# Discriminative and Generative Classifier

Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics

[[CM0623-2] FOUNDATIONS OF ARTIFICIAL INTELLIGENCE
Academic Year 2022 - 2023

**Student** Nicola Aggio 880008

December 28, 2022

# Contents

**1 Introduction** **3**

**2 Classification** **5**
2.1 Discriminative and Generative models . . . 5

**3 Setup and evaluation** **7**

**4 SVM** **8**
4.1 Theoretical definition and properties . . . 8
4.2 Results . . . 11
4.2.1 SVC - Linear kernel . . . 11
4.2.2 SVC - Polinomial kernel . . . 12
4.2.3 SVC - Radial Basis Function kernel . . . 13

**5 Random Forests** **15**
5.1 Theoretical definition and properties . . . 15
5.2 Results . . . 16

**6 K-NN** **17**
6.1 Theoretical definition and properties . . . 17
6.2 Software implementation . . . 17
6.3 Results . . . 18

**7 Naive Bayes** **20**
7.1 Theoretical definition and properties . . . 20
7.2 Software implementation . . . 21
7.3 Results . . . 22

**8 Aggregate results** **23**

**9 Conclusions** **24**

# List of Figures

1.1 Sample images of the MNIST dataset . . . . . . . . . . . . . . . . . . 3

2.1 Discriminative models . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.2 Generative models . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

4.1 Decision boundaries in a 2-dimensions feature space . . . . . . . . . 8
4.2 Geometry representation of SVM . . . . . . . . . . . . . . . . . . . . 9
4.3 Example of non linearly separable data . . . . . . . . . . . . . . . . . 10
4.4 Geometrical representation of the kernel method . . . . . . . . . . . 11
4.5 Accuracy of the model with different cost parameter . . . . . . . . . 12
4.6 Confusion matrix of SVC with linear kernel . . . . . . . . . . . . . 12
4.7 Accuracy of the model with different cost parameter and gamma measures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
4.8 Confusion matrix of SVC with polinomial kernel . . . . . . . . . . . 13
4.9 Accuracy of the model with different cost parameter and gamma measures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
4.10 Confusion matrix of SVC with RBF kernel . . . . . . . . . . . . . . 14

5.1 Confusion matrix of the Random Forests classifier . . . . . . . . . . 16

6.1 Implementation of K-NN algorithm . . . . . . . . . . . . . . . . . . . 18
6.2 Accuracy of K-NN with different values of k . . . . . . . . . . . . . 19
6.3 Confusion matrix of K-NN classifier . . . . . . . . . . . . . . . . . . 19

7.1 Implementation of Naive Bayes algorithm . . . . . . . . . . . . . . . 21
7.2 Confusion matrix of the Random Forests classifier . . . . . . . . . . 22

# List of Tables

4.1 Results of SVC with linear kernel . . . . . 11
4.2 Results of SVC with polinomial kernel . . . . . 13
4.3 Results of SVC with rbf kernel . . . . . 14

5.1 Results of Random Forests classifier . . . . . 16

6.1 Results of K-NN . . . . . 18

7.1 Results of Nave Bayes classifier . . . . . 22

8.1 Aggregate results . . . . . 23

# Chapter 1

# Introduction

The goal of this assignment is to implement a classifier for the MNIST [1] database, which is composed of 7,000 28x28 pixels gray-scale images of handwritten digits: an example of the images of the dataset is shown in the Picture 1.

Figure 1.1: Sample images of the MNIST dataset

More specifically, the task will be completed using the following models:

- SVM, using **linear**, **polynomial of degree 2** and **rbf** kernels;
- Random forests classifier;
- K-NN classifier;
- Naive Bayes classifier.

While for the first two algorithms we were allowed to use the implementation of any library (in this experiment I considered scikit-learn), for N-NN and Naive Bayes we were asked to provide our personal implementation.
This report is organized as follows: in Chapter 2 I will briefly describe the classification problem and the distinction between discriminative and generative models; in Chapter 3 I will provide some informations about the method I used to evaluate the models and I will discuss some choices I made during the experiment; in Chapter 4, 5 and 6 I will present, respectively, SVM, Random forest, K-NN and Naive

[1] http://yann.lecun.com/exdb/mnist/

Bayes algorithms, analyzing their functioning, their properties and their classification performances. Finally, in Chapter 8 I will reveal the aggregate results and performances of all the models, both in terms of execution time and accuracy, while in Chapter 9 I will conclude the report with some discussions about the obtained results.

# Chapter 2

# Classification

As described in [1], **data classification** is a two-step process which consists of:

1. **learning step**, in which both data and a classification algorithm are used in order to build a classifier;

2. **classification step**, in which the classifier built in the learning step is used to predict class labels for new data.

More specifically, if we denote with $X = (x_1, x_2, .., x_n)$ a tuple of the training set of a dataset and with $y$ its class label, the goal of the learning step is to learn a function $f$ such that $y = f(X)$, i.e. that can predict the associated class label $y$ for a tuple $X$. Once the classifier is built, its **accuracy** is calculated by predicting the tuples of the test set, i.e. tuples that are independent of the training tuples, meaning that they were not used to build the classifier, and typically by measuring the percentage of test tuples that are correctly classified by the model.
Since the goal of a classification algorithm is to learn to correctly classify unseen data from previous given labelled observations, this task belongs to a specific type of learning, the so-called **supervised learning**. Among the supervised learning models, we can distinguish two main categories, which are the **discriminative and generative models**, that are described in the following section.

## 2.1 Discriminative and Generative models

**Discriminative models**, also known as **conditional models**, focus on separating classes by learning the boundaries between the classes of the dataset, and for this reason they model the conditional probabilities and they do not make any assumption. From a mathematical point of view, discriminative models assume some functional form $P(y|X)$, where $y$ represents the class label and $X$ represents a tuple of the dataset, and they try to estimate the parameters of $P(y|X)$ directly from the training data. Picture 2.1 provides a visual representation of the goal of a discriminative model.
The main advantage of this type of models is that they're robust to outliers, while on the other hand one disadvantage is the so-called misclassification problem, that is, wrongly classify a data point. Some examples of discriminative models are logistic regression, Support Vector Machines (SVM), decision trees and Random Forests.

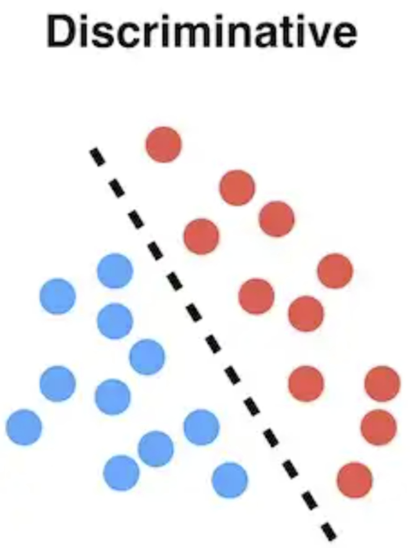


Figure 2.1: Discriminative models

While the focus of discriminative models relies on separating the classes of the dataset, the **generative models**' one relies on the distribution of individual classes of the dataset, and for this reasons they deal with probabilities estimates and likelihood in order to model data points and to distinguish between different class labels in the dataset. From a mathematical point of view, these kind of models estimate the prior $P(Y)$ and the likelihood $P(X|Y)$ using the training data, and they use the Bayes theorem to calculate the posterior probability $P(Y|X)$. A simplified scheme of discriminative models' functioning is represented in Picture 2.1.

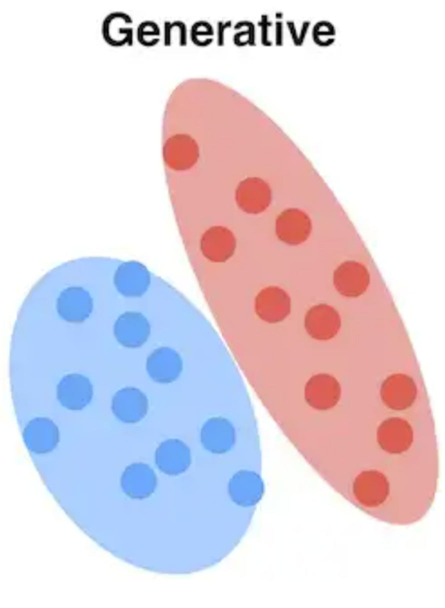


Figure 2.2: Generative models

An important property of generative models is that they're capable of generating new data instances, while their major drawbacks are that they have more impact on outliers than discriminative models and they're more expensive from a computational point of view. Some examples of generative models are Naive Bayes, Bayesian networks and Autoregressive Model.

# Chapter 3

# Setup and evaluation

Before taking into account each of the machine learning models that I used in this study case, I would like to give a brief explanation of the most relevant decisions I made for this experiment. First of all, I decided to delete from the dataset the features containing a unique value (reducing them from 284 to 259) and to rescale them. Then, the dataset was splitted into a train set, validation set and test set according to these percentages: **60%** for **train**, **20%** for **validation** and **20%** for **test**. In order to perform the tuning of the parameters of each model we were asked to use a **10-folds cross-validation approach**, which consists in an iterative technique where the model is evaluated against different test sets in order to avoid overfitting. Scikit-learn's implementation[1] of cross-validation was used for SVM and RandomForests classifiers, while a personal implementaztion was used for K-NN and Naive Bayes classifiers.

[1]https://scikit-learn.org/stable/modules/cross_validation.html

# Chapter 4

# SVM

## 4.1 Theoretical definition and properties

As anticipated in 2, **Support Vector Machines** are examples of discriminative models, i.e. their goal is to find a decision boundary which separates the data points in the best way. In a 2-dimensions feature space the decision boundary is represented by a straight line, while in our study case (784-dimensions feature space), the decision boundary is represented by an hyperplane. An example of decision boundaries in a 2-dimensions feature space is represented in Picture 4.1.

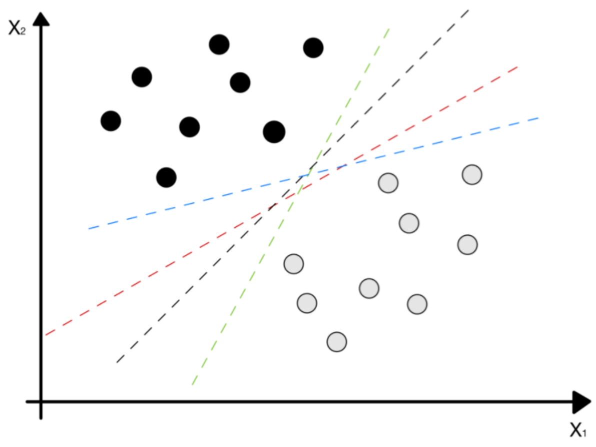


Figure 4.1: Decision boundaries in a 2-dimensions feature space

The data points that are closest to the decision boundary are called **support vectors**, while the distance between the support vectors along the perpendicular direction to the selected hyperplane is called **margin**: the hyperplane chosen by SVM is the one that maximize the margin. By choosing this particular hyperplane, the **misclassification risk** is minimized.
From a mathematical point of view, let's consider a 2-dimension feature space and let's assume labels are such that $y_i \in \{-1, 1\}$. A linear decision boundary $B$ is defined by the equation:

$$w^T x + b = 0$$

, where $w$ weights the features of $x$, so the objects above $B$ are defined by $w^T x + b = k'$, where $k' > 0$, while the objects below $B$ are define by $w^T xb = k''$, where $k'' < 0$. Let $x_s$ and $x_c$ be the support vectors of $B$, we can then rescale $w$ and $b$ such that:

$$w^T x_s + b = 1$$

and

$$w^T x_c + b = -1$$

Let $d_s$ and $d_c$ be the distances between the support vectors and the decision boundary $B$, then by definition:

$$d_s = \frac{|x^T x_s + b|}{||w||_2} = \frac{1}{||w||_2}$$

$$d_c = \frac{|x^T x_c + b|}{||w||_2} = \frac{1}{||w||_2}$$

Then, the margin $d$ is defined as:

$$d = \frac{2}{||w||_2}$$

Then, the goal of SVM is to **maximize** $\frac{2}{||w||_2}$ or, equivalently, **minimize** $|||w||_2$. Finally, we can describe the SVM (binary) classification problem as:

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}||w||_2 \\ \text{s.t.} \quad & y_i(w^T x_i + b) - 1 \geq 0 \end{aligned} \tag{4.1}$$

A scheme of the mathematical interpretation of the functioning of SVMs is represented in Picture 4.1.

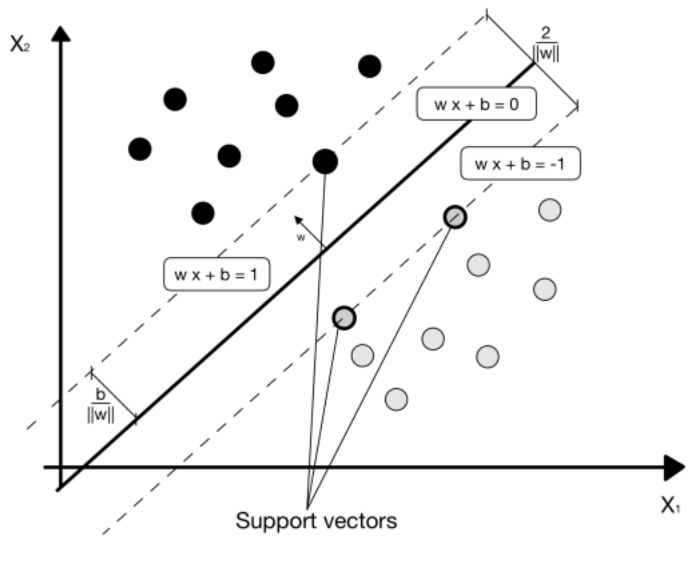


Figure 4.2: Geometry representation of SVM

Since the objective function in 4.2 is quadratic, and the constraints are linear in $w$ and $b$, this is known to be a **convex optimization problem**.
So far I've only considered the case in which the data points are **linearly separable**, i.e. data that can be successfully split by a single linear decision boundary

in the space; however, even if we're dealing with **non linearly separable** (an example is represented in Picture 4.1) data we can still use SVMs with two different approaches.

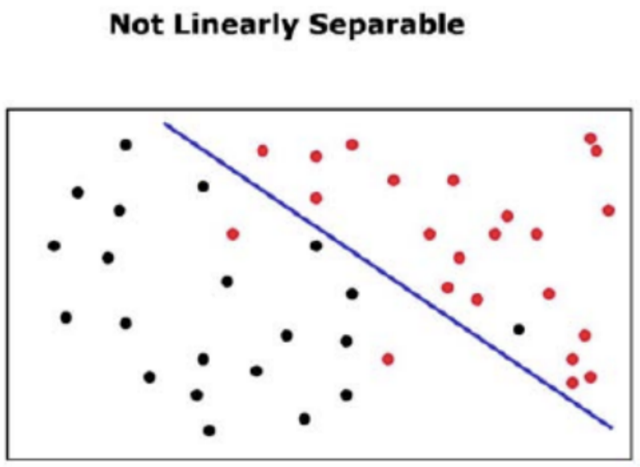


Figure 4.3: Example of non linearly separable data

The first one is characterized by considering the so-called **soft margin**, i.e. by render the condition in 4.2 less strict by adding slack variables to allow for deviation and penalizing the deviation. This approach results in the following formulation:

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}||w||_2 + C\sum_i \xi_i \\ \text{s.t.} \quad & y_i(w \cdot x_i) \geq 1 - \xi_i \quad \forall i \\ & \xi_i \geq 0 \quad \forall i \end{aligned} \tag{4.2}$$

, where $C$ is a balance term controlling how strict the margin must be enforced: by choosing a large value of $C$, the model will work very hard at correctly classifying all the points, whereas a low value of $C$ will allow the model to give up more easily on many of the points so as to achieve a better margin.
However, despite providing a mechanism for weakening the margin, an additional approach is given by transforming the training samples in a higher dimensional space and by training SVM classifier in the higher dimensional space. From a mathematical point of view, suppose that the transformation is performed by a function $\phi(x)$, then, since we know that the SVM classifier only uses dot products of the data (for example, to train the SVM we compute $\phi(x_i) \cdot \phi(x_j)$), from Mercer's Reproducer Theorem we're able to sobstitute any dot products with a positive definite function (called **kernel**) in order to obtain an implicit non-linear mapping to a higher-dimensional space. This method is also known as **kernel method** or **kernel trick**, and it involves a various choice of kernels: in this case study we were asked to focus on the following:

- linear kernel: $K(x_i, x_j) = x_i \cdot x_j$;
- polinomial kernel: $K(x_i, x_j) = (1 + x_i \cdot x_j)^n$;
- radial basis function kernel: $K(x_i, x_j) = \exp(-\frac{1}{2}\frac{||x_i - x_j||^2}{\sigma^2})$.

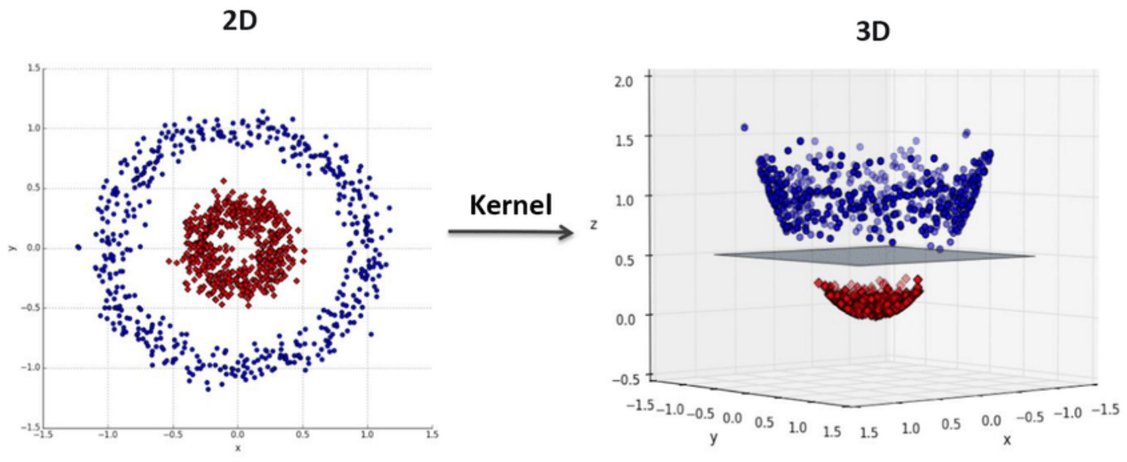


Figure 4.4: Geometrical representation of the kernel method

Picture 4.1 provides a visual representation of the kernel method.
The main **properties** of this classification model rely both on the fact that the SVM problem is a convex optimization problem, a property that guarantees that the model will converge to a single global solution, and on the fact that mapping the data points to a higher-dimensional space do not lead to overfitting, since the accuracy of the model does not depend on the dimensionality of the feature space, but only on the support vectors.

## 4.2 Results

This section shows the results of the Support Vector Classifier on the MNIST dataset using three different kernels: **linear**, **polinomial of degree 2** and **rbf**. For each kernel I will provide:

- the accuracies for each of the tuned hyperparameters,
- the accuracy and the time performances of the tuned model on the test set,
- the confusion matrix, in order to visualize the performances of the algorithm.

### 4.2.1 SVC - Linear kernel

Picure 4.2.1 shows the mean score of accuracy for each value of the parameter $C$ with a 10-folds cross-validation approach: the best score (0.934) is obtained with $C = 0.1$, so I trained and tested an SVM classifier with linear kernel and parameter $C = 0.1$. The results are shown in Table 4.1, while the Picture 4.2.1 represents the confusion matrix of the model.

| Training time(sec) | Test time(sec) | Accuracy |
|---|---|---|
| 158.8 | 103.6 | 93.9% |

Table 4.1: Results of SVC with linear kernel

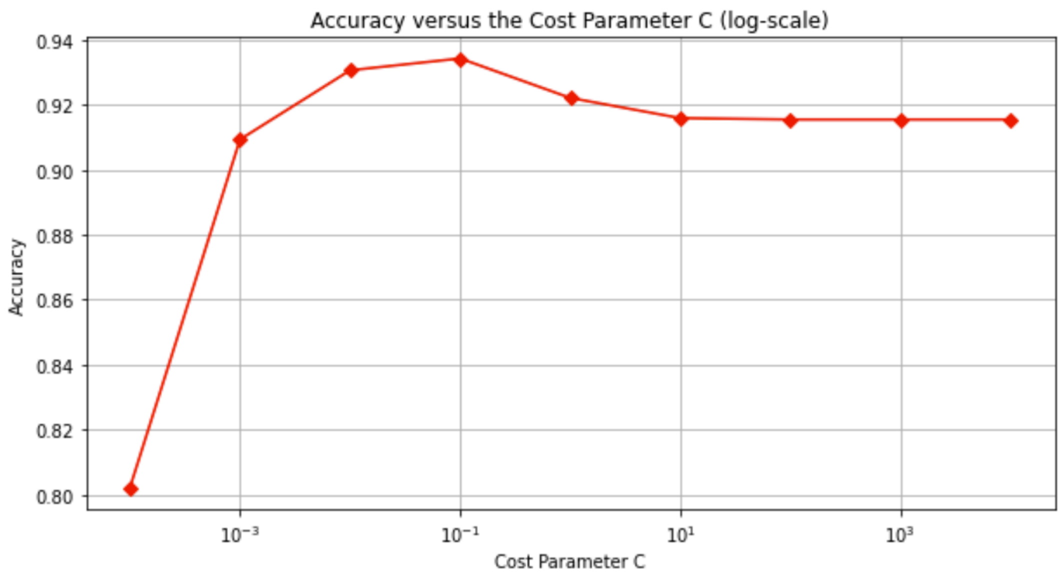
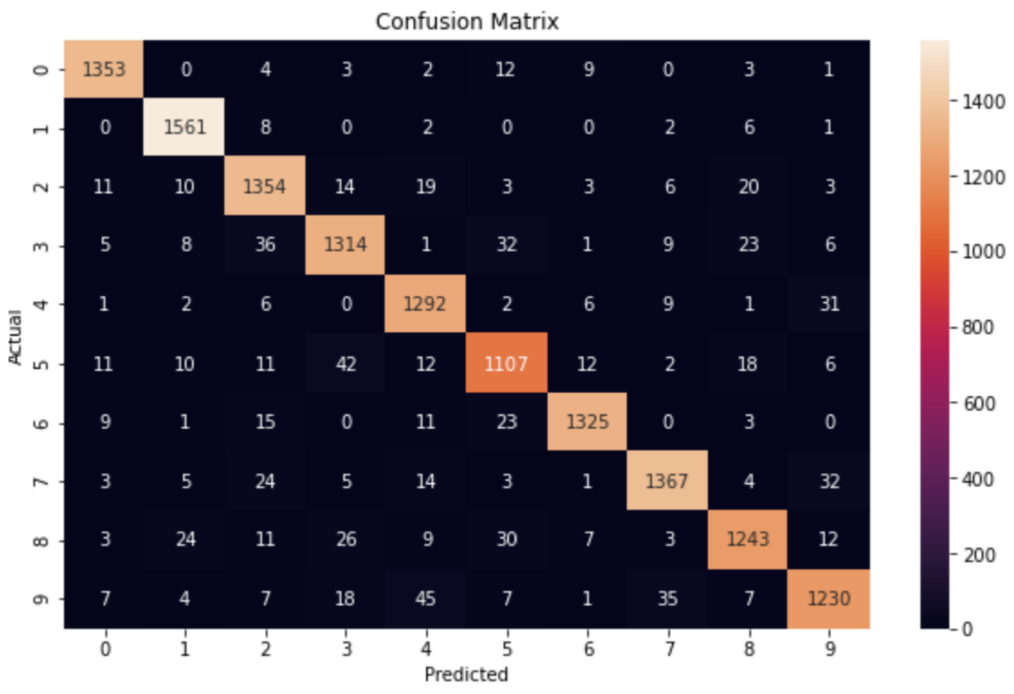


Figure 4.5: Accuracy of the model with different cost parameter



| Actual \ Predicted | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1353 | 0 | 4 | 3 | 2 | 12 | 9 | 0 | 3 | 1 |
| 1 | 0 | 1561 | 8 | 0 | 2 | 0 | 0 | 2 | 6 | 1 |
| 2 | 11 | 10 | 1354 | 14 | 19 | 3 | 3 | 6 | 20 | 3 |
| 3 | 5 | 8 | 36 | 1314 | 1 | 32 | 1 | 9 | 23 | 6 |
| 4 | 1 | 2 | 6 | 0 | 1292 | 2 | 6 | 9 | 1 | 31 |
| 5 | 11 | 10 | 11 | 42 | 12 | 1107 | 12 | 2 | 18 | 6 |
| 6 | 9 | 1 | 15 | 0 | 11 | 23 | 1325 | 0 | 3 | 0 |
| 7 | 3 | 5 | 24 | 5 | 14 | 3 | 1 | 1367 | 4 | 32 |
| 8 | 3 | 24 | 11 | 26 | 9 | 30 | 7 | 3 | 1243 | 12 |
| 9 | 7 | 4 | 7 | 18 | 45 | 7 | 1 | 35 | 7 | 1230 |

Figure 4.6: Confusion matrix of SVC with linear kernel

### 4.2.2 SVC - Polinomial kernel

Picture 4.2.2 shows the mean score of accuracy for each value of the parameter $C$ and the parameter *gamma* with a 10-folds cross-validation approach: the best score (0.968) is obtained with $C = 100$ and $gamma = scale$, so I trained and tested an SVM classifier with polinomial kernel and parameters $C = 0.1$, $gamma = scale$. The results are shown in Table 4.2, while the Picture 4.2.2 represents the confusion matrix of the model.

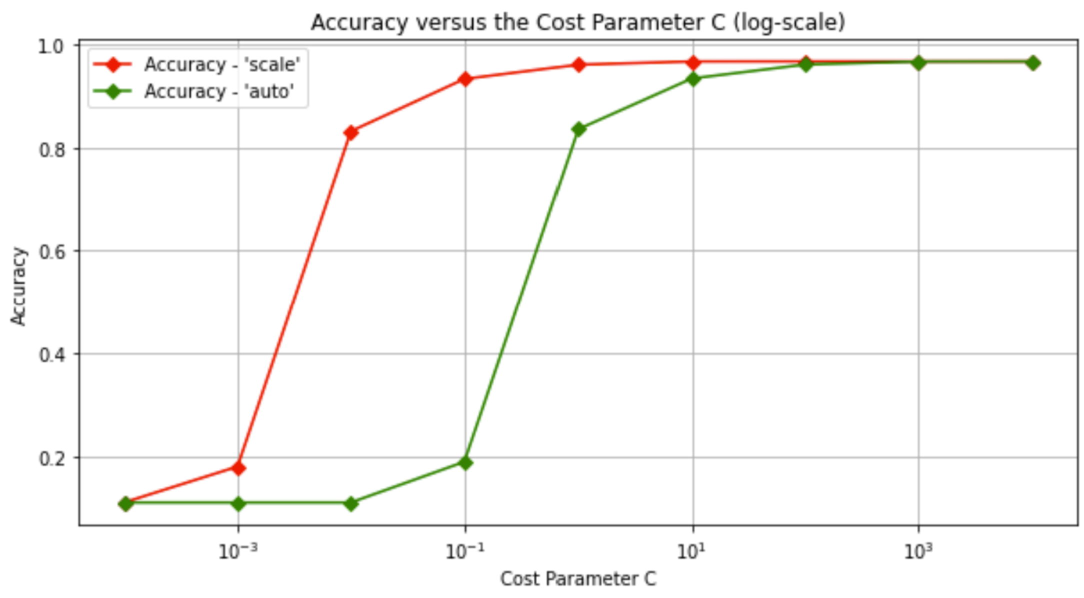
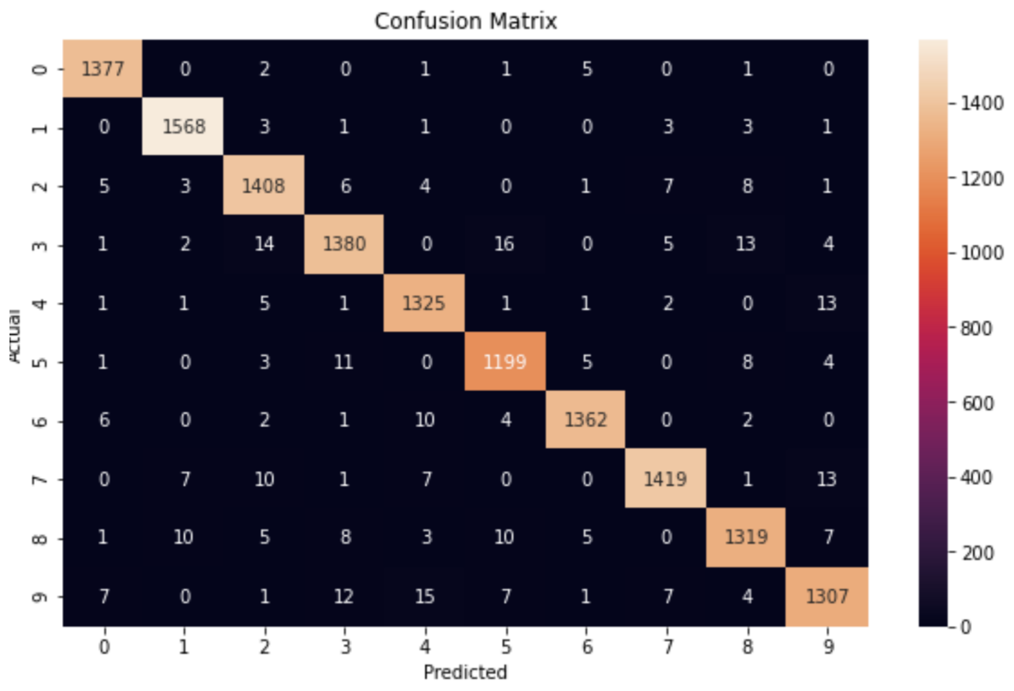


Figure 4.7: Accuracy of the model with different cost parameter and gamma measures

| Training time(sec) | Test time(sec) | Accuracy |
|---|---|---|
| 249.4 | 147.9 | 97.6% |

Table 4.2: Results of SVC with polinomial kernel



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1377 | 0 | 2 | 0 | 1 | 1 | 5 | 0 | 1 | 0 |
| 1 | 0 | 1568 | 3 | 1 | 1 | 0 | 0 | 3 | 3 | 1 |
| 2 | 5 | 3 | 1408 | 6 | 4 | 0 | 1 | 7 | 8 | 1 |
| 3 | 1 | 2 | 14 | 1380 | 0 | 16 | 0 | 5 | 13 | 4 |
| 4 | 1 | 1 | 5 | 1 | 1325 | 1 | 1 | 2 | 0 | 13 |
| 5 | 1 | 0 | 3 | 11 | 0 | 1199 | 5 | 0 | 8 | 4 |
| 6 | 6 | 0 | 2 | 1 | 10 | 4 | 1362 | 0 | 2 | 0 |
| 7 | 0 | 7 | 10 | 1 | 7 | 0 | 0 | 1419 | 1 | 13 |
| 8 | 1 | 10 | 5 | 8 | 3 | 10 | 5 | 0 | 1319 | 7 |
| 9 | 7 | 0 | 1 | 12 | 15 | 7 | 1 | 7 | 4 | 1307 |

Figure 4.8: Confusion matrix of SVC with polinomial kernel

### 4.2.3 SVC - Radial Basis Function kernel

Picure 4.2.3 shows the mean score of accuracy for each value of the parameter $C$ and the parameter $gamma$ with a 10-folds cross-validation approach: the best score (0.968) is obtained with $C = 10$ and $gamma = scale$, so I trained and tested an SVM classifier with rbf kernel and parameters $C = 10$, $gamma = scale$. The results are shown in Table 4.3, while the Picture 4.2.3 represents the confusion matrix of the model.

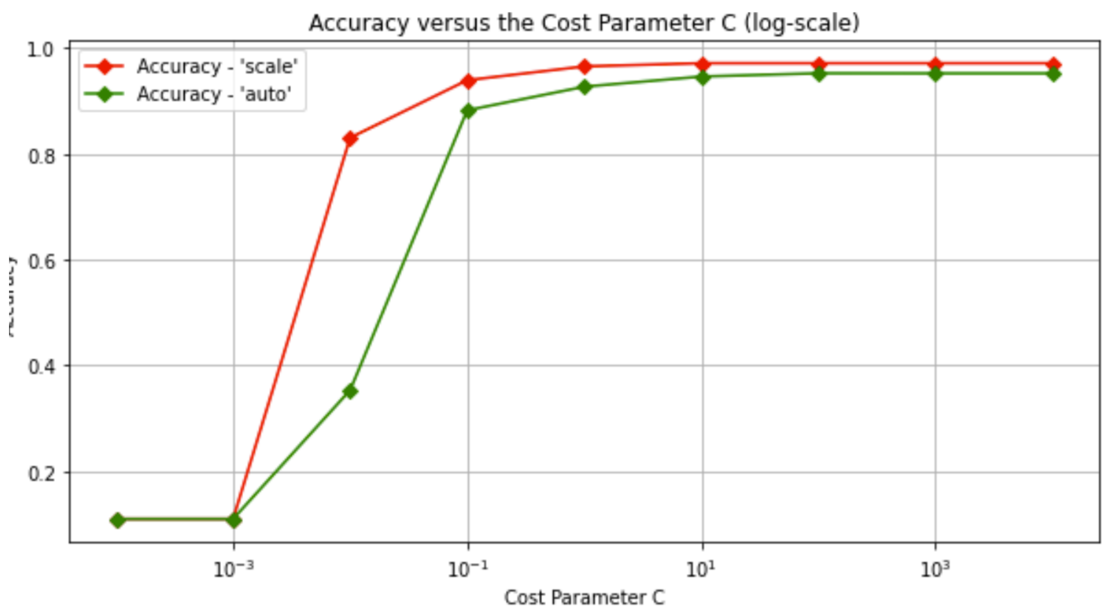
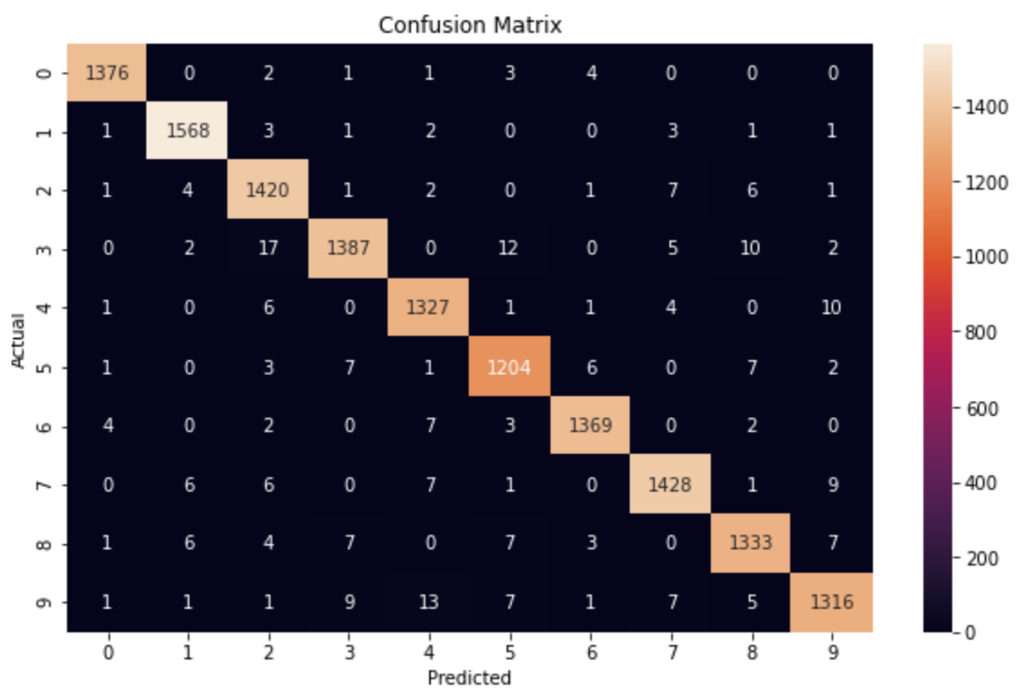


Figure 4.9: Accuracy of the model with different cost parameter and gamma measures

| Training time(sec) | Test time(sec) | Accuracy |
|---|---|---|
| 494.2 | 320.6 | 98.1% |

Table 4.3: Results of SVC with rbf kernel



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1376 | 0 | 2 | 1 | 1 | 3 | 4 | 0 | 0 | 0 |
| 1 | 1 | 1568 | 3 | 1 | 2 | 0 | 0 | 3 | 1 | 1 |
| 2 | 1 | 4 | 1420 | 1 | 2 | 0 | 1 | 7 | 6 | 1 |
| 3 | 0 | 2 | 17 | 1387 | 0 | 12 | 0 | 5 | 10 | 2 |
| 4 | 1 | 0 | 6 | 0 | 1327 | 1 | 1 | 4 | 0 | 10 |
| 5 | 1 | 0 | 3 | 7 | 1 | 1204 | 6 | 0 | 7 | 2 |
| 6 | 4 | 0 | 2 | 0 | 7 | 3 | 1369 | 0 | 2 | 0 |
| 7 | 0 | 6 | 6 | 0 | 7 | 1 | 0 | 1428 | 1 | 9 |
| 8 | 1 | 6 | 4 | 7 | 0 | 7 | 3 | 0 | 1333 | 7 |
| 9 | 1 | 1 | 1 | 9 | 13 | 7 | 1 | 7 | 5 | 1316 |

Figure 4.10: Confusion matrix of SVC with RBF kernel

# Chapter 5

# Random Forests

## 5.1 Theoretical definition and properties

Another example of discriminative models are the **decision trees**. A decision tree is a supervised machine learning model that is used to categorize or make predictions based on available data and previous decisions. Decision trees highly imitate human thinking, and they're defined by the following elements:

- root: this is the base of the decision tree;
- splitting: the process of dividing a node into multiple sub-nodes;
- leaf node: this is a node which is no longer split and provides the classification of the input tuple.

The goal when building a decision tree is to find the splitting of the nodes that leads to the best performances in classifying an input data point: in this sense, several error measures are used, such as Information Gain, Gain Ratio or Gini Index. Among the main advantages of decision trees are that they scale well to large dimensions of the feature space, they can handle data of all types and they're completely nonparametric, whereas its main problems rely on the fact that a single decision tree could face the problem of **overfitting**, which in turns leads to noisy predictions.
A solution to this problem is represented by the usage of the **ensemble methods**, which try to build a combination of independent classifiers in order to obtain both better performances and lower overfitting than the single classifier. Two of the most important ensemble methods are **Bagging** and **Random Forests**. The first one is based on training several quasi-independent models on bootstraps of the dataset and its prediction is given by the aggregation of the predictions of the independent models. On the other hand, Random Forests method tries to increase the independence of the individual models by creating each node of the trees from the splitting of a random subset of the features, and trains fully-grown trees, resulting in a lower overall bias. In this sense, the advantages of using Random Forests ensemble method are that it provides low bias, it is not characterized by overfitting problems, it has very good performances and it is very robust to outliers. On the other hand, the main drawback is that it can become very slow with large datasets and it is overall less interpretable than a single decision tree.

## 5.2 Results

This section shows the results of Random Forests classifier when trained and tested on the MNIST dataset.
First of all, I used the GridSearchCV function of scikit learn[1] in order to find the best values for the following parameters: $n\_estimators$, $criterion$ and $max\_features$. The search was performed with a 10-folds cross-validation approach, and resulted in the following values for the parameters above:

- $n\_estimators = 1000$;
- $criterion =$ gini;
- $max\_features =$ sqrt.

Finally, I trained and tested a Random Forests classifier with the following parameters: $n\_estimators = 1000$, $criterion = gini$ and $max\_features = sqrt$. Table 5.1 represents the obtained results, while the Picture 5.2 represents the confusion matrix of the model.

| Training time(sec) | Test time(sec) | Accuracy |
|---|---|---|
| 72.1 | 2.7 | 96.7% |

Table 5.1: Results of Random Forests classifier

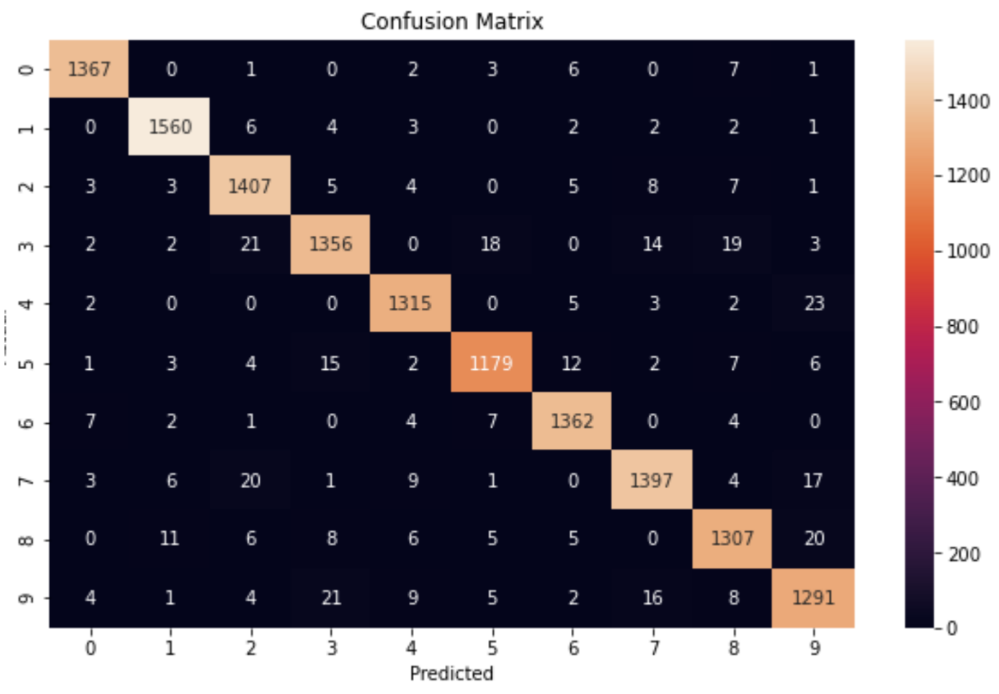


Figure 5.1: Confusion matrix of the Random Forests classifier

[1]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

# Chapter 6

# K-NN

## 6.1 Theoretical definition and properties

**K-Nearest Neighbors** (K-NN) is a supervised learning algorithm which is used for both regression and classification. Its functioning is the following: the K-NN classifier tries to predict the correct class of a test data by considering the most common class of the k-nearest training data points. This algorithm is *non-parametric*, meaning that its predictions are based exclusively on the data points, and it is defined as a *lazy learner*, in the sense that it does not learn any discriminative function from the data, but learns the training data instead.

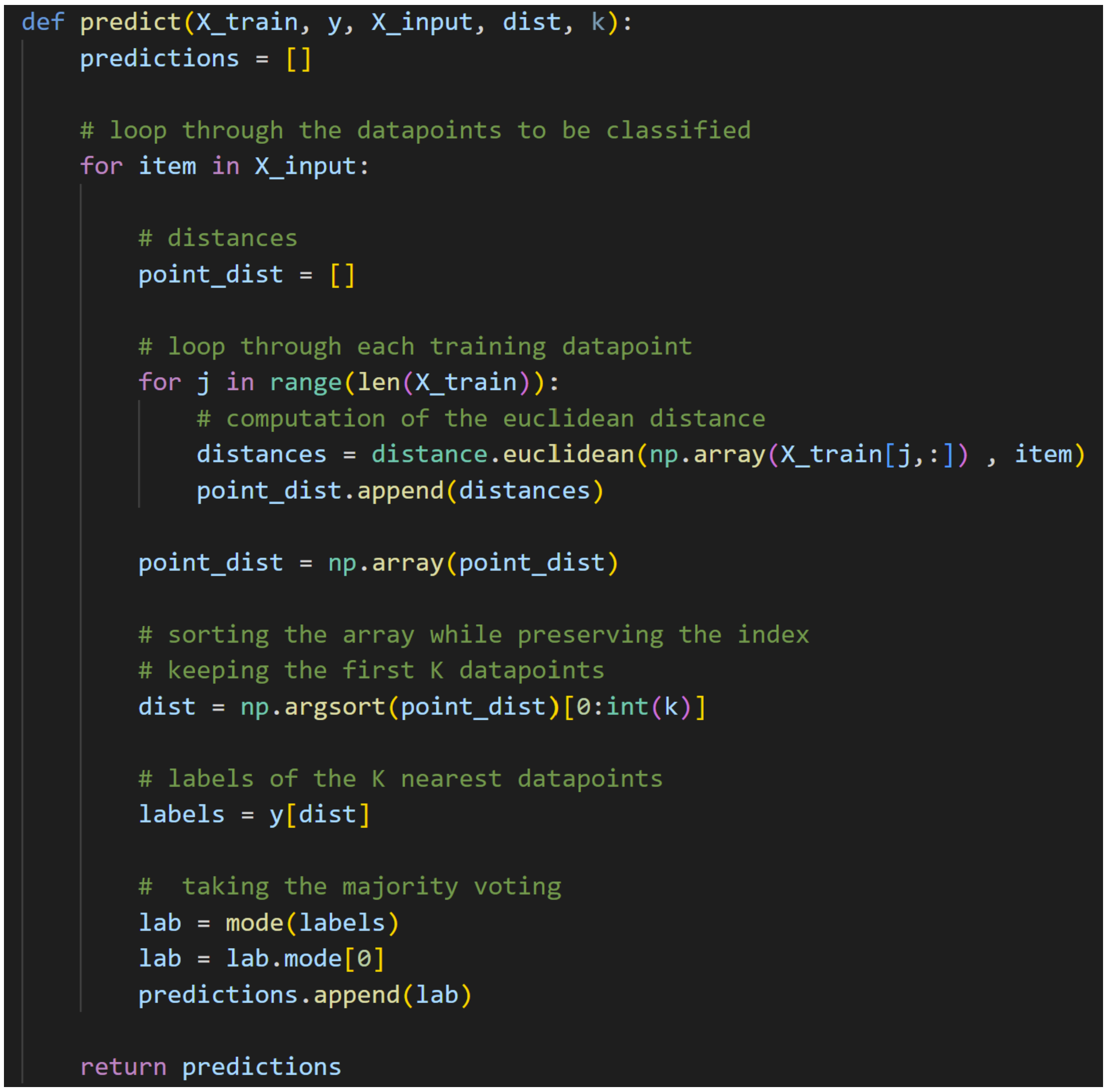
One of the key issue of this algorithm is the choice of **value of k**, i.e. the number of nearest neighbors that are considered for classify the classes. There's no pre-defined statistical method that is used to find the most favourable value of $K$, and for this reason it has to be tuned. In general, by choosing low value of $K$ such as $K = 1$ or $K = 2$, the predictions of the model are more likely to be noisy and subjected to the effects of the outliers. Moreover, the chance of overfitting is also high in such cases. On the other hand, larger values of $K$ lead to smoother decision boundaries and more computationally expensive predictions.

As described before, the main advantages of this model are that it is simple to implement and there's no need to train it, whereas the biggest disadvantage is that it requires high memory storage and its associated computation cost is high.

## 6.2 Software implementation

This section provides the implementation of K-NN algorithm for classifying the classes of the MNIST dataset: the Picture 6.2 represents the implementation of the algorithm.

As showed in the snapshot, in the first place the algorithm iterates through each point that has to be classified; then, it computes the euclidean distances between the current point and all the points of the training set, it chooses the k-closest points and it takes the most common class among them, which represents the classification for the current point.



```python
def predict(X_train, y, X_input, dist, k):
    predictions = []

    # loop through the datapoints to be classified
    for item in X_input:

        # distances
        point_dist = []

        # loop through each training datapoint
        for j in range(len(X_train)):
            # computation of the euclidean distance
            distances = distance.euclidean(np.array(X_train[j,:]) , item)
            point_dist.append(distances)

        point_dist = np.array(point_dist)

        # sorting the array while preserving the index
        # keeping the first K datapoints
        dist = np.argsort(point_dist)[0:int(k)]

        # labels of the K nearest datapoints
        labels = y[dist]

        #  taking the majority voting
        lab = mode(labels)
        lab = lab.mode[0]
        predictions.append(lab)

    return predictions
```

Figure 6.1: Implementation of K-NN algorithm

## 6.3 Results

This section shows the results of K-NN classifier when trained and tested on the MNIST dataset. Picture 6.3 represents the mean score of accuracy for each value of the parameter $k$ with a 10-folds cross-validation approach: the best score (0.954) is obtained with $k = 1$, so I trained and tested a K-NN classifier with parameter $k = 1$. The results are shown in Table 6.1, while the Picture 6.3 represents the confusion matrix of the model. Note that the training time is omitted, since the K-NN algorithm is a lazy learner.

| Test time(sec) | Accuracy |
|---|---|
| 19,447 | 95.8% |

Table 6.1: Results of K-NN

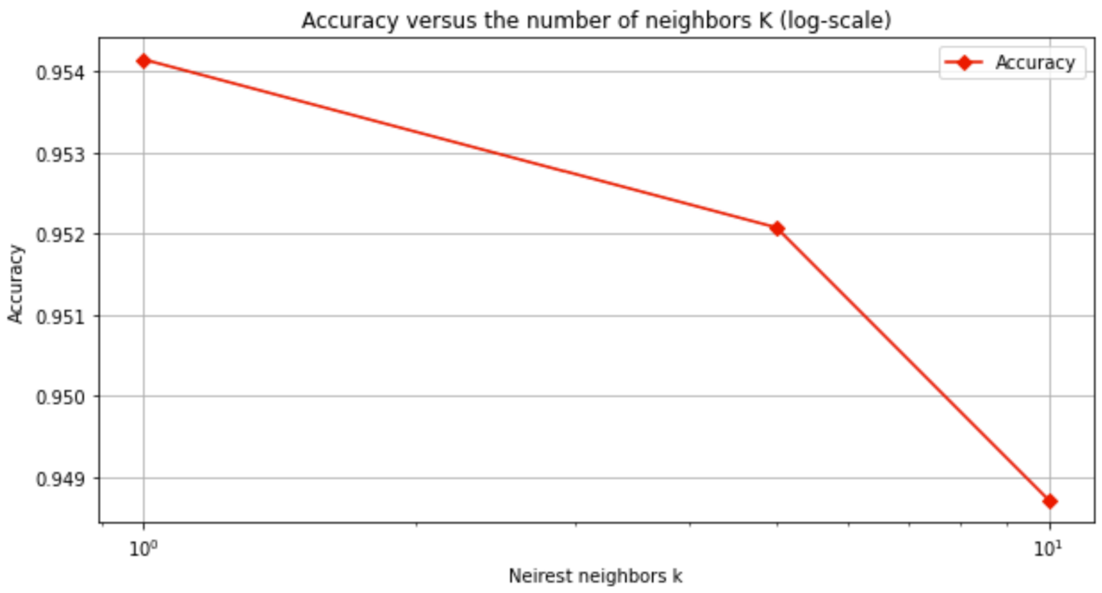
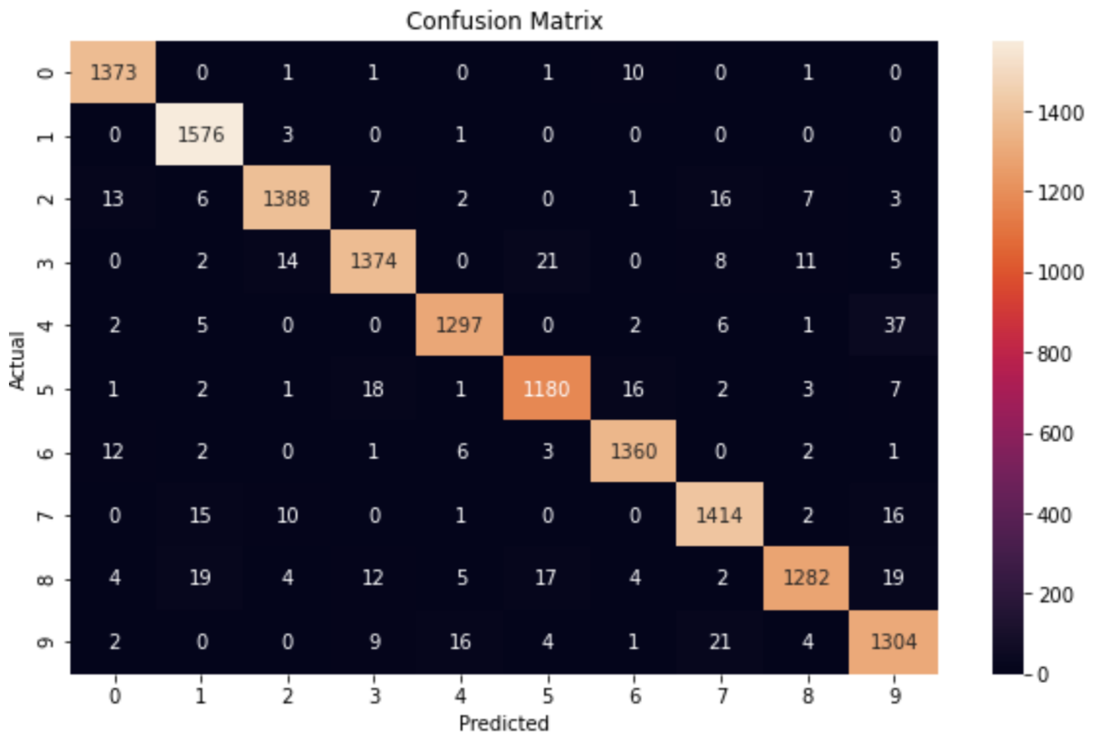


Figure 6.2: Accuracy of K-NN with different values of k



| Actual \ Predicted | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1373 | 0 | 1 | 1 | 0 | 1 | 10 | 0 | 1 | 0 |
| 1 | 0 | 1576 | 3 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 13 | 6 | 1388 | 7 | 2 | 0 | 1 | 16 | 7 | 3 |
| 3 | 0 | 2 | 14 | 1374 | 0 | 21 | 0 | 8 | 11 | 5 |
| 4 | 2 | 5 | 0 | 0 | 1297 | 0 | 2 | 6 | 1 | 37 |
| 5 | 1 | 2 | 1 | 18 | 1 | 1180 | 16 | 2 | 3 | 7 |
| 6 | 12 | 2 | 0 | 1 | 6 | 3 | 1360 | 0 | 2 | 1 |
| 7 | 0 | 15 | 10 | 0 | 1 | 0 | 0 | 1414 | 2 | 16 |
| 8 | 4 | 19 | 4 | 12 | 5 | 17 | 4 | 2 | 1282 | 19 |
| 9 | 2 | 0 | 0 | 9 | 16 | 4 | 1 | 21 | 4 | 1304 |

Figure 6.3: Confusion matrix of K-NN classifier

# Chapter 7

# Naive Bayes

## 7.1 Theoretical definition and properties

The Naive Bayes classifier is a generative machine learning model that is used for classification. Its functioning is based on the **Bayes theorem**, which states:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{7.1}$$

, and from which we can calculate the probability of the **hypothesis** (event $A$) given some **evidence** (event $B$). In the case of a classification task, the evidence corresponds to the feature vectors $X = \{x_1, x_2, .., x_n\}$, while the hypothesis is represented by the class label $y$. In this sense, we can rewrite 7.1 as:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)} \tag{7.2}$$

The fundamental assumption behind the Bayes theorem is that the features are independent, so 7.2 is trasformed into:

$$P(y|x_1, .., x_n) = \frac{P(x_1|y)P(x_2|y)..P(x_n|y)P(y)}{P(x_1)P(x_2)..P(x_n)}$$

However, for each data point of the dataset, the quantity $P(x_1)P(x_2)..P(x_n)$ does not change, and for this reason we can remove it from calculation:

$$P(y|x_1, .., x_n) = P(y)\prod_{i=1}^{n} P(x_i|y)$$

The goal of Naive Bayes classifier is to select the class label $y$ with highest **posterior probability**, i.e. $P(y|x_1, .., x_n)$. Formally, $y$ is chosen as:

$$y = \max_y P(y)\prod_{i=1}^{n} P(x_i|y) \tag{7.3}$$

Moreover, since each $P(x_i|y)$ takes value between 0 and 1, it is possible that their multiplication could lead to very small quantities, which may lead to get some inaccuracies. For this reason we can apply the logarithm and we get the following:

$$y = \max_y \log(P(y)) + \sum_{i=1}^{n} \log P(x_i|y) \tag{7.4}$$

$P(y)$ is called **prior probability**, and it represents the frequency of each class, while each $P(x_i|y)$ is called **class condition probability** and in this study case is modeled with a Beta distribution. The Beta distribution is characterized by the following probability density function:

$$f(x;\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$$

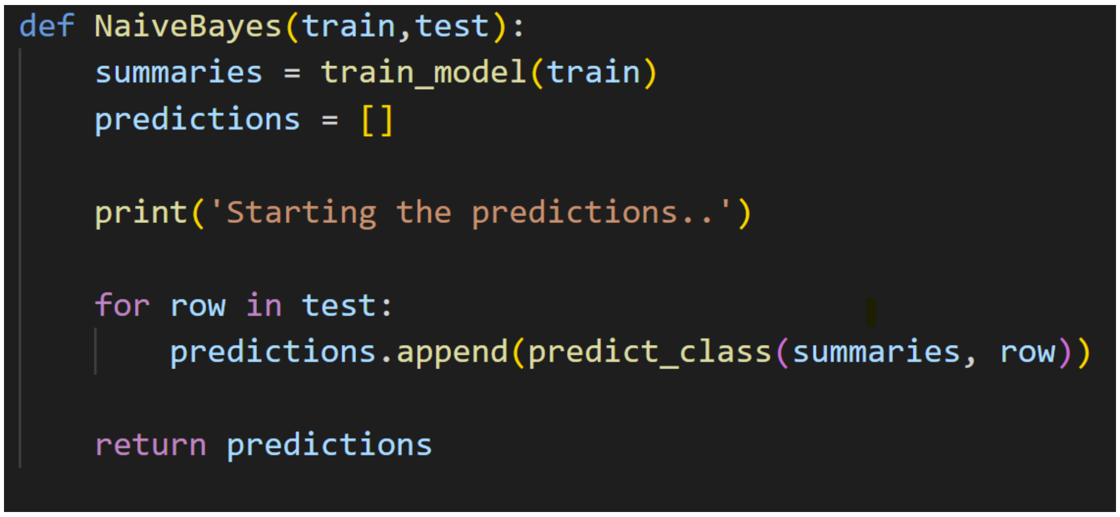
, where $\alpha$ and $\beta$ can be obtained as follows:

- $\alpha = K\mathbf{E}[\mathbf{X}]$
- $\beta = K(1-\mathbf{E}[\mathbf{X}])$

, with $K = \frac{\mathbf{E}[\mathbf{X}](1-\mathbf{E}[\mathbf{X}])}{\mathbf{Var}[\mathbf{X}]} - 1$

## 7.2 Software implementation

This section provides the implementation of Naive Bayes algorithm for classifying the classes of the MNIST dataset: the Picture 7.2 represents the implementation of the algorithm.



```
def NaiveBayes(train,test):
    summaries = train_model(train)
    predictions = []

    print('Starting the predictions..')

    for row in test:
        predictions.append(predict_class(summaries, row))

    return predictions
```

Figure 7.1: Implementation of Naive Bayes algorithm

The **training step** is given by the calculation of the mean, the variance and the prior probability, i.e. the frequency, of each class: this part is covered by the function *train_model*. Then, the **test steps** are the following:

1. calculate the posterior probability, using the prior probability and the class condition probability;
2. choosing the class with highest posterior probability.

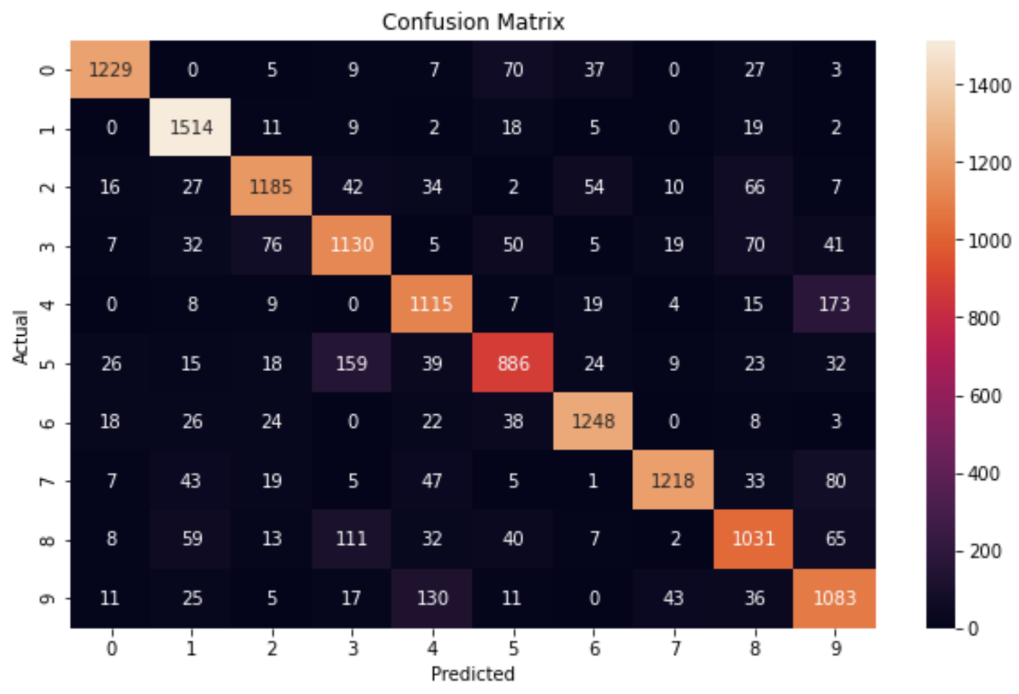
The classification part is covered by the function *predict_class*.

## 7.3 Results

This section shows the results of Naive Bayes classifier when trained and tested on the MNIST dataset. Table 7.1 represents the obtained results, while the Picture 7.3 represents the confusion matrix of the model. Note that the training time is omitted, since it consists on computing the statistics that are used later for prediction.

| Test time(sec) | Accuracy |
|---|---|
| 23,780 | 83.1% |

Table 7.1: Results of Nave Bayes classifier



| Actual \ Predicted | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1229 | 0 | 5 | 9 | 7 | 70 | 37 | 0 | 27 | 3 |
| 1 | 0 | 1514 | 11 | 9 | 2 | 18 | 5 | 0 | 19 | 2 |
| 2 | 16 | 27 | 1185 | 42 | 34 | 2 | 54 | 10 | 66 | 7 |
| 3 | 7 | 32 | 76 | 1130 | 5 | 50 | 5 | 19 | 70 | 41 |
| 4 | 0 | 8 | 9 | 0 | 1115 | 7 | 19 | 4 | 15 | 173 |
| 5 | 26 | 15 | 18 | 159 | 39 | 886 | 24 | 9 | 23 | 32 |
| 6 | 18 | 26 | 24 | 0 | 22 | 38 | 1248 | 0 | 8 | 3 |
| 7 | 7 | 43 | 19 | 5 | 47 | 5 | 1 | 1218 | 33 | 80 |
| 8 | 8 | 59 | 13 | 111 | 32 | 40 | 7 | 2 | 1031 | 65 |
| 9 | 11 | 25 | 5 | 17 | 130 | 11 | 0 | 43 | 36 | 1083 |

Figure 7.2: Confusion matrix of the Random Forests classifier

# Chapter 8

# Aggregate results

This Chapter provides an overview of the results of all the classifier I took into account in this study case. Table 8.1 contains the aggregate results on the MNIST dataset.

| Model | Training time(sec) | Test time(sec) | Accuracy |
|---|---|---|---|
| SVM (linear kernel) | 158.8 | 103.6 | 93.9% |
| SVM (polinomial kernel) | 249.4 | 147.9 | 97.6% |
| SVM (rbf kernel) | 494.2 | 320.6 | 98.1% |
| Random Forests | 72.1 | 2.7 | 96.7% |
| K-NN | / | 19,447 | 95.8% |
| Naive Bayes | / | 23,780 | 83.1% |

Table 8.1: Aggregate results

From this Table we can extract some informations in order to compare the six models according to the three metrics:

- **training time**: exlcuding K-NN (lazy learner) and Naive Bayes (its training time is negligible), it's pretty clear that SVM models spend a lot of time during the training phase, while Random Forests classifier is much faster. This is mainly due to the fact that the computation of SVMs involve matrices and non-linear transformations, while Random Forests classifier is characterized by lighter operations.

- **test time**: this metric is dramatically dominated by K-NN and by the Naive Bayes classifiers, and this result is due to the implementation and the functioning of the algorithms. For example, the prediction of K-NN for a new point is calculated by scanning all the data of the training set and by taking the majority vote of the k-nearest points. This results in a pretty heavy process, from a temporal point of view;

- **accuracy**: the best results are obtained by SVM classifier (expecially with rbf kernel) and by Random Forests. We recall that K-NN classifier suffers from the so-called curse of dimensionality, i.e. it has great performances in low-dimensions feature space, whereas its accuracy decreases as the dimensionality increases. On the other hand, Naive Bayes's implementation relies on the fundamental assumption that the features are independent, which does not correspond to what happens in real dataset. Despite this strong assumption, its performances are pretty satisfying.

# Chapter 9

# Conclusions

In this report I analyzed four different machine learning algorithms for the classification problem. The best results were obtained with two discriminative models: the first one is SVM, which takes advantage of the kernel trick in order to classify non-separable data by performing non-linear transformation ; the second one is Random Forests, which instead is based on a specific ensemble method that lead to great performances and avoid the classifier to overfit the training data. On the other hand, the remaining models obtained worse (but still very encouraging) accuracies: K-NN results to be a very simple classifier, but suffers from the curse of dimensionality, so it is not suitable for large dataset, while Naive Bayes is a generative model whose functioning is based on a very strong assumption, which may not be satisfied in the data we deal with. Moreover, they are both characterized by having extremely high computational requirements from a temporal point of view, and for this reason they cannot be considered an optimal model choice for the MNIST classification.
Concluding, the performances of SVM and Random Forests classifiers triumph over the complexity of their mathematical foundations or their functioning, making them the best handwritten digit classifiers for the MNIST dataset and confirming the performances displayed in the benchmark dashboard[1] of Zalando research.

[1]http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/

# Bibliography

[1] Jiawei Han, Jian Pei, and Hanghang Tong. *Data mining: concepts and techniques*. Morgan kaufmann, 2022.