Linear versus Non-linear 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].

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

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Linear versus Non-linear 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: 85

Date of Completion:
December 13, 2022

Abstract:
This is the abstract...

Contents

Preface

This is the preface. You should put your signatures at the end of the preface.

Aalborg University, December 13, 2022

Author 1
<username@student.aau.dk>

Author 2
<username@student.aau.dk>

Author 3
<usernamestudent.aau.dk>

Author 4
<username@student.aau.dk>

Author 5
<username@student.aau.dk>

Author 6
<username@student.aau.dk>

Chapter 1

Introduction

Chapter should probably contain: The initial problem (If we have one), motivation, and the scope or background of the project or theme. Report outline at the end.

In this project we will study some common methods of dimensionality reduction using the MNIST dataset for digit recognition.

Keywords: dimensionality reduction, linear methods, nonlinear methods, MNIST, Computer Vision (CV), Machine Learning, machine intelligence (artificial intelligence).

Use sources [2] and [3] for superficial overview and explanations of umbrella terms.

Notes to self: Humans vs computers in Neural Network (NN). Why are humans good with little training, and computers only accceptable with much more training? Consider perhaps domains (recongnizing epsilon vs. recognizing a 3)

On theory driven projects

> The overall purpose of the project module is for the student to acquire the ability to analyze and evaluate the application of methods and techniques within database systems and / or machine intelligence to solve a specific problem. **This includes analyzes of the formal properties of the techniques and an assessment of these properties in relation to any requirements for the solution to the specific problem**. [...]
>
> In this project module, the project work is primarily driven by theoretical and analytical considerations about the methods and techniques used. For a specific problem area, a project could, for example, be based on specific performance requirements for the developed software solution, and the project work can thus be guided by the solution's algorithmic time / space complexity as well as formal analyzes and considerations of its theoretical properties and performance guarantees. [4]

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.

For now this is taken directly from the project proposal. We may rewrite this partially at a later time.

1.2 Report outline

The report is structured as follows:

- **Introduction** - This chapter
- **Problem Analysis** - Chapter 2
- **Theory** - Chapter 3
- **Implementation** - Chapter 4
- **Results** - Chapter 5
- **Discussion** - Chapter 6
- **Conclusion** - Chapter 7

The introduction describes the initial problem and the motivation for the project.

The Problem Analysis chapter dives into the initial problem and leads to a final problem statement.

The Methodology chapter describes the methods and theory used to explore the problem statement. It also describes the data used and how it was collected/created.

The Results chapter is an evaluation of the results of the project.

The Discussion chapter is a discussion of the results and the project as a whole.

The Conclusion chapter provides a summary of the project and the results. It also provides perspective and reflection on the project and the process.

Chapter 2

Problem analysis

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

Machine Learning (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 so-called "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].

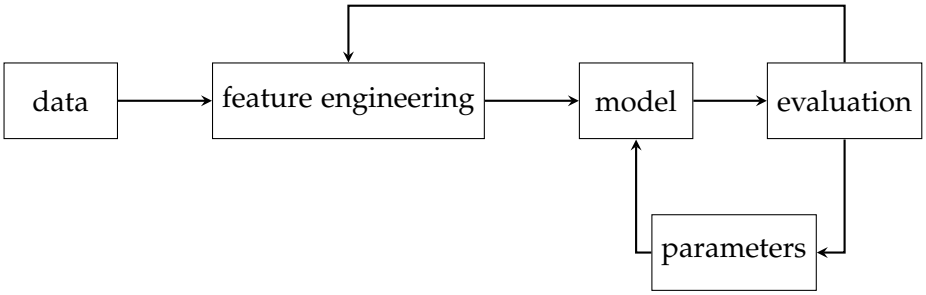


Figure 2.1: Simplified machine learning pipeline

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 [11].

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 [11].

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. [12].

Therefore, when reducing features in a dataset, it is crucial to find the features that are more relevant to the output of the model [13]. 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 [14], 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 [11]. 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 [11].

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.

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 [15]. 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 [16]. 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 [17, 18], 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 [18].

The research paper [17] compares the performance of linear and nonlinear dimensionality reduction methods with some ML models.

According to Laurens [17], 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" [17].

In this project, the focus will be on whether linear or nonlinear methods positively impact the performance of the 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

Rewrite sectiop, potentially include stuff from: The following is an example of how the problem statement should look like This problem aims to answer the question: Are nonlinear reduction techniques better than linear ones when doing machine learning? As a part of this question we need to answer the following smaller questions * where in the pipeline are we gona use the methods? * what data are we gonna use? is it only synthetic or real world or are you gonna use both? * how are we gonna evaluate performance? after classification? as a way to explain errors ? And so on

This project aims to explore the impact of dimensionality reduction, comparing linear and non-linear dimensionality reduction techniques. This will be done in regards to

the performance of a machine learning model, for the computer vision problem of image classification and recognition, of the mnist dataset.

Chapter 3

Theory

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, we have chosen to work with the Modified National Institute of Standards and Technology (MNIST) dataset because it is commonly used in image recognition tasks, specifically for recognizing handwritten digits. This dataset is well-documented, which allows us to compare our results to other similar projects, and it is small in size, making it easy to work with and not requiring a lot of computational power[19]. Additionally, the preprocessing of the MNIST dataset has further reinforced our decision to use it.

It was considered to use the Iris and CIFAR-10 datasets, but the Iris dataset may have too few data samples [20] and the CIFAR-10 dataset could be too complex for this project[21]. The MNIST dataset consists of images, and each image is composed of pixels. In a grayscale image, each pixel has a value between 0 and 255, with 0 representing white and 255 representing black.

In 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 machine learning 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 how machine learning models continuously learn. 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 machine learning 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 [22], 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 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 [23], 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 [24].

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 [25].

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 [26]. This step is purely for hyperparameters. As mentioned in the previous 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 [27].

3.3 Classification

When making a model, it is only sometimes possible to predict the 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 [28]. This is done by training a model on a labeled dataset and then using the model to predict the category of new, unlabeled data [28].

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 [28].

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 [28].

The model is trained by finding the best parameters for the model, such as weights in a neural network or parameters in models, such as Support Vector Machine (SVM) [28]. 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 ones the group has looked into is [3, 19, 29, 30]. 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 [28].

SVM classifies data by finding a mapping (hyperplane) that separates the classes in data [31]. SVM is also known as a large-margin classifier, which means that it relies on finding "a maximum-margin hyperplane to separate classes" [31]. 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 [31]. The kernel function that SVM resembles the way Kernel Principal Component Analysis (kPCA) works, as it also uses a kernel function.

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." [28]. 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. On the downside of SVM, [28] states "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" [28]. 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 [19].

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) [28].

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 [28].

One-vs-All

However, OvA is faster than OvO, as it only uses one class to distinguish if the data is similar. For example, if there are five classes, the model is trained on five variations of the model, one for each class. This makes OvA good to distinguish between the current class that is being modeled from the other classes. However, in OVA, it is harder to distinguish between the other classes that are not being trained. Then the model is then evaluated on all models, and the class with the highest score is chosen as the predicted class [28].

One-vs-One vs One-vs-All

OvO is more computationally expensive than OvA, but is more accurate. The choice of OvO or OvA is, therefore, a trade-off between accuracy and computational cost [28]. OvO is chosen as the method for multi-class classification.

We use OvO because it is faster than OvA, make that clear.

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. [23]

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 [23].

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 [32] 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 [32], 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 [13].

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 [13].

Variance scaling Variance scaling is a more robust way to scale the data, but it is more complex. 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 [13].

Data augmentation

This section will describe data augmentation, its relevance, and which specific augmentation will be used.

> Data augmentation in data analysis are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic from existing data. It acts as a regularizer and helps reduce overfitting when training a machine learning model. [33]

Data augmentation is done for many reasons, typically to make a model more general and less overfitted. Overfitting happens because there is only a limited amount of data, often from multiple sources in the real world. So augmentation is used to make a model less reliant on these less-than-ideal data sources [34].

In the case of this project, augmentation is chosen, so it is possible to show how the FE and the model handle different data shapes while still having a baseline.

Augmentation helps make more general inferences and intuition about FE and ML. It will, in particular, show which FE methods best handle different data shapes. It will also help show how general augmentation can affect performance ML models and FE.

There was chosen augmentations from the following list: Rotation, removing pixels, scaling. These were selected because they are all well-suited for images since they create variations of images [35]. There exist others, but they were not considered for this project.

All the chosen preprocesses for the project are now described. The following section concerns the next point in FE dimensionality reduction.

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, normalize it, and augment 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 scikit-learn (sklearn) did not accept the data in this format because the function `fit` expects: "X: array-like of shape (n_samples, n_features)" [36], 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 [37]. Both methods would have been good choices for normalization, variance scaler was chosen, but either would be acceptable for this project.

Data augmentation

As mentioned in subsubsection 3.4.3, data augmentation increases the amount of data and makes the model more general. Augmentation creates new data from the existing data by applying different augmentations to the original data and using the new data to train the model. For this project, rotation augmentations were chosen. They were also deemed to represent real-world errors in data. For example a number could become more or less rotated due to a writers writing style. These are some of the cases thought of when deciding on the augmentations.

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 [17].

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: Principal Component Analysis (PCA), Factor Analysis (FA), Non-negative Matrix Factorization (NMF), 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 [17].

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.

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 [38].

Non-negative matrix factorization

NMF constructs approximate factorizations of a non-negative data matrix V into two non-negative matrices W and H such that $V \approx WH$. The non-negativity constraints permit the intuition that data may represent parts forming a whole, which can be thought of as a sum of parts [39]. For example, a text document may represent a combination of topics, and each may represent a combination of words.

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 [40].

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 [18]. 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 [41].

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 dat 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.

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 [41].

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 [18].

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 [42].

In the second step, ISOMAP finds the Geodesic distance between each point [42]. 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 [43].

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 [43], 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 scikit-learn's implementation of the methods. The hyperparameters of the respective methods will also be presented.

Linear methods

The reason for choosing PCA is, among others, because it is a popular dimensionality reduction method [17] 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" [44].

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 $dimensions - 1$ [44, 45]. 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. The number of components is the hyperparameters of the LDA method.

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 [17]. 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 [46].

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. Additionally, PCA has the hyperparameter of whether the components will be whitened or not, ISOMAP has the number of neighbors, and kPCA has the kernel and the kernel coefficient.

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 Scikit-learn [45].

3.6.1 Linear data example

As linear data, we will use the Iris dataset, 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 [47]. The dataset is a simple and well-known dataset, which has been used in many machine learning papers [47] and should give intuition as to how the methods behave on the dataset.

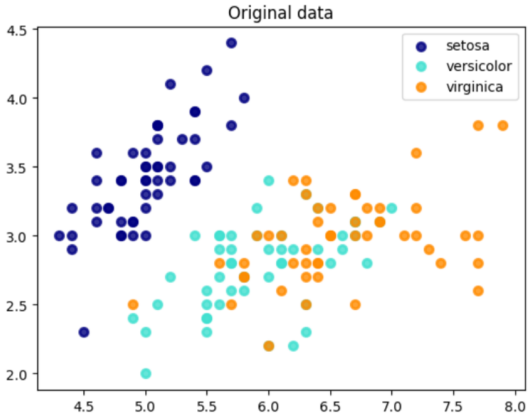


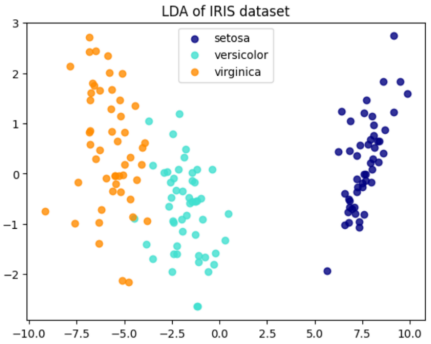
Figure 3.1: Linear data as Iris dataset

Figure 3.1 represents the iris flowers on a 2D plot. According to [47], only one class is separable, which can further be solidified by the fact that the species Setosa is the only visually linearly separable class.

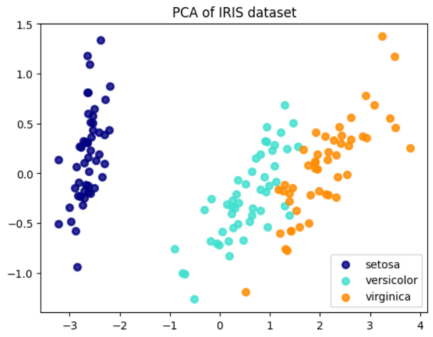
Linear methods

In figure 3.2b, 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.

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

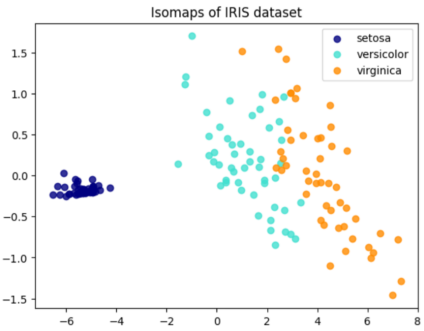


(a) LDA on iris

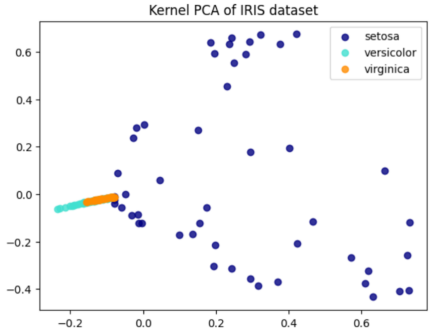


(b) PCA on iris

Figure 3.2: both linear methods on iris



(a) Isomap on iris



(b) KernelPCA on iris

Figure 3.3: both nonlinear methods on iris

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.

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, 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, we will construct two classes of circles, an inner- and an outer circle, which will look like in Figure 3.4.

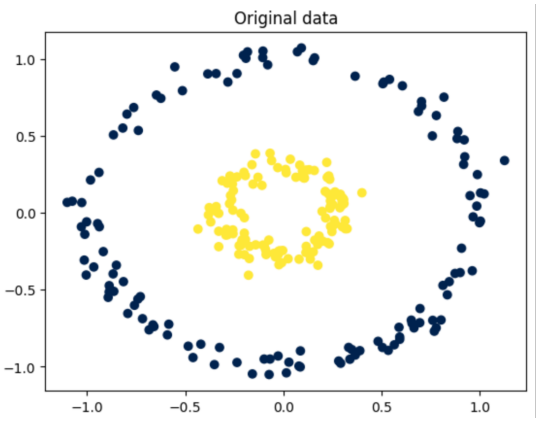


Figure 3.4: Nonlinear data as two circles

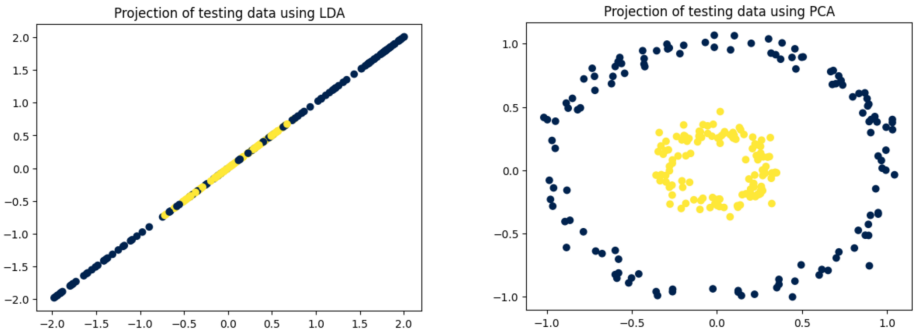
Linear methods

In Figure 3.5b, it can be seen that PCA's transformation could have managed to map the nonlinear data. 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.

Nonlinear methods

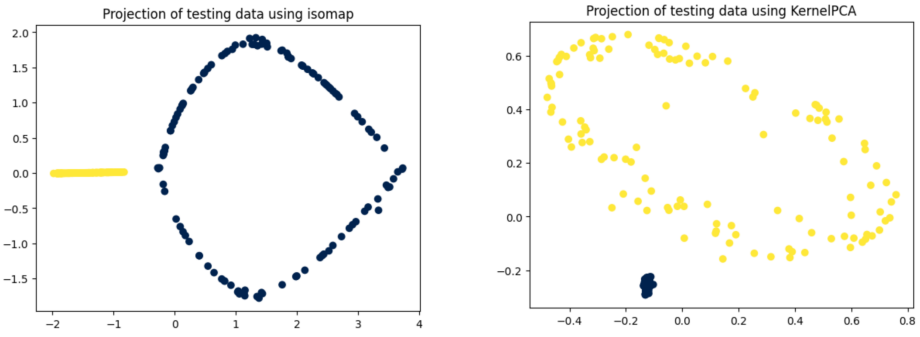
In Figure 3.6a, ISOMAP has separated the data points from the circles. Based on the first projection ISOMAP is capable of separating the data. On the second projection, the method can capture more information about the outer circle, thus approximating the original data. The outer circle has a higher variance than the inner circle. The second projection, ISOMAP, revealed little information about the inner circle.

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.



(a) LDA on circles **(b)** PCA on circles

Figure 3.5: both linear methods on circles



(a) Isomap on circles **(b)** KernelPCA on circles

Figure 3.6: both nonlinear methods on circles

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 [48]. 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.

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 [48].

Instead, the process of finding optimal hyperparameters can be given to the computer [49], 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 [50], 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 [49]. 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" [49]. 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 [51]. 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 [51].

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 hyper parameters

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 [52].

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(comprising 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. 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 [53]. 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. [54]

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 [54].

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 [54].

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 [54].

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 [52]. 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 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` [52].

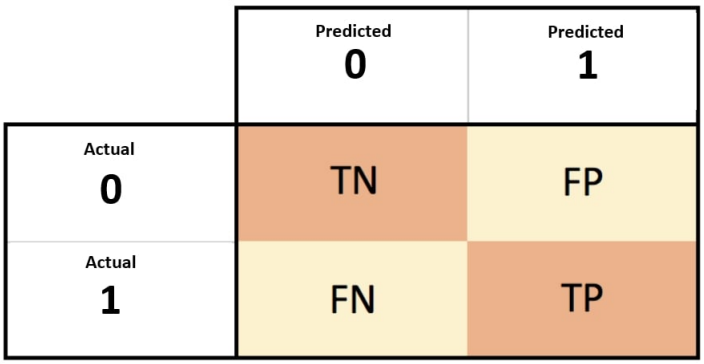


Figure 3.7: Confusion matrix [55]

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 [56]. 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 [56]. 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 [56]. 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. [56]. 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 [56]. 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 [52].

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.

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 data sets, the pre-processing, the dimensionality reduction, the machine learning loop, and the output.

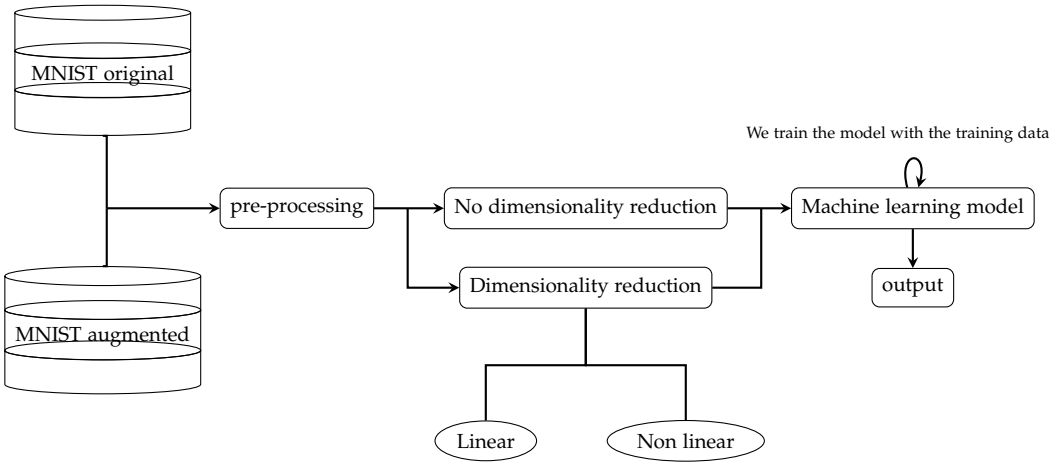


Figure 4.1: Overview of the pipeline used for the project.

The first segment, the data sets, 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.

We should probably add a subsection on the data augmentation.

4.1.1 Data augmentation

Figure 4.2: Data augmentation pipeline segment.

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/`[19] 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 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.

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

```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.

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.

The hyperparameters

The following hyperparameters are used for the tuning loop:

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.

```
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.

Chapter 5

Results

Describe the results of the project.

5.1 Experiment 1

5.1.1 Rules and overview of experiment

The first experiment was to find the best configuration for each method. The methods were SVM, PCA, LDA, kPCA, and ISOMAP. The methods were tested with 15000 samples, and the best configuration was found. The best configuration was then tested with 60000 samples. The results are shown in the following sections. The group chose this experiment because finding the best configuration for each method on the same number of samples was essential. The group did this to make it possible to compare the methods on the same number of samples.

Experiment 1 tested the best configuration for the methods. The pipeline found the best configuration by testing all the possible configurations on 15000 samples. The experiment also ran with 60000 samples, but only for SVM and the two linear methods PCA and LDA. The two nonlinear methods, kPCA and ISOMAP, were not tested with 60000 samples, as the computers used did not have enough power to do all 60000 samples. When used alone, SVM handled the raw data without any dimensionality reduction. The results shown are only for the best configurations for each method on the number of samples used in the experiment. Table 5.1 shows the best parameters for each method.

Every method was tested with the same number of samples, as this was the only way to compare the methods. Besides LDA and SVM, LDA can only use up to 9 components, and SVM does not need any hyperparameters.

The group chose this experiment because finding the best configuration for each method on the same number of samples was essential. The group did this to make it possible to compare the methods on the same number of samples.

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

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

	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

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. The SVM is best at recognizing zeros and one's in pictures, as the model has the f1-score in these classes, with the scores 96.5% and 97.7%. 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 89.6%, eight being 91.5%, and three being 91.4%. With an average f1-score of 93.43%, the baseline SVM with 15000 samples is a good model.

SVM with 60000 samples

	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

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 at recognizing zeros and ones in pictures, as the model has the f1-score in these classes, with scores of 97.3% and 98.1%. 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 91.2%, eight being 92.3%, and three being 93.1%. 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

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. 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 94.9% and 95.3%. 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 83.1%, eight being 82.0%, and three being 85.9%. 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.

	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

LDA with 60000 samples

	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

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 94.8% and 95.6%. 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 83.9%, eight being 83.3%, and three being 86.4%. SVM using PCA as the dimensionality

reduction method has an average f1-score of 89.16%.

PCA with 15000 samples

	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

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. 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 96.1% and 97.6%. 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 88.1%, eight being 89.2%, and three being 89.9%. SVM using PCA as the dimensionality reduction method has an average f1-score of 92.25%.

PCA with 60000 samples

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 96.5% and 97.7%. 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 89.1%, eight being 90.6%, and three being 90.5%. SVM using PCA as the dimensionality reduction method has an average f1-score of 93.14%.

	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

	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

kPCA with 15000 samples

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. 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 96.2% and 97.8%. 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 87.1%, nine being 90.0%, and three being 89.4%. SVM using kPCA as the

dimensionality reduction method has an average f1-score of 92.22%.

ISOMAP 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

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. 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 95.1% and 96.6%. 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 87.8%, eight being 87.4%, and three being 85.1%. SVM using ISOMAP as the dimensionality reduction method has an average f1-score of 90.50%.

5.1.3 Discussion 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.

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.

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.

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

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.

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 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, ISOMAPP 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 data set 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.

5.2.1 Rules and evaluation of the experiment

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

The second experiment will be done on a subset of the entire data set. 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 non-linear methods, as they need more memory than was available.

The values from the mnist dataset will be normalized using standard scalar for each dimensionality reduction method. Ensuring the values are on the same scale so that different scales do not skew the results.

Hyperparameter tuning will be done using grid-search on each dimensionality reduction method. This is to ensure good hyperparameters are found for each number of components so that the results do not become skewed by hyperparameters that are not optimal for some parts of the model. If it was chosen to use a fixed value for the hyperparameters, the results could get skewed for a certain number of components. The dimensionality reduction methods used are PCA, LDA, ISOMAP, and kPCA. 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 non-linear methods, five components, instead of 2, were chosen as the lowest amount, as reducing the data to further takes much time, and the gain in insight was believed not to be worth the increased time cost. 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. Also, the `mean_fit_time` will be discussed as the time it takes to fit the model can determine how many components to use.

With the results from running the experiment, a plot shows the models' accuracy with the number of components used. The plot is used to represent when the accuracy starts to drop visually.

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, different thresholds will be used based on the highest value of accuracy for each of the methods. The thresholds will be a 1% loss in accuracy, a 5% loss in accuracy, and a 10% loss in accuracy. 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 5.1.

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. With all 50 components, the model's accuracy is 91.9%, and first really drops below 91% around 43 components.

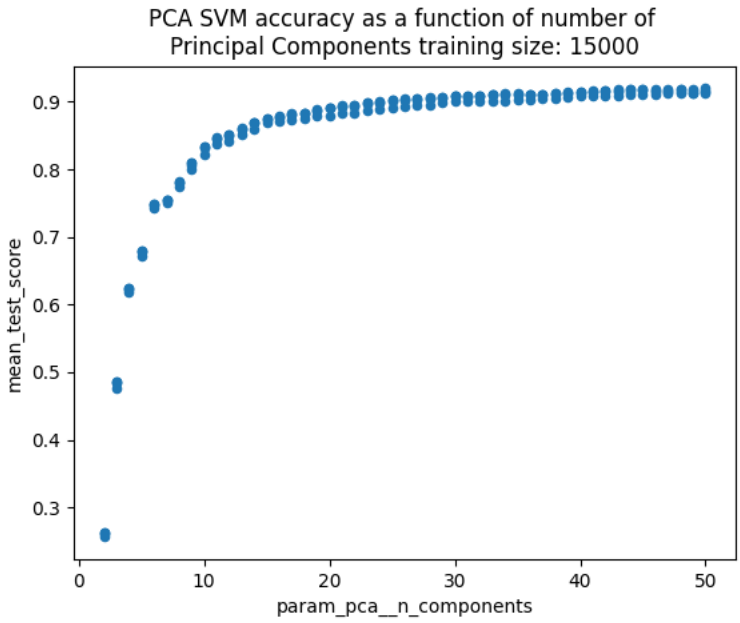


Figure 5.1: Accuracy of the SVM model with PCA as dimensionality reduction method, with the number of components used.

For PCA, the highest accuracy value is 91.9% with 50 components. The thresholds for PCA are: 91.9% - 1% = 90.9, 91.9% - 5% = 87.3%, and 91.9% - 10% = 82.7%. 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 90.9% is with an accuracy of 90.8% at 31 components, which means that the model's accuracy only increases by 1% with the last 19 components, which is close to half the total amount of components. The next threshold of 87.3% accuracy is found at 87.1% with 16 components. By removing only three components, the accuracy dropped from a 1% loss to a 5% loss. The final threshold of 82.7% accuracy is found at 82.1% with ten components. By removing only six components, the accuracy dropped from a 5% loss to a 10% loss.

LDA

LDA is another linear dimensionality reduction method, and the scatter plot representing this method can be seen in Figure 5.2. 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. Similar to PCA, the accuracy of the model increases as the number of components increases. However, around 6-7 components, a noticeable drop in accuracy can be seen. The highest accuracy score for LDA is 87.3% with nine components, and the lowest accuracy score is 50.4% with only two components.

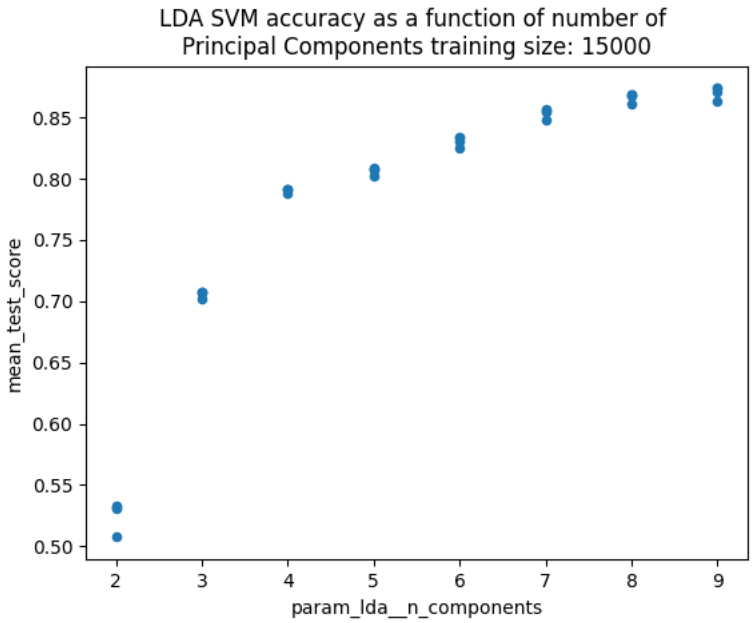


Figure 5.2: Accuracy of the SVM model with LDA as dimensionality reduction method, with the number of components used.

For LDA, the highest accuracy score is at 87.3% with nine components. The thresholds for LDA are: 87.3 - 1% = 86.3%, 87.3 - 5% = 82.7%, and 87.3 - 10% = 78.1%. 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 86.3% is at 86.2% with all nine components. Some

values with only eight components have a higher accuracy score, and some with a lower score than this—showing the impact of hyperparameter tuning. The next threshold of 82.7% accuracy is found at 82.4% with six components. By removing three 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 third of the components can be removed with a 5% accuracy loss. The final threshold of 78.1% accuracy is found at 70.7% 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 non-linear dimensionality reduction method, and the scatter plot representing this method can be seen in Figure 5.3.

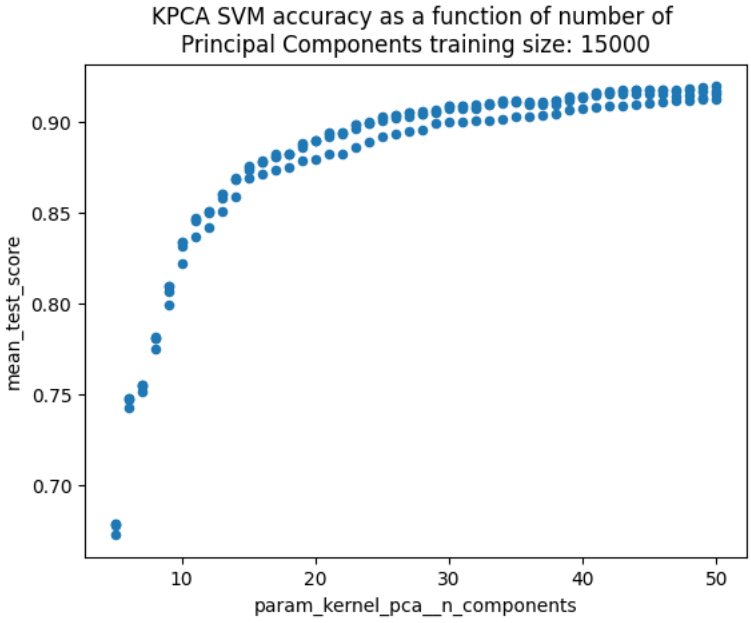


Figure 5.3: Accuracy of the SVM model with kPCA as dimensionality reduction method, with the number of components used.

Figure 5.3 is very similar to the other methods. However, it should be noted that although the accuracy seems to drop drastically as the number of components decreases, by comparing Figure 5.3 and Figure 5.1, it is seen that the accuracy of kPCA overall is higher than PCA. The highest accuracy score for kPCA is 91.9% with 50 components similar to PCA, but the lowest accuracy score is 64.4% with two components, unlike PCA, which has its lowest accuracy score of 25.6%.

kPCA has a top accuracy score of 91.9%, meaning that the thresholds for kPCA are: 91.9 - 1% = 90.9%, 91.9 - 5% = 87.3%, and 91.9 - 10% = 82.7%. 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. The first threshold of 90.9% accuracy is found at 90.8% with 31 components. The next threshold of 87.3% accuracy is found at 87.1% with 16 components. The final threshold of 82.7% accuracy is found at 82.1% with ten components.

ISOMAP

ISOMAP is the final method of the second experiment, which is non-linear, and the scatter plot representing this method can be seen in Figure 5.4.

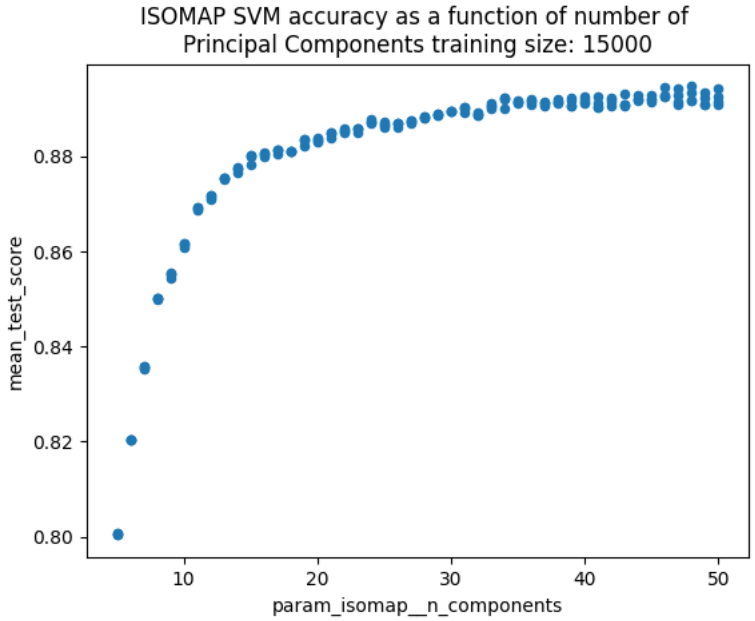


Figure 5.4: Accuracy of the SVM model with ISOMAP as dimensionality reduction method, with the number of components used.

Figure 5.4 is similar to kPCA as the accuracy drops drastically as the number of components decreases, but even the lowest accuracy is still relatively high. The span from best to worst accuracy is much smaller than in linear methods. The highest accuracy score for ISOMAP is 89.4% with 48 components, 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 89.4%, meaning that the thresholds for ISOMAP are: 89.4% - 1% = 88.4%, 89.4% - 5% = 84.4%, and 89.4% - 10% = 79.4%.

The first threshold of 88.4% accuracy is found at 88.3% with 20 components, the next threshold of 84.4% accuracy is found at 83.5% with seven components, and the final threshold of 79.4% accuracy is not found in the data, as the lowest accuracy score is 80% 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 non-linear 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%	**62% (31)**	**100% (9)**
5%	**32% (16)**	**66% (6)**
10%	**20% (10)**	**33% (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, 62% of the total number of components remain before reaching the first threshold of 1%, whereas, for LDA, there is 100%.

From Table 5.12, it can be seen that before reaching the first threshold of losing 1% accuracy, PCA can cut off 38% of its components, while LDA can not afford to lose any components. It shows that for LDA, the number of components and the hyperparameters have a significant impact and that it is not as easy to cut off components as it is for PCA without losing accuracy.

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 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.

Non-linear methods

Unlike the linear methods, the non-linear 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 non-linear methods can be seen in Table 5.13.

Thresholds	KPCA % 50	ISOMAP % 50
1%	**62% (31)**	**40% (20)**
5%	**32% (16)**	**14% (7)**
10%	**20% (10)**	**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 38% of its components, while ISOMAP can cut off 60% 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 consistent than kpca, as the span of accuracy score for ISOMAP is smaller.

Consider adding a table with the best and worst accuracy score for each method.

Comparison of methods

Now that the linear and non-linear 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%	**62% (31)**	**100% (9)**	**62% (31)**	**40% (20)**
5%	**32% (16)**	**66% (6)**	**32% (16)**	**14% (7)**
10%	**20% (10)**	**33% (3)**	**20% (10)**	**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 drastic 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 is targetted towards PCA and kPCA. The goal is to compare the performance of PCA and kPCA, because these methods are similar, with the exception that kPCA implements a kernel. Beause of the similarity of the methods, comparison will be more focused on the impact the different kernels can have on the model, namely what numbers does the machine learning model confuse given the different kernels.

As explained in the Chapter Theory, KPCA can have kernels, which will project the data into a higher dimensional feature space, where a hyperplane can be constructed, and perform PCA on it. KPCA does not require the transformation of the inputs into the feature space with the kernel function, but can use the kernels so as to get the dot product of the pair-wise input points [57].

The kernels are a measure of similarity between the points [45], which means that points that are close to each other have higher similarity score, which is computed with the kernel function. The kernels chosen for the experiment are the Radial Basis Function (RBF) and sigmoid kernels. The sigmoid kernel "computes the sigmoid kernel between two vectors" [45], which outputs a value between -1 and 1 for the two given input vectors. The RBF kernel "computes the radial basis function kernel between two vectors" [45], which outputs a value between 0 and 1.

5.3.1 Rules

This experiment will not use gridsearch. The input of the data samples will be 15000 for both of the methods, because of hardware limitations. The amount of components used for the methods will be the ones that were used in experiment one, namely 49 components. The evaluation will be based on the confusion matrices, and eventually also for the score of the methods. The kernels used will be RBF and sigmoid.

5.3.2 Results

Figure 5.5 shows the results for PCA. Figure 5.6a shows the results for kPCA with sigmoid kernel. Figure 5.6b shows the results for kPCA with rbf kernel.

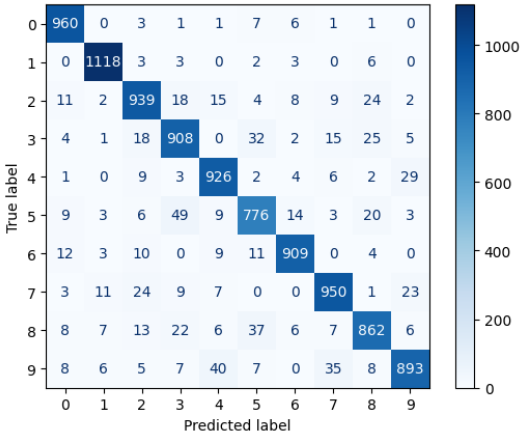


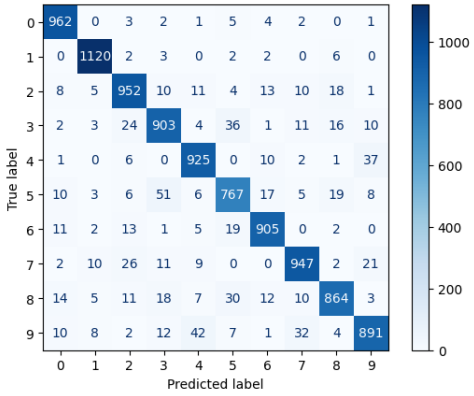
Figure 5.5: Confusion matrix for PCA

5.3.3 Results

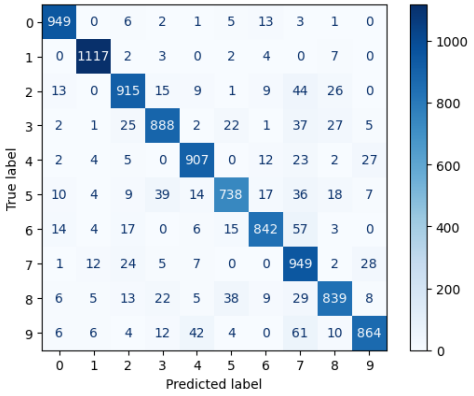
A slight improvement over the number 0 has been seen in Kernel PCA with Sigmoid kernel (KPCA-S) in that it mapped 11 numbers better than PCA, and the same can be said for the number 2.

Both methods acquired the same amount of correct predictions for the numbers 3 and 4.

A slight difference appears in the number 5, where PCA has 780 correct predictions, and KPCA-S got 767, which is worse than PCA. The reason for that is because it confused it with the numbers 3 and 6.



(a) Confusion matrix for kPCA Sigmoid



(b) Confusion matrix for kPCA RBF

Figure 5.6: both kPCA kernels confusion matrices

A slight improvement in KPCA-S is seen at the number 6, where it got five numbers more than PCA. Such a low number is not worth considering, however.

It can be seen that sigmoid confuses the number 7 with the number 9 more often than PCA normally would, which worsens KPCA-S' ability to predict the number 7 correctly.

The notable impressions regarding the differences between PCA and KPCA-S are: KPCA-S improved by about ten numbers at the numbers 0 and 2, became worse at the number 5, became worse at the number 7, and shined the most at the number 8. The most interesting cases are the numbers 0,2, 5, and 8.

The RBF kernel has confused more number 2's than any other method presented in this experiment. As opposed to PCA and KPCA-S, Kernel PCA with RBF kernel (KPCA-R) confused it 33 times more with the number 7 and around ten times more with the number 8. The same pattern can be seen at the number 3, where it was again confused with numbers 7 and 8.

KPCA-R has confused the number 4 mainly with the numbers 7 and 9, and KPCA-S has only confused it with the number 9. Concerning the number 4, KPCA-S is the only method that has confused it the most with the number 9.

KPCA-R mainly confuses the number 5 with the numbers 3 and 7, whereas KPCA-S does it with the numbers 3, 6, and 8, and the same can be said about PCA.

The number 6 reveals a potentially interesting finding: KPCA-R is the only method that managed to confuse the number 6 with the number 7, as opposed to the other methods, which did not confuse it with the number 7.

The only place where KPCA-R outshines KPCA-S is at the number 7, where KPCA-R confuses with two numbers less than KPCA-S. Such a finding, however, is not of major importance because KPCA-R underperforms most of the time.

Summary of the differences between the methods

From the results presented so far about the differences between the kernels it can be seen that all the methods often confuse the number 2 with 8; 3 with 2,5,8 and to a certain degree(PCA and much worse with KPCA-R), 7. The methods also confuse the number 4 with 9; 5 with 3,6, and 8 (KPCA-R also does it with 7). The number 6 gets confused with the number 0,2,5 (KPCA-R also does it with 7). The number 7 gets confused with the number 2,3 and 9. KPCA-R is the only one that did not confuse the number 3 as much as the other methods.

At number 8, the methods have confused 0 (except for KPCA-R),2,3 and 5. KPCA-R confused it again with the number 7, but it is the only one that managed to map the number 0 less than the other methods. Lastly the number 9 got confused with the numbers the most with the numbers 4 and 7. It can be further noted that KPCA-R is the worst at confusing various numbers with the number 7.

From the overview provided regarding the difference in numbers, a percentage would be more preferable, more specifically, a percentage of the errors made in the numbers 0-9. Table 5.15 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 5.15: Error percentage for each number for the methods

Another finding which could be explored is the difference in the number of errors made by the methods for each number. Table 5.15 shows the difference in the number of errors made by the methods for each number.

Yet another finding that could be explored is to research why KPCA-R is the only method that confuses many numbers with the number 7.

5.4 Experiment 4

In this experiment, a machine learning pipeline is used to explore the effects of different sizes of samples on the performance of linear and non-linear dimensionality reduction methods. The MNIST data set is used as the source of data, and a range of sample sizes are selected, starting with 200 and ending with 5000. The pipeline uses SVM as the model and applies four different dimensionality reduction techniques: PCA, LDA, kPCA, and ISOMAP.

The performance of the pipeline is evaluated using three metrics: accuracy, f1-score, and time. The results of the experiment are analyzed to compare the performance of the different dimensionality reduction methods under varying sample sizes and to explore the differences between linear and non-linear approaches.

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 the performance of machine learning models and dimensionality reduction.

5.4.1 Parameters and evaluation

In this experiment the same general hyper parameter tuning as in experiment 1 will be used on datasamples from MNIST of the size 200, 300, 400, 500, 600, 700, 800,

900, 1000, 2000, 3000, 4000, 5000. It will be evaluated based on f1, accuracy and how much time it uses.

5.4.2 Results

In this section, we present the results of the fourth experiment, which compare the performance of different dimensionality reduction methods on a classification task. We show the results for different sample sizes using confusion matrices and tables, as well as scatter plots to visualize the relationship between sample size, accuracy, and time taken. We evaluate the results based on the rules of the experiment, focusing on accuracy, F1 score, time taken, and the size of the sample required to achieve good accuracy. We also discuss the specific values of accuracy obtained from the csv files generated by the experiment. The scatter plots provide a visual representation of the results and make it easy to see when accuracy and time start to improve. generally each method and the base case will show the results of the samples 200, 1000, 5000.

SVM Model

	precision	recall	f1-score	support
0	0.830224	0.908163	0.867446	980.000000
1	0.782424	0.972687	0.867243	1135.000000
2	0.678335	0.694767	0.686453	1032.000000
3	0.736059	0.784158	0.759348	1010.000000
4	0.707377	0.878819	0.783833	982.000000
5	0.716243	0.410314	0.521739	892.000000
6	0.913043	0.635699	0.749538	958.000000
7	0.783178	0.815175	0.798856	1028.000000
8	0.731092	0.714579	0.722741	974.000000
9	0.717842	0.685828	0.701470	1009.000000
accuracy	0.756700	0.756700	0.756700	0.756700
macro avg	0.759582	0.750019	0.745867	10000.000000
weighted avg	0.759485	0.756700	0.749591	10000.000000

Table 5.16: Classification report for baseline_svm_200

The first part of the experiment involved running a machine learning pipeline without applying any dimensionality reduction. The pipeline used a SVM model and was trained on the MNIST data set. The sample size for this part of the experiment was 100.

The results of this experiment are shown in 5.16. The table shows the precision, recall, and f1-score for each of the ten classes in the data set, as well as the overall accuracy of the model. The table also shows the macro average and weighted average scores.

	precision	recall	f1-score	support
0	0.925998	0.970408	0.947683	980.000000
1	0.921667	0.974449	0.947323	1135.000000
2	0.865842	0.881783	0.873740	1032.000000
3	0.877095	0.777228	0.824147	1010.000000
4	0.867005	0.869654	0.868327	982.000000
5	0.785340	0.840807	0.812128	892.000000
6	0.914621	0.894572	0.904485	958.000000
7	0.860927	0.885214	0.872902	1028.000000
8	0.864350	0.791581	0.826367	974.000000
9	0.822178	0.815659	0.818905	1009.000000
accuracy	0.871700	0.871700	0.871700	0.871700
macro avg	0.870502	0.870136	0.869601	10000.000000
weighted avg	0.871760	0.871700	0.871014	10000.000000

Table 5.17: Classification report for baseline_svm_1000

	precision	recall	f1-score	support
0	0.955224	0.979592	0.967254	980.000000
1	0.953072	0.984141	0.968357	1135.000000
2	0.909980	0.901163	0.905550	1032.000000
3	0.890279	0.915842	0.902879	1010.000000
4	0.904854	0.949084	0.926441	982.000000
5	0.901278	0.869955	0.885339	892.000000
6	0.952128	0.934238	0.943098	958.000000
7	0.925123	0.913424	0.919236	1028.000000
8	0.904555	0.856263	0.879747	974.000000
9	0.902414	0.888999	0.895657	1009.000000
accuracy	0.920500	0.920500	0.920500	0.920500
macro avg	0.919891	0.919270	0.919356	10000.000000
weighted avg	0.920338	0.920500	0.920197	10000.000000

Table 5.18: Classification report for baseline_svm_5000

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 5.17. Compared to the results of the first part of the experiment, the accuracy 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 next sample of size 5000 can be seen in 5.18. Comparing 5.18 and 5.18, 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% for the 1000-sample data set to 92% for the 5000-sample data set. Similarly, the precision of the model increased for most of the classes, with the largest increase being seen for class 5, where the precision increased from 78% to 90%.

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 data set.

PCA

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.

	precision	recall	f1-score	support
0	0.838207	0.877551	0.857428	980.000000
1	0.770701	0.959471	0.854788	1135.000000
2	0.675052	0.624031	0.648540	1032.000000
3	0.744227	0.829703	0.784644	1010.000000
4	0.692244	0.845214	0.761119	982.000000
5	0.747492	0.501121	0.600000	892.000000
6	0.931649	0.654489	0.768853	958.000000
7	0.832487	0.797665	0.814704	1028.000000
8	0.737317	0.671458	0.702848	974.000000
9	0.641791	0.724480	0.680633	1009.000000
accuracy	0.754000	0.754000	0.754000	0.754000
macro avg	0.761117	0.748518	0.747356	10000.000000
weighted avg	0.760509	0.754000	0.750028	10000.000000

Table 5.19: Classification report for pca_svm_200

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 5.19 For

example, class 0 has a precision of 0.838207 and a recall of 0.877551, while class 9 has a precision of 0.641791 and a recall of 0.724480. 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 0.857428, while the f1-score for class 9 is 0.680633, 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.

	precision	recall	f1-score	support
0	0.917235	0.961224	0.938714	980.000000
1	0.926271	0.962996	0.944276	1135.000000
2	0.880611	0.893411	0.886965	1032.000000
3	0.876923	0.790099	0.831250	1010.000000
4	0.900826	0.887984	0.894359	982.000000
5	0.789038	0.855381	0.820871	892.000000
6	0.915565	0.939457	0.927357	958.000000
7	0.902119	0.869650	0.885587	1028.000000
8	0.845890	0.760780	0.801081	974.000000
9	0.815414	0.849356	0.832039	1009.000000
accuracy	0.878200	0.878200	0.878200	0.878200
macro avg	0.876989	0.877034	0.876250	10000.000000
weighted avg	0.878426	0.878200	0.877565	10000.000000

Table 5.20: Classification report for pca_svm_1000

In 5.20 we 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 0.917235 and a recall of 0.961224 in the second report, compared to 0.838207 and 0.877551 in the first report. The f1-score for class 0 is also higher in the second report (0.938714) than in the first report (0.857428).

One interesting difference between the two reports is the performance on class 3. In the first report, class 3 has a recall of 0.829703, with an f1-score of 0.784644. In the second report, class 3 a lower recall of 0.790099, resulting in a f1-score of 0.831250. 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 5.21 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. This indicates that the pca_svm_5000 model is more accurate at predicting the correct class for a given input.

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,

	precision	recall	f1-score	support
0	0.940476	0.967347	0.953722	980.000000
1	0.958656	0.980617	0.969512	1135.000000
2	0.890927	0.894380	0.892650	1032.000000
3	0.866218	0.891089	0.878477	1010.000000
4	0.907389	0.937882	0.922384	982.000000
5	0.876417	0.866592	0.871477	892.000000
6	0.936259	0.935282	0.935770	958.000000
7	0.925000	0.899805	0.912229	1028.000000
8	0.897577	0.836756	0.866100	974.000000
9	0.891348	0.878097	0.884673	1009.000000
accuracy	0.910000	0.910000	0.910000	0.910000
macro avg	0.909027	0.908785	0.908699	10000.000000
weighted avg	0.909832	0.910000	0.909711	10000.000000

Table 5.21: Classification report for pca-svm-5000

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.

The results show that the model performed better with larger sample sizes, as evidenced by the higher overall accuracy and f1-scores. In particular, classes 0 and 9 showed the largest differences in performance across the different sample sizes. Overall, the experiment demonstrates the importance of using a sufficient amount of data for training machine learning models.

LDA

This classification report seen in 5.22 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 0.604938 and a recall of 0.949780, while class 5 has a precision of 0.481481 and a recall of 0.204036. 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 0.739116, while the f1-score for class 5 is 0.286614, 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 5.23 shows generally better performance than the classification report for lda_svm_200. For example, class 1 has a precision of 0.745657 and a recall of 0.945374 in the lda_svm_1000 report, compared to a precision of 0.604938 and a recall of 0.949780 in the lda_svm_200 report. Additionally, the f1-score for class 1 is 0.833722 in the lda_svm_1000 report,

	precision	recall	f1-score	support
0	0.801397	0.819388	0.810293	980.000000
1	0.604938	0.949780	0.739116	1135.000000
2	0.514586	0.427326	0.466914	1032.000000
3	0.667053	0.569307	0.614316	1010.000000
4	0.606034	0.695519	0.647700	982.000000
5	0.481481	0.204036	0.286614	892.000000
6	0.722449	0.554280	0.627289	958.000000
7	0.749458	0.672179	0.708718	1028.000000
8	0.590597	0.619097	0.604511	974.000000
9	0.437595	0.569871	0.495050	1009.000000
accuracy	0.616200	0.616200	0.616200	0.616200
macro avg	0.617559	0.608078	0.600052	10000.000000
weighted avg	0.618068	0.616200	0.604480	10000.000000

Table 5.22: Classification report for lda_svm_200

	precision	recall	f1-score	support
0	0.699825	0.818367	0.754468	980.000000
1	0.745657	0.945374	0.833722	1135.000000
2	0.677530	0.382752	0.489164	1032.000000
3	0.625000	0.519802	0.567568	1010.000000
4	0.629767	0.689409	0.658240	982.000000
5	0.496101	0.570628	0.530761	892.000000
6	0.662461	0.657620	0.660031	958.000000
7	0.648130	0.657588	0.652825	1028.000000
8	0.594987	0.463039	0.520785	974.000000
9	0.552239	0.623389	0.585661	1009.000000
accuracy	0.636700	0.636700	0.636700	0.636700
macro avg	0.633170	0.632797	0.625323	10000.000000
weighted avg	0.636121	0.636700	0.628514	10000.000000

Table 5.23: Classification report for lda-svm-1000

compared to 0.739116 in the lda_svm_200 report. This indicates that the model trained on 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 0.920039 in the lda_svm_5000 report, while it is 0.699825 in the lda_svm_1000 report. Similarly, the recall for class 0 is 0.951020 in the lda_svm_5000

	precision	recall	f1-score	support
0	0.920039	0.951020	0.935273	980.000000
1	0.907577	0.960352	0.933219	1135.000000
2	0.871277	0.793605	0.830629	1032.000000
3	0.854692	0.838614	0.846577	1010.000000
4	0.827977	0.892057	0.858824	982.000000
5	0.806306	0.802691	0.804494	892.000000
6	0.881178	0.874739	0.877947	958.000000
7	0.869347	0.841440	0.855166	1028.000000
8	0.806999	0.781314	0.793949	974.000000
9	0.807843	0.816650	0.812223	1009.000000
accuracy	0.856800	0.856800	0.856800	0.856800
macro avg	0.855324	0.855248	0.854830	10000.000000
weighted avg	0.856542	0.856800	0.856202	10000.000000

Table 5.24: Classification report for lda-svm-5000

report, while it is 0.818367 in the lda_svm_1000 report. The f1-score for class 0 is also higher in the lda_svm_5000 report (0.935273) compared to the lda_svm_1000 report (0.754468). 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.

kPCA

In this part of the experiment we expect 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 data set 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 data sets.

The values in 5.25 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 data sets. In general, the model trained on the larger data set (kernel_pca_svm_1000) which can be seen

	precision	recall	f1-score	support
0	0.752715	0.919388	0.827745	980.000000
1	0.687965	0.977093	0.807426	1135.000000
2	0.678387	0.635659	0.656328	1032.000000
3	0.707317	0.746535	0.726397	1010.000000
4	0.690129	0.818737	0.748952	982.000000
5	0.769231	0.426009	0.548341	892.000000
6	0.879245	0.729645	0.797490	958.000000
7	0.875664	0.801556	0.836973	1028.000000
8	0.793492	0.650924	0.715172	974.000000
9	0.705394	0.673935	0.689306	1009.000000
accuracy	0.744100	0.744100	0.744100	0.744100
macro avg	0.753954	0.737948	0.735413	10000.000000
weighted avg	0.752395	0.744100	0.737969	10000.000000

Table 5.25: Classification report for kernel_pca_svm_200

	precision	recall	f1-score	support
0	0.912745	0.950000	0.931000	980.000000
1	0.926236	0.973568	0.949313	1135.000000
2	0.873466	0.896318	0.884744	1032.000000
3	0.884279	0.801980	0.841121	1010.000000
4	0.847573	0.889002	0.867793	982.000000
5	0.790487	0.782511	0.786479	892.000000
6	0.907466	0.900835	0.904138	958.000000
7	0.884837	0.896887	0.890821	1028.000000
8	0.850627	0.765914	0.806051	974.000000
9	0.832847	0.849356	0.841021	1009.000000
accuracy	0.873000	0.873000	0.873000	0.873000
macro avg	0.871056	0.870637	0.870248	10000.000000
weighted avg	0.872556	0.873000	0.872176	10000.000000

Table 5.26: Classification report for kernel_pca_svm_1000

in 5.26 appears to have higher precision, recall, and f1-scores for most classes.

In general, the model trained on the larger data set (kernel_pca_svm_5000) which can be seen in 5.27 appears to have higher precision, recall, and f1-scores for most classes. This suggests that, in this case, increasing the size of the data set 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 0.950385, a recall of 0.978855, and an f1-score of 0.964410, while the model trained on kernel_pca_svm_1000 has a precision of

	precision	recall	f1-score	support
0	0.932485	0.972449	0.952048	980.000000
1	0.950385	0.978855	0.964410	1135.000000
2	0.915187	0.899225	0.907136	1032.000000
3	0.898204	0.891089	0.894632	1010.000000
4	0.901174	0.937882	0.919162	982.000000
5	0.885417	0.857623	0.871298	892.000000
6	0.924742	0.936326	0.930498	958.000000
7	0.922772	0.906615	0.914622	1028.000000
8	0.908207	0.863450	0.885263	974.000000
9	0.892108	0.885035	0.888557	1009.000000
accuracy	0.914100	0.914100	0.914100	0.914100
macro avg	0.913068	0.912855	0.912763	10000.000000
weighted avg	0.913817	0.914100	0.913762	10000.000000

Table 5.27: Classification report for kernel_pca_svm_5000

0.926236, a recall of 0.973568, and an f1-score of 0.949313. This indicates that the larger model is 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 0.901174, a recall of 0.937882, and an f1-score of 0.919162, while the model trained on kernel_pca_svm_1000 has a precision of 0.847573, a recall of 0.889002, and an f1-score of 0.867793. 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 0.924742, a recall of 0.936326, and an f1-score of 0.930498, while the model trained on kernel_pca_svm_1000 has a precision of 0.907466, a recall of 0.900835, and an f1-score of 0.904138. This indicates that the larger model is more effective at correctly identifying and classifying examples from class 6.

ISOMAP

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.

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 0.879502, compared to 0.770581 in isomap_svm_1000 and 0.724705 in isomap_svm_200. This trend is also seen in other evaluation metrics, such as precision and recall.

Additionally, we can see that the model's performance varies across the different

	precision	recall	f1-score	support
0	0.913760	0.962245	0.937376	980.000000
1	0.883046	0.991189	0.933998	1135.000000
2	0.908021	0.822674	0.863244	1032.000000
3	0.867126	0.872277	0.869694	1010.000000
4	0.866667	0.900204	0.883117	982.000000
5	0.855140	0.820628	0.837529	892.000000
6	0.931987	0.915449	0.923644	958.000000
7	0.880642	0.854086	0.867160	1028.000000
8	0.874058	0.833676	0.853389	974.000000
9	0.830000	0.822597	0.826282	1009.000000
accuracy	0.881100	0.881100	0.881100	0.881100
macro avg	0.881045	0.879502	0.879543	10000.000000
weighted avg	0.881141	0.881100	0.880348	10000.000000

Table 5.28: Classification report for isomap_svm_5000

	precision	recall	f1-score	support
0	0.859023	0.932653	0.894325	980.000000
1	0.812364	0.984141	0.890040	1135.000000
2	0.825607	0.724806	0.771930	1032.000000
3	0.795249	0.696040	0.742344	1010.000000
4	0.737938	0.794297	0.765081	982.000000
5	0.668719	0.608744	0.637324	892.000000
6	0.867841	0.822547	0.844587	958.000000
7	0.780198	0.766537	0.773307	1028.000000
8	0.714912	0.669405	0.691410	974.000000
9	0.665112	0.706640	0.685247	1009.000000
accuracy	0.774600	0.774600	0.774600	0.774600
macro avg	0.772696	0.770581	0.769560	10000.000000
weighted avg	0.774111	0.774600	0.772176	10000.000000

Table 5.29: Classification report for isomap_svm_1000

classes in the data set. 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 Comparison and discussion

In terms of performance, 5.7 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.

	precision	recall	f1-score	support
0	0.780198	0.804082	0.791960	980.000000
1	0.620288	0.985903	0.761483	1135.000000
2	0.641854	0.442829	0.524083	1032.000000
3	0.592240	0.800990	0.680976	1010.000000
4	0.655914	0.559063	0.603628	982.000000
5	0.649083	0.317265	0.426205	892.000000
6	0.755760	0.684760	0.718510	958.000000
7	0.572629	0.464008	0.512628	1028.000000
8	0.706505	0.479466	0.571254	974.000000
9	0.455533	0.665015	0.540693	1009.000000
accuracy	0.627600	0.627600	0.627600	0.627600
macro avg	0.643000	0.620338	0.613142	10000.000000
weighted avg	0.641272	0.627600	0.615926	10000.000000

Table 5.30: Classification report for isomap_svm_200

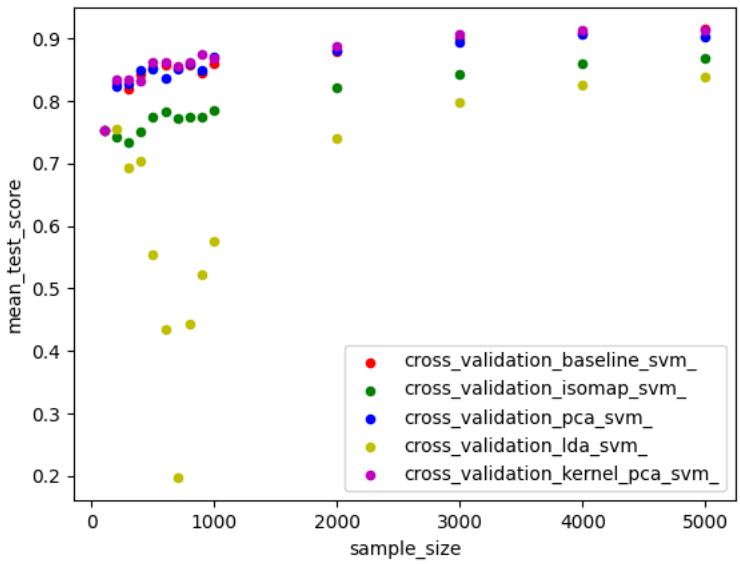


Figure 5.7

In terms of speed, 5.8 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 data set, as they offer both high

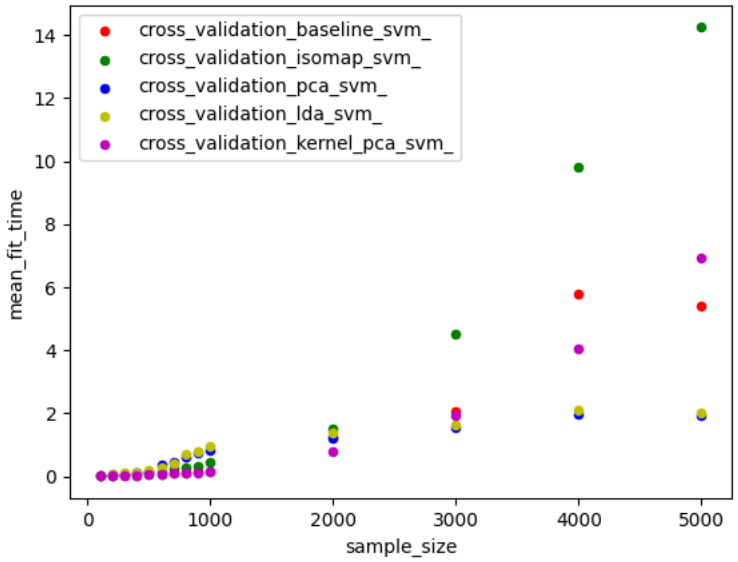


Figure 5.8

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.

Chapter 6

Discussion

This chapter covers the results from chapter chapter 5 and compares the findings to the project's hypothesis. 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 covers topics such as the dataset chosen for the project. It discusses whether MNIST was an adequate dataset to display the project hypothesis and what other options could have been used. The hyperparameters used, their impact on the results, and whether or not the correct ones were chosen. The technical limitations and their impact on the project are also covered.

6.1 Discussion of experiments

This section will discuss the experiments we ran to test our hypothesis. The experiments will be discussed in the following order: Experiment 1, Experiment 2, Experiment 3, and Experiment 4. The hypothesis for the project is that nonlinear methods will not be able to outperform linear methods. Running experiments tested this hypothesis to see if the hypothesis held.

Each experiment has been conducted with a different purpose. Experiment 1 tested the hypothesis by comparing the linear methods' performance to the nonlinear methods' performance. Experiment 2 tested the number of components before a significant drop-off in the accuracy score. Experiment 3 tested the influence kernels have on the accuracy score. Experiment 4 tested the hypothesis on different sample sizes and how the dimensionality reduction methods grew with the number of samples and checked the differences between the dimensionality reduction scaled with the different numbers of samples.

The result from experiment 5.1 shows that LDA is the fastest method, followed by PCA, kPCA, and ISOMAP. This was expected as LDA and PCA are linear methods, and kPCA is a linear method with a nonlinear kernel, making it more computationally expensive. ISOMAP is also a nonlinear method and therefore expected to be slower than both linear methods. By knowing PCA and kPCA is similar, their accuracy scores would be similar. This was expected, as PCA and

kPCA are both linear methods, and the only difference is the kernel. ISOMAP should be close in accuracy score, as it is a nonlinear method, but it could have been more accurate than expected, as it is a more complex method. LDA was the least accurate method, as it is a linear method, which was unexpected as it was expected to be one of the most accurate methods. The drop from PCA and kPCA to ISOMAP and LDA was more significant, as the methods were displayed with their best side. By looking at the overall accuracy score, the results in the hypothesis need to be corrected, as the nonlinear methods did outperform the linear methods, as they had a higher accuracy score. The time it takes to run the methods was also tested, and the result shows that LDA is the fastest method, followed by PCA, kPCA, and last, ISOMAP. This was expected as LDA and PCA are linear methods, and kPCA is a linear method with a nonlinear kernel, making it more computationally expensive. ISOMAP is also a nonlinear method and expects to be slower than both linear methods.

The result from experiment 5.2 shows that the number of components that can remove before the highest accuracy score drops by 1% is 0 for LDA, 31 for PCA and kPCA, and 20 for ISOMAP. ISOMAP and kPCA expect to be very robust as they are both nonlinear methods, taking more components to drop a percentage. LDA is a linear method and expects a quick drop-off in the accuracy score. PCA expected a quick drop-off in the accuracy score, as it is a linear method. However, as PCA and kPCA are similar, it was expected that both could have a similar number of components removed before a significant drop-off in the accuracy score. The result shows that the nonlinear methods are more robust, as they can remove more components before a significant drop-off in the accuracy score. This also does not hold for the hypothesis, as the nonlinear methods outperformed the experiment's linear methods. 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. A machine with more memory could also have allowed running more models on the same machine, which would have been beneficial for comparing the different methods. The machines used for this project did not have the same CPU power. Therefore it could have been beneficial to use a single powerful machine to run all the models. A single powerful machine would have reduced the time to fit the models and allowed more precise comparisons between the different methods.

The result from experiment 5.3 shows that the accuracy score is the highest for PCA, then the sigmoid kernel, followed by the RBF kernel. The hypothesis says that PCA would outperform the other methods, as it is the linear method in the experiment. Therefore the result withholds the hypothesis as the nonlinear methods, the sigmoid and RBF kernels, did not outperform the linear methods in the experiment.

The result from experiment 5.4 shows that each method grows with the number of samples. This was expected, ISOMAP and kPCA are the slowest methods, as they are nonlinear and expected to be slower than the linear methods, because of the added complexity. LDA and PCA are the fastest methods, as they are linear and

expected to be faster than the nonlinear methods. Adding more data increases the accuracy score, as the model has more data to learn from. PCA and kPCA are the most accurate methods. LDA is the least accurate method, as it is a linear method.

In conclusion of the experiments, the hypothesis holds, as the nonlinear methods did not outperform linear ones. However, LDA and PCA are slower with data samples under 2000 than the nonlinear methods. This is surprising, as the nonlinear methods are more computationally expensive, and therefore should be slower.

6.2 The effect of the dataset

Based on the results presented in Table 5.10, and 6.1 the diskussion 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 Canadian Institute For Advanced Research (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 Effect of hyperparameters on the hypothesis

The choice of hyperparameters could contribute to the hypothesis. Taking kPCA as an example, during cross-validation, the scores for the different combinations widely varied, from the worst being 2%, to the best being 91.5%.

In Experiment 5.3 it was shown that the choice of kPCA kernel could severely impact the performance of the confusion matrices that were generated. The experiment used the best configurations for the given kernels found in Experiment 5.1. The CSV file containing information regarding the different combinations can be found on the project's repository (here). The difference in accuracy between KPCA-R and KPCA-S was about 2%. Looking at the Table 5.15, the difference between the kernels was more visible, as the difference in the percentage of numbers that were wrongly predicted between the numbers varied from 0,2% to 5,6%.

The different versions of the kPCA also used slightly different hyperparameters, contributing to the difference in accuracy. The best KPCA-S, for example, used γ=0.001, but the best KPCA-R used γ=0.01, and achieved 89.5% accuracy. If only the kernels were changed, then the γ would make a difference. From the CSV file, KPCA-R with γ=0.01 achieved an accuracy of 56%, and KPCA-S with γ=0.001 achieved 91.2%. Thus, from the experiments, hyperparameters could impact the hypothesis.

The choice of hyperparameters could have been made better. According to sklearn, other kernels could have been used [45]. Furthermore, the number of components could have been expanded on, and used more different values for kPCA's γ value. Other methods also use different hyperparameters, which need to be considered. The results regarding the methods with other hyperparameters are unknown, and therefore it cannot be stated whether they will impact on the hypothesis. In future works, experiments regarding the effect of these hyperparameters could be done in more detail to research the effect of the hyperparameters on the hypothesis.

6.4 Impact of memory limitations

This section will discuss the limitations of memory and the impact of the limitations on the models, with results and other factors in mind. 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.

6.4.1 Impact on the results

As stated throughout the project, a limiting factor when running the non-linear 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 non-linear 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 non-linear 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 5.7 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 non-linear methods.

6.4.2 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 increased memory usage. Since non-linear methods already used much memory, it was only sometimes possible to run the models on multiple cores. An example of how much faster it was with multiple cores was when running ISOMAP, which took about 3+ days to run on a machine with only eight GB of memory, meaning that there was not enough ram to run on multiple cores. Whereas for a machine with 32 GB of memory, it took about one day to run on three cores. This shows the impact of limited memory on time to fit the models. The CPU's power also had an effect, but the general still shows that using multiple cores significantly reduced 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. A machine with more memory could also have allowed running more models on the same machine, which would have been beneficial for comparing the different methods. The machines used for this project did not have the same CPU power. Therefore it could have been beneficial to use a single powerful machine to run all the models. A single powerful machine would have reduced the time to fit the models and allowed more precise comparisons between the different methods.

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. By having multiple machines to run the models, the time impact was shared, had there only been a single machine available that had limited memory, the time impact could have been much more significant.

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:

In this project, we aimed to explore the impact of dimensionality reduction on the performance of a ML for image classification and recognition. We focused on comparing linear and nonlinear dimensionality reduction techniques on MNIST, using a SVM as our model of choice. Our results showed that, for sample sizes greater than 2000, linear dimensionality reduction techniques were generally faster than nonlinear methods, while achieving similar levels of accuracy. This supports our hypothesis "nonlinear dimensionality reduction methods work as well as linear methods on the MNIST dataset". However, our results also indicated that nonlinear methods were more robust, in that they could remove more components before a significant drop in accuracy occurred. Overall, our findings support our hypothesis and provide valuable insights into the relative effectiveness of different dimensionality reduction techniques for image classification tasks.

In conclusion, our project has provided valuable insights into the use of dimensionality reduction techniques in image classification and recognition tasks. We have shown that, while nonlinear methods may be more computationally expensive, they can be equally as effective as linear methods in terms of performance. Our findings have important implications for the field of computer vision and can inform future research in this area.

Acronyms

AI Artificial Intelligence. 3

CIFAR-10 Canadian Institute For Advanced Research. 73

cMDS Classical Multi Dimensional Scaling. 19, 20

FA Factor Analysis. 17, 18, 20

fashion-MNIST Fashion MNIST. 73

FE Feature Engineering. 4, 9, 15, 16, 25, 26

ISOMAP Isometric Feature Mapping. 18–23, 32, 33, 37, 43–47, 51, 53, 54, 57, 66–69, 71–73, 75

kPCA Kernel Principal Component Analysis. 12, 18–23, 32, 33, 37, 42, 44, 45, 47, 50, 51, 53–55, 57, 64, 67, 68, 71–74

KPCA-R Kernel PCA with RBF kernel. 56, 57, 74

KPCA-S Kernel PCA with Sigmoid kernel. 55, 56, 74

LDA Linear Discriminant Analysis. 17, 18, 20–23, 32, 33, 37, 39, 40, 44–47, 49, 50, 52, 54, 57, 62, 64, 67–69, 71–73

MDS Multi Dimensional Scaling. 19

ML Machine Learning. 3–6, 9, 11–14, 16, 25, 27, 77

MNIST Modified National Institute of Standards and Technology. 9, 12, 13, 16, 20, 29, 32, 47, 49, 57, 58, 62, 64, 68, 71, 73, 74, 77, 79

NMF Non-negative Matrix Factorization. 17, 18

NN Neural Network. 1, 12

OvA One-versus-All. 13

OvO One-versus-One. 13

PCA Principal Component Analysis. 17–21, 23, 25, 32–34, 37, 40, 41, 44–50, 52–57, 60, 61, 67, 68, 71–73

RBF Radial Basis Function. 54

sklearn scikit-learn. 16, 31–33, 74

SVM Support Vector Machine. 12, 13, 25, 32–34, 37–46, 57–62, 64, 66, 68, 77

Bibliography

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

[2] *Machine Learning*. IBM. URL: https://www.ibm.com/cloud/learn/machine-learning (visited on 09/27/2022).

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

[4] *Theory-driven Data Analysis and Modeling*. Aalborg University. URL: https://moduler.aau.dk/course/2022-2023/DSNDATB521 (visited on 09/27/2022).

[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] 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.

[12] 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.

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

[14] 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.

[15] Dan Margalit, Joseph Rabinoff, and L Rolen. "Interactive linear algebra". In: (2017).

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

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

[18] 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).

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

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

[21] Uniqtech. *Famous Machine Learning Datasets You Need to Know*. 2019. URL: https://medium.com/data-science-bootcamp/famous-machine-learning-datasets-you-need-to-know-dd031bf74dd.

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

[23] 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.

[24] 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.

[25] 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.

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

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

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

[33] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: (2019).

[34] Kiran Maharana, Surajit Mondal, and Bhushankumar Nemade. "A review: Data pre-processing and data augmentation techniques". In: *Global Transitions Proceedings* (2022). DOI: https://doi.org/10.1016/j.gltp.2022.04.020. URL: https://www.sciencedirect.com/science/article/pii/S2666285X22000565.

[35] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. 2008. ISBN: 9780131687288 013168728X 9780135052679 013505267X.

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

[37] 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).

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

[39] Daniel D Lee and H Sebastian Seung. "Learning the parts of objects by non-negative matrix factorization". In: *Nature* 401.6755 (1999), pp. 788–791.

[40] 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.

[41] 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.

[42] 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.

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

[44] 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.

[45] 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.

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

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

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

[49] 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.

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

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

[52] 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/.

[53] 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.

[54] 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.

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

[56] 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.

[57] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. *Kernel principal component analysis*. 1997, pp. 583–588.