a higher accuracy score than ISOMAP but also reaches a lower accuracy score than ISOMAP, which means that ISOMAP could be considered more consistent than kpca, as the span of accuracy score for ISOMAP is smaller.

Comparison of methods

Now that the linear and nonlinear methods have been compared, it is time to compare the methods to each other. The results can be seen in Table 5.14.

Thresholds	PCA % 50	LDA % 9	KPCA % 50	ISOMAP % 50
1%	58% (29)	77.7% (7)	68% (34)	44% (22)
5%	28% (14)	55.5% (5)	28% (14)	14% (7)
10%	18% (9)	33.3% (3)	18% (9)	1% (5)*

Table 5.14: Percentage of the components remaining after the threshold is reached, with the corresponding number of components in paratheses.

The first noticeable thing is that PCA and kPCA are similar in the number of components they can cut off before losing accuracy enough to reach the thresholds. The highest accuracy score is the same between the two methods, but the lowest accuracy score is lower for PCA, this lower accuracy score is likely due to the range of components PCA has. The lower the percentage in Table 5.14, the more components can be cut off without losing accuracy. So from this, it can be concluded that LDA is the method that has the most drastic accuracy loss when cutting off components. This is likely because LDA has so few components to work in the first place, and therefore it is more sensitive to the removal of components. ISOMAP is the method that has the most negligible accuracy loss when cutting off components. Additionally, ISOMAP's worst accuracy score is better than any other method, but its best accuracy score is also the lowest.

From the second experiment, it can be concluded that ISOMAP is the method that can cut off most components before losing accuracy. However, each method has its strengths and weaknesses, and it is essential to consider the context of the problem when choosing a method.

5.3 Experiment 3

This experiment explores the differences in the kernels implemented in cross-validation regarding kPCA. Experiment 1, see section 5.1, used the sigmoid kernel, and this experiment's goal is to assess whether the choice of the kernel could significantly impact the model's performance.

The configuration for the sigmoid kernel also used a different gamma value compared to the configuration for the RBF kernel, which opens up researching the effect of the kernel and gamma for the model's accuracy. The experiment will focus on the confusion matrices and scores obtained from the different kernels.

5.3.1 Rules and overview of the experiment

The dimensionality reduction methods used in this experiment will be PCA and kPCA. The classification method will be SVM. The kernels used for kPCA will be RBF and sigmoid.

The best configurations for each method used in this experiment is shown in Table 5.15. The input of the data samples will be 15000 for both of the methods. The number of components used for all methods will be 49 components. The evaluation will be based on the confusion matrices and the results from the CSV file for cross-validation for kPCA.

This experiment is run on pc-2. See Table A.1 for the specific specs for the computer used in the experiment.

5.3.2 Results

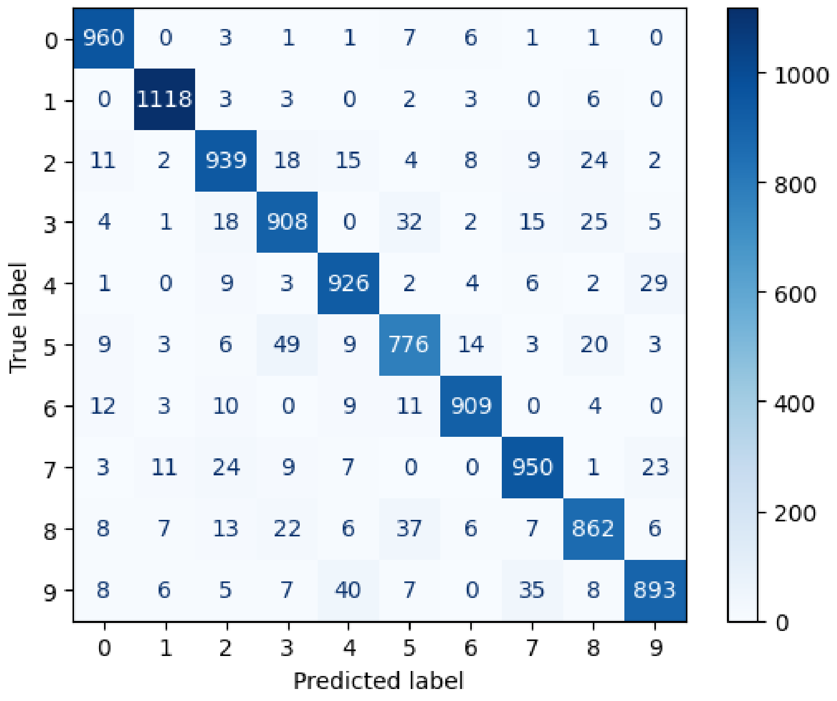
Below is shown the results for the methods. The results are in the form of confusion matrices. Accuracy percentages from the confusion matrices will compare the methods. The results for the two tests run on different γ values are not presented

method	components	C	parameter	parameter
PCA-15	49	C = 0.01		
KPCA-15	49	C = 1.0	Gamma = 0.01	Sigmoid
KPCA-15	49	C = 1.0	Gamma = 0.001	RBF
KPCA-15	49	C = 1.0	Gamma = 0.001	Sigmoid
KPCA-15	49	C = 1.0	Gamma = 0.01	RBF

Table 5.15: Best configuration for each method used for experiment-3, method-15 means the method with 15 thousand samples.

as a confusion matrix, as the accuracy will only be compared, therefore the results are only shown in the CSV file and the discussion, see subsubsection 5.3.3.

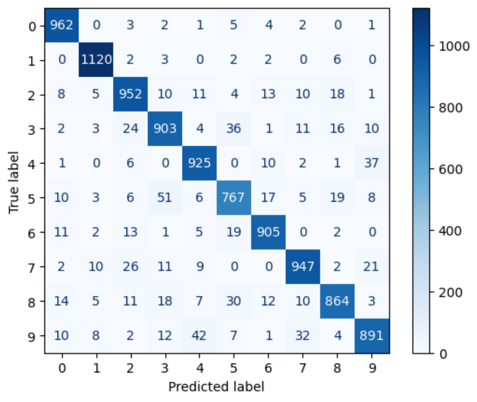
PCA



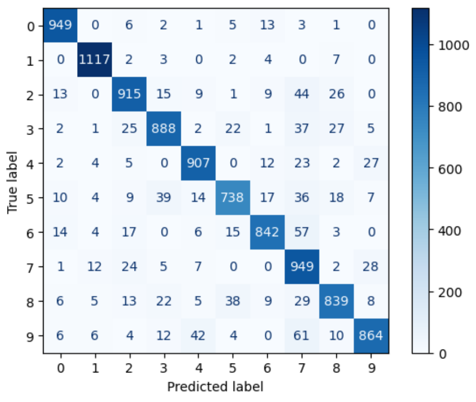
	0	1	2	3	4	5	6	7	8	9
0	960	0	3	1	1	7	6	1	1	0
1	0	1118	3	3	0	2	3	0	6	0
2	11	2	939	18	15	4	8	9	24	2
3	4	1	18	908	0	32	2	15	25	5
4	1	0	9	3	926	2	4	6	2	29
5	9	3	6	49	9	776	14	3	20	3
6	12	3	10	0	9	11	909	0	4	0
7	3	11	24	9	7	0	0	950	1	23
8	8	7	13	22	6	37	6	7	862	6
9	8	6	5	7	40	7	0	35	8	893

Figure 5.11: Confusion matrix for PCA

Figure 5.11 shows the results for PCA. It has an accuracy of 92.37%. It shows PCA is best at recognizing zeros and ones in pictures, as the model had guessed lees on the other numbers when the picture was zero or one. The model has trouble recognizing nines from fours and sevens, threes from fives, and fives from threes and eights.



(a) Confusion matrix for kPCA Sigmoid



(b) Confusion matrix for kPCA RBF

Figure 5.12: both kPCA kernels confusion matrices

KPCA

Figure 5.12a shows the results for kPCA with the sigmoid kernel. It has an accuracy of 91.50%. It shows Kernel PCA with Sigmoid kernel (KPCA-S) is best at recognizing zeros and ones in pictures, as the model had guessed lees on the other numbers when the picture was zero or one. The model has trouble recognizing nines from fours and sevens and tells threes, fives, and eights from each other.

Figure 5.12b shows the results for kPCA with the RBF kernel. It has an accuracy of 89.5%. It shows Kernel PCA with RBF kernel (KPCA-R) is best at recognizing zeros and ones in pictures, as the model had guessed lees on the other numbers when the picture was zero or one. The model has trouble recognizing nines from fours and sevens and tells fives, threes, and sevens from each other.

5.3.3 Discussion of experiment 3

The results are compared two times, the best configurations on two kernels, where the main comparisons are in error percentage for each class, and last looking at the γ hyperparameters influence on the results.

Comparison of the best configurations

From the results presented in Subsection5.3.3, all the methods mostly confuse nines with fours and sevens. They also confuse threes with fives and sevens or eights. It can be further noted that KPCA-R is the worst at confusing various numbers with the number seven.

From the overview provided regarding the difference in numbers, a percentage would be preferable, more specifically, a percentage of the errors made in the numbers 0-9. The error percentage can be calculated as the number of correct predictions divided by the number of numbers from the given class. Table 5.16 shows the difference in percentages of errors made by the methods for each number.

	pca	kpca-s	kpca-r
0	2.959	1.836	3.163
1	1.585	1.321	1.585
2	8.817	7.751	11.337
3	10.594	10.594	12.079
4	5.702	5.804	7.637
5	12.556	14.013	17.264
6	6.054	5.532	12.108
7	7.101	7.879	7.684
8	13.552	11.293	13.860
9	11.992	11.694	14.370

Table 5.16: Error percentage for each number for the methods

In Table 5.16 it can be seen that PCA has the best performance for the numbers four and seven, while KPCA-S has the best performance for the numbers two and eight. PCA and KPCA-S have similar performance for most numbers, while KPCA-R has the worst performance for most numbers. In particular, KPCA-R has the worst performance for numbers four, five, and six. Regarding number six, KPCA-R has a twice as bad score as KPCA-S.

However, the overall error percentage is one of many things that could be considered. Another exciting thing that can be considered is observing the difference in the error percentages between the two kernels of Kernel with PCA, as presented in Table 5.17.

	kpca-s	kpca-r
0	-1.123	+0.204
1	-0.264	0
2	-1.066	+2.52
3	0	+1.485
4	+0.102	+1.935
5	+1.457	+4.708
6	-0.522	+6.054
7	+0.778	+0.583
8	-2.259	+0.308
9	-0.298	+2.378

Table 5.17: Difference between the error percentage for the kernels compared to pca. The difference is calculated by subtracting the error percentage of the kernel from the error percentage of pca.

Based on the Table 5.17, one can see that KPCA-R has a higher percentage of errors than KPCA-S. However, both methods have around the same error percentage with the number seven.

The worst performance of KPCA-R is more visible at number six, where the difference between KPCA-R and KPCA-S is around 6.57%. The second most erroneous class for KPCA-R is five, at 4.70%, and it is the same class that KPCA-S has a hard time with and manages to confuse more numbers than PCA. Still, the difference between methods at number five is about 3.25%. Around the same percentage difference between methods is also visible for numbers two, eight, and nine.

The slightest difference in both methods, where the KPCA-R and KPCA-S improve the model compared to PCA, is number one, and the difference, where they do not improve the model compared to PCA, is number seven.

From Table 5.17, one can conclude that, with the current configurations, KPCA-R's kernel is detrimental to the model's performance.

The influence of γ on the results

This section presents the influence of the γ hyperparameter on the results. The results are presented in Table 5.18. The two kernels of kPCA are presented in the table, along with the γ value and the model's accuracy.

Accuracy	γ=0.001	γ=0.01
kpca-r	89.49%	55.97%
kpca-s	91.22%	91.50%

Table 5.18: Accuracy of kPCA with different γ values

Table 5.18 shows that the accuracy of KPCA-S is not affected as much by the γ value as KPCA-R. The finding shows, among others, that KPCA-R is more sensible to the changes in the γ hyperparameter, as the accuracy can be reduced from 89.49% to 55.97%, around 33.50% difference. This is a big difference compared to the 0.28% difference for KPCA-S, as it only drops from 91.50% to 91.22%. Such a difference is an important finding, as it shows that the model is more sensitive to the changes in the γ hyperparameter.

In this experiment, a comparison of the performance of PCA and kPCA with different kernels on the MNIST dataset is presented. The results show that PCA performs better than both kPCA kernels used on the MNIST dataset. The results also show that kPCA depends on the γ hyperparameter value, as the accuracy change can be remarkable if the optimal value for the specific kernel is changed.

5.4 Experiment 4

In this experiment, a machine learning pipeline is used to explore how the performance of linear and nonlinear dimensionality reduction methods compare using different sizes of samples.

Overall, the results of this experiment provide insight into the factors that can influence the effectiveness of dimensionality reduction in machine learning and can inform the choice of dimensionality reduction method in real-world scenarios.

Furthermore, the findings can contribute to the broader understanding of the role of sample size in choices of dimensionality reduction and its effect on machine learning models, which is relevant especially when working with cross validation.

5.4.1 Rules and overview of the experiment

This experiment uses the original hyperparameters. These can also be read in subsubsection 4.2.2. The difference is that it will be used on datasamples from MNIST of the size 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000, 5000.

This experiment is run on pc-4. See Table A.1 for the specific specs for the computer used in the experiment.

5.4.2 Results

This section presents the results of the fourth experiment, which compares the performance of different dimensionality reduction methods on a classification task. The results are shown for different sample sizes using confusion matrices, tables of the classification reports, and scatter plots, which visualize the relationship between sample size, accuracy, and time taken.

The results are evaluated based on the rules of the experiment, focusing on accuracy, F1 score, and time taken. The specific accuracy values obtained from the experiment's CSV files are also discussed. The scatter plots provide a visual representation of the results and enable the comparison of when the different methods accuracy and time start to rise. In general, the results for each method and the base case will show the results for samples of sizes of 200, 1000 and 5000 in detail.

Support vector machine model

The results of this experiment are shown in Table 5.19. The table shows the precision, recall, and f1-score for each of the ten classes in the dataset and the overall accuracy of the model. The table also shows the macro average and weighted average scores.

Overall, the results show that the SVM model achieved an accuracy of approximately 67%. This indicates that the model performed relatively well but still had some errors in its predictions. The precision and recall scores for each class varied, with some classes having higher scores than others. For example, the model had a precision of approximately 87% for class 0 but only a precision of approximately 45% for class 9.

These results provide a baseline for comparison with the results of the other parts of the experiment, in which different dimensionality reduction methods were applied. The results of this initial experiment will be used to evaluate the effectiveness of the different dimensionality reduction methods in improving the performance of the SVM model.

The second part of the base case involved using a sample size of 1000; it can be seen in 5.20. Compared to the results of the first part of the experiment, the accuracy

	precision	recall	f1-score	support
0	83.0224	90.8163	86.7446	980
1	78.2424	97.2687	86.7243	1135
2	67.8335	69.4767	68.6453	1032
3	73.6059	78.4158	75.9348	1010
4	70.7377	87.8819	78.3833	982
5	71.6243	41.0314	52.1739	892
6	91.3043	63.5699	74.9538	958
7	78.3178	81.5175	79.8856	1028
8	73.1092	71.4579	72.2741	974
9	71.7842	68.5828	70.1470	1009
accuracy			75.6700	10000
macro avg	75.9582	75.0019	74.5867	10000
weighted avg	75.9485	75.6700	74.9591	10000

Table 5.19: Classification report for baseline_svm_200

	precision	recall	f1-score	support
0	92.5998	97.0408	94.7683	980
1	92.1667	97.4449	94.7323	1135
2	86.5842	88.1783	87.3740	1032
3	87.7095	77.7228	82.4147	1010
4	86.7005	86.9654	86.8327	982
5	78.5340	84.0807	81.2128	892
6	91.4621	89.4572	90.4485	958
7	86.0927	88.5214	87.2902	1028
8	86.4350	79.1581	82.6367	974
9	82.2178	81.5659	81.8905	1009
accuracy			87.1700	10000
macro avg	87.0502	87.0136	86.9601	10000
weighted avg	87.1760	87.1700	87.1014	10000

Table 5.20: Classification report for baseline_svm_1000

of the SVM model improved significantly, achieving an accuracy of approximately 87%. This indicates that using a larger sample size improved the performance of the model. The precision and recall scores for each class also improved, with most classes having higher scores than in the first part of the experiment.

The following sample of size 5000 can be seen in Table 5.21. Comparing Table 5.21 and Table 5.21, it is clear that the SVM model achieved higher accuracy and precision when trained on a larger sample size. For example, the accuracy of the model increased from 87.17% for the 1000-sample dataset to 92.05% for the 5000-sample dataset. Similarly, the precision of the model increased for most of the classes, with

	precision	recall	f1-score	support
0	95.5224	97.9592	96.7254	980
1	95.3072	98.4141	96.8357	1135
2	90.9980	90.1163	90.5550	1032
3	89.0279	91.5842	90.2879	1010
4	90.4854	94.9084	92.6441	982
5	90.1278	86.9955	88.5339	892
6	95.2128	93.4238	94.3098	958
7	92.5123	91.3424	91.9236	1028
8	90.4555	85.6263	87.9747	974
9	90.2414	88.8999	89.5657	1009
accuracy			92.0500	10000
macro avg	91.9891	91.9270	91.9356	10000
weighted avg	92.0338	92.0500	92.0197	10000

Table 5.21: Classification report for baseline_svm_5000

the most significant increase being seen for class 5, where the precision increased from 71.62% to 90.12%.

Additionally, the f1-score, which is a measure of the balance between precision and recall, also improved for most classes when using a larger sample size. This suggests that the model was able to make more accurate predictions and avoid false positives and false negatives more effectively when trained on a larger dataset.

Principal component analysis

The second part of the experiment involved running a machine learning pipeline applying PCA as a dimensionality reduction method. The sample sizes for this part of the experiment were 200, 1000, and 5000.

This classification report shows the performance of a SVM model that has been trained using PCA. The result of the first sample of 200 can be seen in 5.22 For example, class 0 has a precision of 83.82% and a recall of 87.75%, while class 9 has a precision of 64.17% and a recall of 72.44%. This indicates that the model is more accurate at correctly identifying instances of class 0 than it is at correctly identifying instances of class 9. The f1-score for class 0 is 85.74%, while the f1-score for class 9 is 68.06%, further highlighting the difference in performance between the two classes. Overall, it appears that class 0 and class 9 have the largest differences in performance on this classification report.

In 5.23 it can see that overall, the second model (pca_svm_1000) performs better on the classification task than the first model (pca_svm_200), as shown by the higher accuracy, precision, recall, and f1-score values for most classes in the second report. For example, class 0 has a precision of 91.72% and a recall of 96.12% in the second report, compared to 83.82% and 87.75% in the first report. The f1-score for class 0 is also higher in the second report (93.87%) than in the first report (85.87%).

	precision	recall	f1-score	support
0	83.8207	87.7551	85.7428	980
1	77.0701	95.9471	85.4788	1135
2	67.5052	62.4031	64.8540	1032
3	74.4227	82.9703	78.4644	1010
4	69.2244	84.5214	76.1119	982
5	74.7492	50.1121	60.0000	892
6	93.1649	65.4489	76.8853	958
7	83.2487	79.7665	81.4704	1028
8	73.7317	67.1458	70.2848	974
9	64.1791	72.4480	68.0633	1009
accuracy			75.4000	10000
macro avg	76.1117	74.8518	74.7356	10000
weighted avg	76.0509	75.4000	75.0028	10000

Table 5.22: Classification report for pca_svm_200

	precision	recall	f1-score	support
0	91.7235	96.1224	93.8714	980
1	92.6271	96.2996	94.4276	1135
2	88.0611	89.3411	88.6965	1032
3	87.6923	79.0099	83.1250	1010
4	90.0826	88.7984	89.4359	982
5	78.9038	85.5381	82.0871	892
6	91.5565	93.9457	92.7357	958
7	90.2119	86.9650	88.5587	1028
8	84.5890	76.0780	80.1081	974
9	81.5414	84.9356	83.2039	1009
accuracy			87.8200	10000
macro avg	87.6989	87.7034	87.6250	10000
weighted avg	87.8426	87.8200	87.7565	10000

Table 5.23: Classification report for pca_svm_1000

One interesting difference between the two reports is the performance on class 3. In the first report, class 3 has a recall of 82.97%, with an f1-score of 78.46%. In the second report, class 3 had a lower recall of 79%, resulting in a f1-score of 83.12%. This indicates that the second model (pca_svm_1000) is less accurate at correctly identifying instances of class 3 than the first model the f1 score is still higher due to the precision being suitably higher to compensate(pca_svm_200).

The last model is seen in Table 5.24 and is overall, the pca_svm_5000 model appears to have better performance across most of the metrics, with higher values for precision, recall, and f1-score for most of the classes.

	precision	recall	f1-score	support
0	94.0476	96.7347	95.3722	980
1	95.8656	98.0617	96.9512	1135
2	89.0927	89.4380	89.2650	1032
3	86.6218	89.1089	87.8477	1010
4	90.7389	93.7882	92.2384	982
5	87.6417	86.6592	87.1477	892
6	93.6259	93.5282	93.5770	958
7	92.5000	89.9805	91.2229	1028
8	89.7577	83.6756	86.6100	974
9	89.1348	87.8097	88.4673	1009
accuracy			91.0000	10000
macro avg	90.9027	90.8785	90.8699	10000
weighted avg	90.9832	91.0000	90.9711	10000

Table 5.24: Classification report for pca_svm_5000

Overall, the results show that the SVM model using PCA achieved an accuracy of approximately 75%, 77%, and 84% for the 200, 1000, and 5000 sample sizes, respectively. This indicates that the model performed relatively well, but still had some errors in its predictions. The precision and recall scores for each class varied, with some classes having higher scores than others. For example, the model had a precision of approximately 83% for class 0 with a 200 sample size, but only a precision of approximately 64% for class 9 with a 1000 sample size. Classes 0 and 9 showed the most significant differences in performance across the different sample sizes.

Linear discriminant analysis

This classification report seen in Table 5.25 shows the performance of a SVM model that has been trained using LDA on classifying handwritten digits from the MNIST data set. For example, class 1 has a precision of 60.49% and a recall of 94.97%, while class 5 has a precision of 48.14% and a recall of 20.40%. This indicates that the model is more accurate at correctly identifying instances of class 1 than it is at correctly identifying instances of class 5. The f1-score for class 1 is 73.91%, while the f1-score for class 5 is 28.66%, further highlighting the difference in performance between the two classes. Overall, it appears that class 1 and class 5 have the largest differences in performance on this classification report.

The classification report for lda_svm_1000 seen in Table 5.26hows generally better performance than the classification report for lda_svm_200. For example, class 1 has a precision of 74.56% and a recall of 94.53% in the lda_svm_1000 report, compared to a precision of 60.49% and a recall of 94.97% in the lda_svm_200 report. Additionally, the f1-score for class 1 is 83.37% in the lda_svm_1000 report, compared to 73.91% in the lda_svm_200 report. This indicates that the model trained on

	precision	recall	f1-score	support
0	80.1397	81.9388	81.0293	980
1	60.4938	94.9780	73.9116	1135
2	51.4586	42.7326	46.6914	1032
3	66.7053	56.9307	61.4316	1010
4	60.6034	69.5519	64.7700	982
5	48.1481	20.4036	28.6614	892
6	72.2449	55.4280	62.7289	958
7	74.9458	67.2179	70.8718	1028
8	59.0597	61.9097	60.4511	974
9	43.7595	56.9871	49.5050	1009
accuracy			61.06200	10000
macro avg	61.7559	60.8078	60.0052	10000
weighted avg	61.8068	61.6200	60.4480	10000

Table 5.25: Classification report for lda_svm_200

	precision	recall	f1-score	support
0	69.9825	81.8367	75.4468	980
1	74.5657	94.5374	83.3722	1135
2	67.7530	38.2752	48.9164	1032
3	62.5000	51.9802	56.7568	1010
4	62.9767	68.9409	65.8240	982
5	49.6101	57.0628	53.0761	892
6	66.2461	65.7620	66.0031	958
7	64.8130	65.7588	65.2825	1028
8	59.4987	46.3039	52.0785	974
9	55.2239	62.3389	58.5661	1009
accuracy			63.6700	10000
macro avg	63.3170	63.2797	62.5323	10000
weighted avg	63.6121	63.6700	62.8514	10000

Table 5.26: Classification report for lda_svm_1000

a larger sample size of 1000 has improved performance in correctly identifying instances of class 1. Overall, it appears that several classes, including 1, 3, 5, and 9, have seen improvements in precision, recall, and f1-score when trained on a larger sample size.

The classification report for lda_svm_5000 has higher precision, recall, and f1-score values for each class compared to the lda_svm_1000 report. For example, the precision for class 0 is 92.00% in the lda_svm_5000 report, while it is 69.98% in the lda_svm_1000 report. Similarly, the recall for class 0 is 95.10% in the lda_svm_5000 report, while it is 81.83% in the lda_svm_1000 report. The f1-score for class 0 is also

	precision	recall	f1-score	support
0	92.0039	95.1020	93.5273	980
1	90.7577	96.0352	93.3219	1135
2	87.1277	79.3605	83.0629	1032
3	85.4692	83.8614	84.6577	1010
4	82.7977	89.2057	85.8824	982
5	80.6306	80.2691	80.4494	892
6	88.1178	87.4739	87.7947	958
7	86.9347	84.1440	85.5166	1028
8	80.6999	78.1314	79.3949	974
9	80.7843	81.6650	81.2223	1009
accuracy			85.6800	10000
macro avg	85.5324	85.5248	85.4830	10000
weighted avg	85.6542	85.6800	85.6202	10000

Table 5.27: Classification report for lda_svm_5000

higher in the lda_svm_5000 report (93.52%) compared to the lda_svm_1000 report (75.44%). Overall, it appears that the model trained on a larger sample size is more effective at correctly classifying instances in the MNIST data set.

Based on these classification report, the model with LDA is best at recognizing instances of class 1. This is because it has the highest precision and recall among all classes, as well as the highest f1-score. This indicates that the model is able to accurately identify instances of class 1 with a high degree of precision and recall. Furthermore, the difference in performance between class 1 and other classes is the largest, further highlighting the model's superior performance on this class. It also appears that the model is worst at recognizing classes 8, 9, 2 and 5.

Kernel PCA

In this part of the experiment it is expected that The performance of a SVM model using kPCA for dimensionality reduction is likely to differ from that of an SVM model without kPCA. kPCA can reduce the dimensionality of a dataset by projecting it onto a lower-dimensional space, which can improve the SVM model's decision boundary and performance. In contrast, an SVM model without kPCA may be more sensitive to the curse of dimensionality and overfitting, especially on high-dimensional datasets.

The values in Table 5.28 indicate that the model has relatively high precision and recall for most classes, with a few exceptions. Overall, the model has an accuracy of approximately 74%.

The classification report for kernel_pca_svm_200 and kernel_pca_svm_1000 show differences in the performance of the model on the two datasets. In general, the model trained on the larger dataset (kernel_pca_svm_1000) which can be seen in Table 5.29 appears to have higher precision, recall, and f1-scores for most classes.

	precision	recall	f1-score	support
0	75.2715	91.9388	82.7745	980
1	68.7965	97.7093	80.7426	1135
2	67.8387	63.5659	65.6328	1032
3	70.7317	74.6535	72.6397	1010
4	69.0129	81.8737	74.8952	982
5	76.9231	42.6009	54.8341	892
6	87.9245	72.9645	79.7490	958
7	87.5664	80.1556	83.6973	1028
8	79.3492	65.0924	71.5172	974
9	70.5394	67.3935	68.9306	1009
accuracy			74.4100	10000
macro avg	75.3954	73.7948	73.5413	10000
weighted avg	75.2395	74.4100	73.7969	10000

Table 5.28: Classification report for kernel_pca_svm_200

	precision	recall	f1-score	support
0	91.2745	95.0000	93.1000	980
1	92.6236	97.3568	94.9313	1135
2	87.3466	89.6318	88.4744	1032
3	88.4279	80.1980	84.1121	1010
4	84.7573	88.9002	86.7793	982
5	79.0487	78.2511	78.6479	892
6	90.7466	90.0835	90.4138	958
7	88.4837	89.6887	89.0821	1028
8	85.0627	76.5914	80.6051	974
9	83.2847	84.9356	84.1021	1009
accuracy			87.3000	10000
macro avg	87.1056	87.0637	87.0248	10000
weighted avg	87.2556	87.3000	87.2176	10000

Table 5.29: Classification report for kernel_pca_svm_1000

In general, the model trained on the larger dataset (kernel_pca_svm_5000) which can be seen in Table 5.30 appears to have higher precision, recall, and f1-scores for most classes. This suggests that, in this case, increasing the size of the dataset has led to a more accurate model.

Looking at the individual classes, some of the largest differences in performance can be seen in classes 1, 4, and 6. For class 1, the model trained on kernel_pca_svm_5000 has a precision of 95.03%, a recall of 97.88%, and an f1-score of 96.44%, while the model trained on kernel_pca_svm_1000 has a precision of 92.62%, a recall of 97.35%, and an f1-score of 94.93%. This indicates that the larger model is

	precision	recall	f1-score	support
0	93.2485	97.2449	95.2048	980
1	95.0385	97.8855	96.4410	1135
2	91.5187	89.9225	90.7136	1032
3	89.8204	89.1089	89.4632	1010
4	90.1174	93.7882	91.9162	982
5	88.5417	85.7623	87.1298	892
6	92.4742	93.6326	93.0498	958
7	92.2772	90.6615	91.4622	1028
8	90.8207	86.3450	88.5263	974
9	89.2108	88.5035	88.8557	1009
accuracy			91.4100	10000
macro avg	91.3068	91.2855	91.2763	10000
weighted avg	91.3817	91.4100	91.3762	10000

Table 5.30: Classification report for kernel_pca_svm_5000

more effective at correctly identifying and classifying examples from class 1.

For class 4, the model trained on kernel_pca_svm_5000 has a precision of 90.11%, a recall of 93.78%, and an f1-score of 91.91%, while the model trained on kernel_pca_svm_1000 has a precision of 84.75%, a recall of 88.90%, and an f1-score of 86.77%. This indicates that the larger model is more effective at correctly identifying and classifying examples from class 4.

For class 6, the model trained on kernel_pca_svm_5000 has a precision of 92.47%, a recall of 93.63%, and an f1-score of 93.04%, while the model trained on kernel_pca_svm_1000 has a precision of 90.74%, a recall of 90.08%, and an f1-score of 90.41%. This indicates that the larger model is more effective at correctly identifying and classifying examples from class 6.

ISOMAP embedding

This section presents the results of an experiment that was conducted to investigate the effects of sample size on the performance of ISOMAP and SVM. In this experiment, a set of data points representing a particular problem or phenomenon was divided into multiple groups, each containing a different number of samples.

Table 5.31 shows the evaluation metrics for a classifier trained on 500 instances of the MNIST dataset. The classifier has an overall accuracy of 69% and an f1-score ranging from 48% to 83%. The classifier performs relatively poorly across all classes, with f1-scores below 80% for all classes except 1 and 7.

Table 5.32 shows the evaluation metrics for a classifier trained on 1000 instances of the MNIST dataset. The classifier has an overall accuracy of 77% and an f1-score ranging from 63% to 89%. The classifier performs well on classes 1, 6, and 7, but relatively poorly on class 5, with an f1-score of 64%.

Table 5.33 shows the evaluation metrics for a classifier trained on 5000 instances

	precision	recall	f1-score	support
0	78.0198	80.4082	79.1960	980
1	62.0288	98.5903	76.1483	1135
2	64.1854	44.2829	52.4083	1032
3	59.2240	80.0990	68.0976	1010
4	65.5914	55.9063	60.3628	982
5	64.9083	31.7265	42.6205	892
6	75.5760	68.4760	71.8510	958
7	57.2629	46.4008	51.2628	1028
8	70.6505	47.9466	57.1254	974
9	45.5533	66.5015	54.0693	1009
accuracy			62.7600	10000
macro avg	64.3000	62.0338	61.3142	10000
weighted avg	64.1272	62.7600	61.5926	10000

Table 5.31: Classification report for isomap_svm_200

	precision	recall	f1-score	support
0	85.9023	93.2653	89.4325	980
1	81.2364	98.4141	89.0040	1135
2	82.5607	72.4806	77.1930	1032
3	79.5249	69.6040	74.2344	1010
4	73.7938	79.4297	76.5081	982
5	66.8719	60.8744	63.7324	892
6	86.7841	82.2547	84.4587	958
7	78.0198	76.6537	77.3307	1028
8	71.4912	66.9405	69.1410	974
9	66.5112	70.6640	68.5247	1009
accuracy			77.4600	10000
macro avg	77.2696	77.0581	76.9560	10000
weighted avg	77.4111	77.4600	77.2176	10000

Table 5.32: Classification report for isomap_svm_1000

of the MNIST dataset. The classifier has an overall accuracy of 88% and an f1-score ranging from 79% to 93%. The classifier performs particularly well on classes 1, 6, and 7, with f1-scores above 90%.

The figures above show that in general, it appears that increasing the number of components in the Isomap technique leads to an improvement in the model's performance. In the classification report for isomap_svm_5000, the model has the highest average f1-score of 87.95%, compared to 77.05% in isomap_svm_1000 and 72.47% in isomap_svm_200. This trend is also seen in other evaluation metrics, such as precision and recall.

	precision	recall	f1-score	support
0	91.3760	96.2245	93.7376	980
1	88.3046	99.1189	93.3998	1135
2	90.8021	82.2674	86.3244	1032
3	86.7126	87.2277	86.9694	1010
4	86.6667	90.0204	88.3117	982
5	85.5140	82.0628	83.7529	892
6	93.1987	91.5449	92.3644	958
7	88.0642	85.4086	86.7160	1028
8	87.4058	83.3676	85.3389	974
9	83.0000	82.2597	82.6282	1009
accuracy			88.1100	10000
macro avg	88.1045	87.9502	87.9543	10000
weighted avg	88.1141	88.1100	88.0348	10000

Table 5.33: Classification report for isomap_svm_5000

Additionally, it can be seen that the model's performance varies across the different classes in the dataset. For example, in isomap_svm_5000, the model has a high f1-score for classes 1, 6, and 7, but a relatively low f1-score for class 2.

5.4.3 Discussion of experiment 4

In terms of performance, Figure 5.13 shows that both baseline PCA and kPCA perform similarly well across all sample sizes, with consistently high F1 scores. In contrast, LDA performs worse overall, while ISOMAP has slightly lower F1 scores in smaller sample sizes but performs better in larger samples.

In terms of performance, Figure 5.13 shows that both baseline PCA and kPCA perform similarly well across all sample sizes, with consistently high F1 scores. In contrast, LDA performs worse overall, while ISOMAP has slightly lower F1 scores in smaller sample sizes but performs better in larger samples.

In terms of speed, Figure 5.14 shows that both PCA and LDA are the fastest when the sample size is larger than 2000, although they are slightly slower in smaller samples. Baseline PCA is the third slowest, while kPCA is the second slowest at a sample size of 5000. ISOMAP is the slowest overall, with longer runtimes for sample sizes of 2000 and above.

Overall, it appears that both PCA and kPCA are good choices for improving the performance of a SVM model on the MNIST dataset, as they offer both high performance and fast runtime. LDA may be a less optimal choice due to its lower performance, while ISOMAP may be a bad option for larger sample sizes but may be suitable for smaller samples due to its slower runtime.

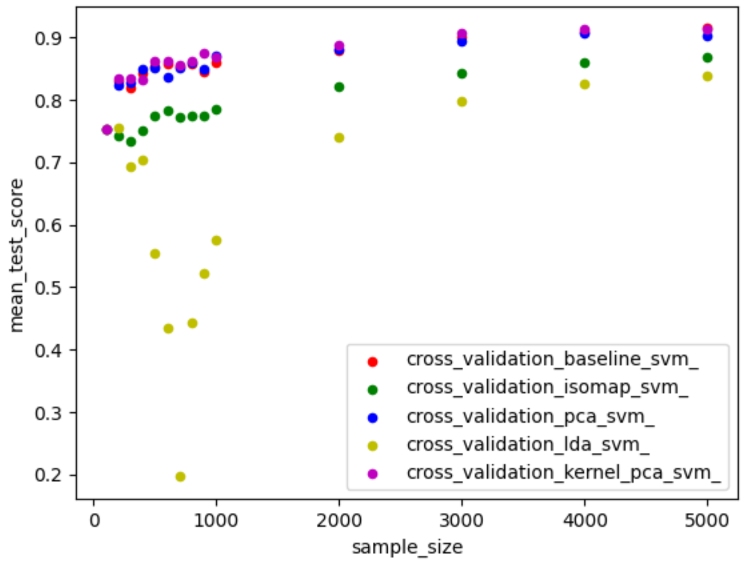


Figure 5.13

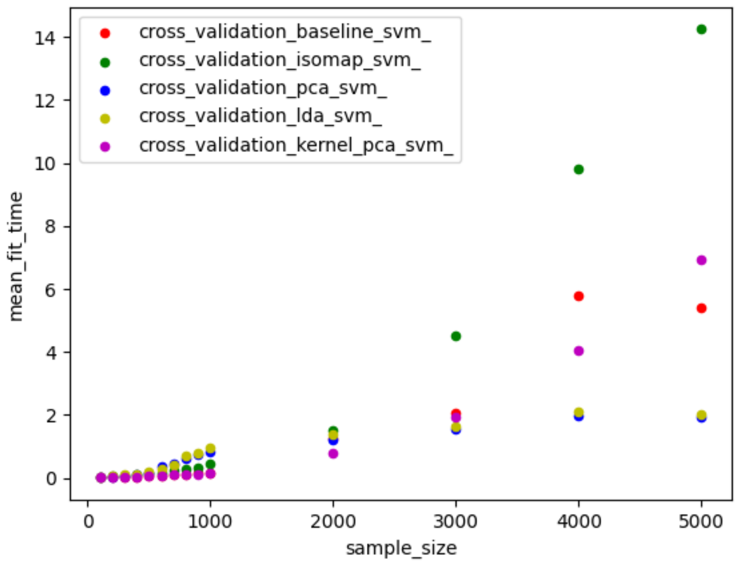


Figure 5.14

Chapter 6

Discussion

This chapter contains a discussion of the results from chapter 5 and relates the findings to the project's problem statement. The discussion aims to answer what went well, what could have been done better and what interesting details were discovered that were not expected and could prove helpful in future projects.

The discussion starts with the four experiments: their individual purposes, the results gathered from each experiment, and how they relate to the project's problem statement.

Then, the effect of choosing to work with MNIST is discussed. The chapter covers the effect of the chosen dataset; whether MNIST was a good choice of dataset in relation to the experiments and the problem statement, as well as what alternatives could have been used.

The chapter continues with a discussion of the insights on effect of memory limitations in the project and as a consideration in future projects.

The chapter ends with a summary of what was learned during the project with regards to dimensionality reduction and FE.

6.1 Discussion of experiments

This section will discuss the experiments we ran to compare the dimensionality reduction techniques. The experiments will be discussed in order from 1 to 4.

Each experiment has been conducted with a different purpose. Experiment 1 compared the linear methods' performance to the nonlinear methods' performance in regards to errors, time and F1 score. Experiment 2 tested the number of components before a significant drop-off in the accuracy score. Experiment 3 compared the two kernels in kPCA. Experiment 4 compared how sample size affects the score of each dimensionality reduction method, to determine if some methods are more or less reliant on data.

Experiment 1

In the first experiment, the Table 5.10 shows that at 15000 samples PCA and LDA are the fastest, with PCA being a little slower, but with higher accuracy. The lower accuracy to be expected as LDA performs a much larger reduction of dimensionality, and the difference in time is also to be expected as linear methods are typically faster than nonlinear methods.

Comparing kPCA at 15000 samples, and PCA at 60000 samples, it can be seen that they are similar in both time and accuracy, with PCA having gained only a small amount of additional accuracy despite quadrupling the amount of samples compared to the 15000 sample run.

ISOMAP is by far the slowest method and has lower accuracy than both PCA and kPCA, being $\approx$3% more accurate than LDA but taking $\approx$23 times longer to run. It is unlikely that ISOMAP would be a good choice for this dataset, as it is slower than all the other methods, and less accurate than all but LDA.

From this experiment it seems that MNIST is not nonlinear enough to benefit from ISOMAP and kPCA, but kPCA still has potential. However, because the SVM model does so well even before dimensionality reduction it may be that the advantages of a nonlinear approach is lost in the MNIST dataset.

This is further supported by the fact that no method is able to reach the accuracy of the SVM model, even with 60000 samples. This is likely because the SVM model is already able to find the relationships in the data, and the methods are not able to improve on this.

Experiment 2

From the results of the second experiment in Table 5.14 it can be seen that the stability of the methods are vastly different. LDA is the most accurate method at low dimensions, and retains its accuracy relative to itself even at very low dimensions.

ISOMAP however is the most stable method, retaining its accuracy relative to itself at as little as 20 dimensions - the difference between 50 dimensions and 20 dimensions is only 1%, suggesting that ISOMAP is able to find some nonlinear relationships in the data that PCA and kPCA are not able to find.

PCA and kPCA are less stable in relative accuracy than LDA at low dimensions, but does well at higher dimensions. This is to be expected, as the linear approach will always have a certain amount of data loss that scales directly with the number of linearly important dimensions.

Lastly, LDA and ISOMAP are almost equal in terms of accuracy at 9 dimensions, possibly suggesting that the data is not nonlinear enough to benefit from ISOMAP, or that the data is linear enough at this level of dimensionality that ISOMAP is not able to find any nonlinear relationships.

Experiment 3

The results from experiment 3 suggest that generally a sigmoid kernel performs better than a RBF kernel on MNIST. Both kernels give similar results at low dimensions as seen in experiment 2, but the sigmoid kernel performs better in general. Further testing would be needed to determine if this is a general trend for MNIST, or if the results are specific to this particular run, but it is expected to be a general trend because the sigmoid function intuitively can be closely similar to a linear function.

Interestingly, the kernels performed their best at different gamma values. Where the sigmoid kernel did best at $\gamma = 0.01$, the RBF kernel did best at $\gamma = 0.001$. The RBF kernel performed abysmally at $\gamma = 0.01$, and the sigmoid kernel performed more or less as well at $\gamma = 0.001$. This result showcases the importance of performing hyperparameter tuning.

The RBF kernel is considered a good default kernel [39]. However for MNIST RBF seems to be less accurate than a sigmoid kernel. This suggests that the kernels are not interchangeable, and that the best kernel for a given dataset is not necessarily the same as the best kernel for another dataset.

Because this experiment only compared two kernels, it is possible that there are other kernels that perform better than both of these. Further testing would be needed to determine if this is the case, but based on the previous experiments it is not expected to that there are any kernels that perform better than the sigmoid kernel for MNIST. This is because the sigmoid kernel performed close to the SVM model and PCA.

Experiment 4

Experiment 4 shows that LDA and PCA scales the best with the number of samples for time, followed by kPCA and ISOMAP. This was expected as LDA and PCA are both linear methods, while kPCA and ISOMAP are nonlinear methods, making them more computationally complex and expensive.

It was also expected and shown that kPCA is faster than ISOMAP. As a note to this, at low sample sizes it appears that the nonlinear methods are faster to train than the linear methods, indicating that the linear methods have a larger overhead than the nonlinear methods. This may be an implementation detail of the sklearn library, and not a general truth.

PCA and LDA seem to scale linearly with the number of samples for time, while kPCA and ISOMAP seem to scale exponentially with the number of samples. This is somewhat expected, as the kernel trick applied for kPCA scales quadratically with the number of samples, and ISOMAP is a nonlinear method that builds graphs of the data, which scales with both sample size and dimensionality.

When looking at accuracy of the models, each method scales with the number of samples, which is, of course, expected, and there does not seem to be any method that requires more samples to perform well than the others. With a size of at least 4000 elements, the accuracy of the methods relative to each other is fairly stable and run parallel, matching the results from experiment 1. This is interesting, because if this holds true for other datasets, it would mean that the best method for dimensionality reduction can be determined by a relatively small subset of the data.

However it can be observed that LDA performs worse with more data between 400 and 800 samples, and then increases again to match the trend. We do not know why this dip in accuracy happens, but it is likely an error to do with the amount of data required in LDA.

Summary of experiments

This section summarizes our findings and what we have learned from the experiments.

It makes sense that PCA is as popular as it is. It is fast and accurate, and it is a good default method for dimensionality reduction. Based on the examples in section 3.6 it is clear that it does not work well for all data, but for MNIST at least it seems to be a good default method. LDA also performed well for large levels of dimensionality reduction, and may be a good method for use on systems with limited resources, such as embedded systems.

ISOMAP and kPCA are both nonlinear methods, and both performed worse than PCA and LDA in their respective niches of accuracy and maximized dimensionality reduction. However, ISOMAP was able to retain its accuracy at low dimensions, and kPCA was able to perform well generally. PCA and kPCA performed very similarly, and it is likely that if resources are not a problem kPCA is the better method in general. This is particularly true as shown in experiment 3, where the choice of hyperparameters had a large impact on the accuracy of the model.

In the case that heavy hardware restrictions were imposed in a similar project in the future, we would choose PCA as a default method. It gives results fast and is indicated well on relatively little data. If PCA was not able to give good results, we would choose LDA as it has the lowest memory and Central Processing Unit (CPU) requirements of the tested methods, and generally performs decently. In particular because LDA removes a large amount of dimensions on data like MNIST, it is likely that it would be able to run well on restricted hardware, for example embedded systems.

In general it seems that nonlinear dimensionality reduction does not perform better than linear dimensionality reduction on MNIST when used with a SVM classifier. This is likely because MNIST is a relatively simple dataset, and that the nonlinear methods are not able to find any relationships in the data that SVM can't find on its own. However, it is possible that nonlinear dimensionality reduction methods would perform better on more complex datasets or with other classification models, and further testing would be needed to determine if this is the case.

6.2 The effect of the dataset

Based on the results presented in Table 5.10, and section 6.1 it appears that the MNIST dataset is somewhat well-suited for running both linear and nonlinear dimensionality reduction methods.

In general, the nonlinear dimensionality reduction methods performed similarly to the linear methods in terms of accuracy. kPCA had the highest accuracy, but it was slower to train compared to the linear methods. ISOMAP had a lower accuracy than the linear methods, but it was also slower to train. Overall, the results support the hypothesis that nonlinear dimensionality reduction methods work as well as linear methods on the MNIST dataset.

It is on the other hand possible that using a different dataset could have resulted in different conclusions regarding the performance of linear and nonlinear dimensionality reduction methods. The characteristics of the dataset, such as the number of data samples and the complexity of the data, can affect the performance of the different methods. For example, if the dataset has a larger number of data samples or is more complex, the nonlinear methods may outperform the linear methods in terms of accuracy. On the other hand, if the dataset is small or simple, the linear methods may perform better. Therefore, using a different dataset may have altered our conclusions about the performance of linear and nonlinear methods.

If the CIFAR-10 or Fashion MNIST (fashion-MNIST) datasets had been used instead of the MNIST dataset, the results and conclusions regarding the performance of linear and nonlinear dimensionality reduction methods may have been different. Both the CIFAR-10 and fashion-MNIST datasets are more complex than the MNIST dataset, with more data samples and more classes to classify.

In general, nonlinear methods tend to perform better on complex and high-dimensional datasets compared to linear methods. Therefore, it is likely that the nonlinear dimensionality reduction methods would have outperformed the linear methods in terms of accuracy on the CIFAR-10 or fashion-MNIST datasets. This could have led to different conclusions about the performance of linear and nonlinear methods.

It was also discussed the potential impact of using a different dataset on the performance of linear and nonlinear dimensionality reduction methods. The characteristics of the dataset, such as the number of data samples and the complexity of the data, can affect the performance of the different methods.

Using a more complex and high-dimensional dataset may have led to different conclusions about the performance of linear and nonlinear methods. In conclusion, the MNIST dataset is well-suited for testing dimensionality reduction methods, and the results support the hypothesis that nonlinear dimensionality reduction methods work as well as linear methods on the MNIST dataset.

6.3 Impact of memory limitations

During development and testing, it was noticed that the memory of the available machines was a limiting factor. This section will discuss the impact of the memory limitations on the project, and how it affected the results and time to fit the models. It will be discussed whether or not the impact of limited memory was significant or if it was minor and could be ignored. If the impact was significant, what could be done to improve the situation.

6.3.1 Impact on the results

As stated throughout the project, a limiting factor when running the nonlinear models was the memory of available machines. This meant that it was impossible to properly run the model with the entire dataset when reducing the data with a nonlinear method; instead, only a smaller section of the dataset was used. The linear methods were not affected by this limitation, as they could be run on the entire dataset.

Using a smaller dataset could have an impact on the results of the final model when using nonlinear methods, but through the experiments done, it was noticed that the general trend of the models did not seem to change much after a certain number of samples.

Experiment four covered the effect different sample sizes had on the model's accuracy. Figure 5.13 shows how at low sample sizes, the model with best accuracy can vary, but as the sample size increases, they all seem to fall into order. This would indicate that the different dimensionality methods are not significantly affected by the limited sample size, after a certain amount, and that the results should still be valid.

If however it was found that the results were significantly affected by the limited sample size, it might be considered finding a smaller dataset, use different methods to reduce the dimensionality, or try to gain access to a more powerfull machine that could handle the larger datasets, with nonlinear methods.

6.3.2 Impact on time to fit the models

The time to fit the models was also affected by memory limitations. The models could use multiple CPU cores, but this also increased memory usage since each job required its own segment of memory to work with. Since nonlinear methods already used much memory, running the models on multiple cores was only possible on machines with enough memory.

An example of how much faster it was with multiple cores was when running ISOMAP in the second experiment. The model was attempted to be run on a machine with only 8 GB of memory, which was not enough to run on multiple cores. The model was then run on a machine with 32 GB of memory, which was enough to run the model on three cores. On the first attempt it took about 3+ days to run, whereas on the second attempt it took about one day to run. This shows the impact of using multiple cores, and how it can reduce the time to fit the models.

Had time been a significant factor for the project other than comparing the different methods, it would have been considered to run the models on a machine with more memory, as a powerful machine would have reduced the time to fit the models because it would allow the models to run with multiple cores. As it was, all models were run on a machine with 32 GB of memory, which was enough to run the models in acceptable time.

The impact of limited memory was significant, but it was not a major factor in the project. As the results were not significantly affected by the limited sample size. Instead, it was generally the time it took to fit the model that could be reduced by having access to more memory. However while while the time to fit the models was not of importance for this particular project, it did showcase the importance of choosing the right model for the machine.

6.4 What we learned

To wrap the discussion up, the group will present the insights they have gained from the project.

Over the course of this project, we learned several important lessons about data analysis and ML. Practically, we learned how to construct and develop a comprehensive ML pipeline for data preprocessing, hyperparameter optimization, cross-validation, and evaluation. This allowed us to effectively train and evaluate our models on various datasets, and taught us how to do so in a systematic and efficient manner in future projects.

Second, we gained a deeper understanding of various dimensionality reduction techniques and their applications. We explored the differences between linear and nonlinear dimensionality reduction methods, and found a strong case for the use of linear methods in the case of MNIST. We witnessed the impacts of feature engineering on the performance and training time of our models, and learned how dimensionality reduction can be used to mitigate the curse of dimensionality or present a tradeoff between training time and computation requirements with little loss in accuracy. While we did not find an actual usecase for nonlinear dimensionality reduction on our data, the examples presented in section 3.6 shows that there at least exists a theoretical use.

Third, we gained experience with the SVM classification algorithm and its applications. We learned how to use classification algorithms to make predictions on our data and evaluate their performance.

Finally, we learned how to use various tools and techniques to reduce the dimensionality of our data and improve the performance of our models. We explored the use of PCA, kPCA, LDA, and ISOMAP for dimensionality reduction, and learned about the strengths and weaknesses of each method.

Overall, this project has given us valuable experience in data analysis and ML, and has provided us with a solid first step for further study and research in these fields. It has also provided a good introduction to working theoretically in a scientific setting.

Chapter 7

Conclusion

Based on the discussion in chapter 6, the results from chapter 5 and the problem statement in section 2.4, we conclude:

The project provided valuable insights into data analysis and ML. We learned how to construct and develop a comprehensive ML pipeline and gained a deeper understanding of various dimensionality reduction techniques and their applications. We also gained experience with the SVM classification algorithm and learned how to use it to make predictions and evaluate performance.

We found that the linear methods, PCA and LDA, performed particularly well in terms of speed on the MNIST dataset when used with an SVM classifier, with an acceptable loss of accuracy for PCA in particular, while LDA has an advantage at low dimensions.

Nonlinear methods, such as ISOMAP and kPCA, did not outperform linear methods in this case but may have the potential to be useful in certain situations.

Overall, this project has given us a strong foundation for further study and research in data analysis and provided a good introduction to working theoretically in a scientific setting with ML.

Chapter 8

Future works

With the insights gained from the project, we have determined several points of interest worth looking into in the future.

Initially, the project included plans for data augmentation in the pipeline, such as slight rotation of the images or artificial errors by whitening pixels of the numbers, making this a prime candidate for future work. These augmentations would artificially increase the size of the dataset, allowing for more splits, more model accuracy, and possibly making the data more nonlinear.

The MNIST dataset is relatively simple, so it may be interesting to explore more complex datasets and study the impact of dimensionality reduction on those. Some potential datasets already mentioned in the report are the CIFAR-10 and fashion-MNIST datasets, which may increase the potential of the nonlinear approaches. Alternatively, other data types, for example, audio, would be interesting to look at because it is very different from MNIST.

Different dimensionality reduction methods would also be interesting. Especially methods not covered in depth for this report, such as Non-negative Matrix Factorization (NMF), would be exciting.

Aside from different methods, kPCA has different kernels that would be interesting to explore. Section 1 shows that the choice of the kernel has a significant impact on the final accuracy. It would be interesting to explore the different kernels available.

Lastly, having access to a more powerful computer would allow us to use the nonlinear dimensionality reductions of the report on the full MNIST dataset and measure the memory usage to more accurately compare the methods' computational requirements.

Acronyms

AI Artificial Intelligence. 3

CIFAR-10 Canadian Institute For Advanced Research. 7, 75, 81

cMDS Classical Multi Dimensional Scaling. 16, 17

CPU Central Processing Unit. 74, 89

CV Computer Vision. 1

FA Factor Analysis. 15, 17

fashion-MNIST Fashion MNIST. 75, 81

FE Feature Engineering. 4, 7, 22, 23, 71

ISOMAP Isometric Feature Mapping. 16–21, 26, 27, 29, 31, 34, 40, 41, 43–46, 50–53, 66, 68, 72–77, 79

kPCA Kernel Principal Component Analysis. 1, 16–21, 26, 27, 29, 31, 34, 40, 43–45, 49, 52, 53, 55, 57, 64, 68, 71–75, 77, 79, 81

KPCA-R Kernel PCA with RBF kernel. 55–57

KPCA-S Kernel PCA with Sigmoid kernel. 55–57

LDA Linear Discriminant Analysis. 15, 17–20, 26, 27, 29, 31, 34, 36, 37, 43–46, 48, 49, 51, 53, 62, 64, 68, 72–74, 77, 79

MDS Multi Dimensional Scaling. 16

ML Machine Learning. iii, 1, 3–13, 22, 24, 34, 77, 79

MNIST Modified National Institute of Standards and Technology. iii, 1, 6, 7, 10, 11, 14, 17, 25, 26, 28, 33, 46, 48, 57, 58, 62, 64, 68, 71–75, 77, 79, 81, 83

NMF Non-negative Matrix Factorization. 81

NN Neural Network. 10, 11

OS Operating System. 89

OvA One-versus-All. 11

OvO One-versus-One. 11, 12

PCA Principal Component Analysis. 1, 15–20, 22, 26, 27, 29, 30, 34, 37–40, 43–45, 47–49, 51–54, 57, 60, 62, 68, 72–74, 77, 79

RAM Random Access Memory. 89

RBF Radial Basis Function. 31, 53, 73

sklearn scikit-learn. 12, 14, 17, 18, 26–30, 73

SVM Support Vector Machine. iii, 1, 10–12, 22, 26, 27, 29–31, 34–45, 48–50, 53, 58–60, 62, 64, 66, 68, 72–74, 77, 79

Bibliography

[1] Daniel Runge Petersen. *AAU-Dat templates*. URL: https://github.com/AAU-Dat/templates (visited on 08/17/2022).

[2] Zhun Cheng and Zhixiong Lu. “A Novel Efficient Feature Dimensionality Reduction Method and Its Application in Engineering”. In: *Complexity* (2018). DOI: 10.1155/2018/2879640.

[3] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. 2013. URL: https://hastie.su.domains/ISLR2/ISLRv2_website.pdf.

[4] *MNIST database*. Yann LeCun. URL: http://yann.lecun.com/exdb/mnist/ (visited on 10/04/2022).

[5] *Domo Releases 10th Annual “Data Never Sleeps” Infographic*. Sept. 2022. URL: https://www.proquest.com/wire-feeds/domo-releases-10th-annual-data-never-sleeps/docview/2716042192/se-2%7D.

[6] Alon Halevy, Peter Norvig, and Fernando Pereira. “The Unreasonable Effectiveness of Data”. In: (2009). DOI: 10.1109/MIS.2009.36.

[7] Etham Alpaydin. *Introduction to Machine Learning. Fourth*. 2020.

[8] Richard Ernest Bellman. “Rand Corporation (1957)”. In: *Dynamic programming* ().

[9] John A. Lee and Michel Verleysen. *Characteristics of an Analysis Method*. Ed. by John A. Lee and Michel Verleysen. New York, NY: Springer New York, 2007, pp. 17–45. ISBN: 978-0-387-39351-3. DOI: 10.1007/978-0-387-39351-3_2. URL: https://doi.org/10.1007/978-0-387-39351-3_2.

[10] Altexsoft. *Machine Learning Pipeline: Architecture of ML Platform in Production*. URL: https://www.altexsoft.com/blog/machine-learning-pipeline/ (visited on 11/23/2022).

[11] G. Thippa Reddy, M. Praveen Kumar Reddy, Kuruva Lakshmanna, Rajesh Kaluri, Dharmendra Singh Rajput, Gautam Srivastava, and Thar Baker. “Analysis of Dimensionality Reduction Techniques on Big Data”. In: *IEEE Access* 8 (2020), pp. 54776–54788. DOI: 10.1109/ACCESS.2020.2980942.

[12] Alice Zheng and Amanda Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. 2018. ISBN: 1491953241.

[13] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. *Application of dimensionality reduction in recommender system-a case study*. Tech. rep. Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[14] Dan Margalit, Joseph Rabinoff, and L Rolen. “Interactive linear algebra”. In: (2017).

[15] Mordecai Avriel. *Nonlinear programming: analysis and methods*. 2003.

[16] Laurens van der Maaten, Eric Postma, and H. Herik. “Dimensionality Reduction: A Comparative Review”. In: *Journal of Machine Learning Research - JMLR* 10 (Jan. 2007).

[17] John C. Langford Joshua B. Tennenbaum Vin de Silva. *A Global Geometric Framework for Nonlinear Dimensionality Reduction*. URL: `https://wearables.cc.gatech.edu/paper_of_week/isomap.pdf` (visited on 10/11/2022).

[18] Matteo Kimura. *Introductory Datasets*. URL: `https://lamfo-unb.github.io/2019/05/17/Introductory-Datasets/` (visited on 11/15/2022).

[19] *Fashion-MNIST*. Zalando Research. URL: `https://github.com/zalandoresearch/fashion-mnist` (visited on 10/31/2022).

[20] DataRobot. *The importance of machine learning data*. URL: `https://www.datarobot.com/blog/the-importance-of-machine-learning-data/` (visited on 11/23/2022).

[21] Zhang Shichao, Zhang Chengqi, and Yang Qiang. “Data preparation for data mining”. In: (2003). DOI: `10.1080/713827180`. URL: `https://doi.org/10.1080/713827180`.

[22] G. Thippa Reddy, M. Praveen Kumar Reddy, Kuruva Lakshmanna, Rajesh Kaluri, Dharmendra Singh Rajput, Gautam Srivastava, and Thar Baker. “Analysis of Dimensionality Reduction Techniques on Big Data”. In: *IEEE Access* (2020). DOI: `10.1109/ACCESS.2020.2980942`.

[23] Bichitrananda Behera, G. Kumaravelan, and Prem Kumar B. “Performance Evaluation of Deep Learning Algorithms in Biomedical Document Classification”. In: *2019 11th International Conference on Advanced Computing (ICoAC)*. 2019. DOI: `10.1109/ICoAC48765.2019.246843`.

[24] Anyscale. *What is hyperparameter tuning?* URL: `https://www.anyscale.com/blog/what-is-hyperparameter-tuning` (visited on 02/08/2022).

[25] Anyscale. *What is hyperparameter tuning?* URL: `https://www.anyscale.com/blog/what-is-hyperparameter-tuning` (visited on 02/08/2022).

[26] *What is computer vision?* IBM. URL: `https://www.ibm.com/topics/computer-vision` (visited on 09/27/2022).

[27] Sanghyeon An, Minjun Lee, Sanglee Park, Heerin Yang, and Jungmin So. *An Ensemble of Simple Convolutional Neural Network Models for MNIST Digit Recognition*. 2020. DOI: `10.48550/ARXIV.2008.10400`. URL: `https://arxiv.org/abs/2008.10400`.

[28] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. "Multi-column deep neural networks for image classification". In: 2012. DOI: 10.1109/CVPR.2012.6248110.

[29] Sebastian Schlag, Matthias Schmitt, and Christian Schulz. *Faster Support Vector Machines*. 2018. DOI: 10.48550/ARXIV.1808.06394. URL: https://arxiv.org/abs/1808.06394.

[30] Leonardo Moreira, Christofer Dantas, Leonardo Oliveira, Jorge Soares, and Eduardo Ogasawara. "On Evaluating Data Preprocessing Methods for Machine Learning Models for Flight Delays". In: 2018. DOI: 10.1109/IJCNN.2018.8489294.

[31] *sklearn.decomposition.PCA*. Nov. 2022. URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html#sklearn.decomposition.PCA.fit.

[32] TowardsAi. *How, When, and Why Should You Normalize / Standardize / Rescale Your Data?* URL: https://towardsai.net/p/data-science/how-when-and-why-should-you-normalize-standardize-rescale-your-data-3f083def38ff (visited on 11/30/2022).

[33] Jamie DeCoster. *Overview of factor analysis*. 1998.

[34] Alaa Tharwat, Tarek Gaber, Abdelhameed Ibrahim, and Aboul Ella Hassanien. "Linear discriminant analysis: A detailed tutorial". In: *AI communications* 30.2 (2017), pp. 169–190.

[35] Quan Wang. *Kernel Principal Component Analysis and its Applications in Face Recognition and Active Shape Models*. 2012. DOI: 10.48550/ARXIV.1207.3538. URL: https://arxiv.org/abs/1207.3538.

[36] Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, and Mark Crowley. *Multidimensional Scaling, Sammon Mapping, and Isomap: Tutorial and Survey*. 2020. URL: https://arxiv.org/abs/2009.08136.

[37] Jan de Leeuw. "Multidimensional Scaling". In: (2000).

[38] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. "API design for machine learning software: experiences from the scikit-learn project". In: (2013), pp. 108–122.

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

[40] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[41] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: `http://archive.ics.uci.edu/ml`.

[42] Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. "Tunability: Importance of hyperparameters of machine learning algorithms". In: *The Journal of Machine Learning Research* (2019).

[43] Matthias Feurer and Frank Hutter. *Automated Machine Learning: Methods, Systems, Challenges*. 2019. ISBN: 978-3-030-05318-5. URL: `https://doi.org/10.1007/978-3-030-05318-5_1`.

[44] Marc Claesen and Bart De Moor. "Hyperparameter Search in Machine Learning". In: (2015). URL: `http://arxiv.org/abs/1502.02127`.

[45] Li Yang and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice". In: *Neurocomputing* (2020).

[46] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013. URL: `https://faculty.marshall.usc.edu/gareth-james/ISL/`.

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

[48] Payam Refaeilzadeh, Lei Tang, and Huan Liu. *Cross-Validation*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 532–538. URL: `https://doi.org/10.1007/978-0-387-39940-9_565`.

[49] *Confusion Matrix*. 2022. URL: `https://subscription.packtpub.com/book/data/9781838555078/6/ch06lvl1sec34/confusion-matrix`.

[50] Margherita Grandini, Enrico Bagli, and Giorgio Visani. *Metrics for Multi-Class Classification: an Overview*. 2020. DOI: `10.48550/ARXIV.2008.05756`. URL: `https://arxiv.org/abs/2008.05756`.

Appendix A

PC specifications

Table A.1 shows the specifications of the PCs used in the experiments. PC-3 was not actually used in any of the experiments in the end, only during development.

	CPU	RAM	OS
PC-1	AMD Ryzen 5600X	32GB 3200MHz	Windows 10 & WSL
PC-2	Intel Core I5-10300H	16GB 2900MHz	Windows 10 & WSL
PC-3	AMD Ryzen 5 3600	16GB 3600MHz	Windows 10 & WSL
PC-4	Intel Core I5-4460	8GB 1600Mhz	Windows 10 & WSL

Table A.1: The specifications of the PCs used in the experiments.