Training of different models for handwritten digits classification.

Project report

Davide Pietrasanta 844824
Fabio D'Elia 829937

Machine Learning

Department of Computer Science DISCo
Università Milano-Bicocca
Professor: Prof. Dott.ssa E. Fersini
Tutor: Dott.sa A. Saibene
January, 2021

Contents

Chapter 1

Introduction

This paper aims to show the use of machine learning models for classification tasks. We want to classify handwritten digits, from MNIST dataset, using *Naive Bayes* e *Neural Network* models.
The paper has three parts:

- Explorative analysis of the data.
- Application of models.
- Evaluation of models.

This project is the implementation of an assigment, you can find all the code and much more here.

Chapter 2

Data Analysis

2.1 Dataset description

The MNIST database of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on pre-processing and formatting.

The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. NIST originally designated SD-3 as their training set and SD-1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-school students. Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples. Therefore it was necessary to build a new database by mixing NIST's datasets.

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-1. The 60,000 pattern training set

contained examples from approximately 250 writers. We made sure that the sets of writers of the training set and test set were disjoint [LeCun, 1998].

Training set and Test set are already formed so we use training set to operate a 10-fold cross validation and we use test set as a holdout dataset.

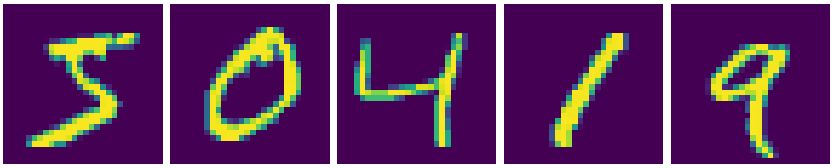

Figure 2.1: Example of images from MNIST dataset.

A single observation in the dataset is a flatten of the image. After we've talked with an expert of the sector we can say that this will not effect excessively the accuracy of our models, since these observation are relatively simple images. Nevertheless we know that our models are not quite optimal and don't use spacial information.
A single observation is the label followed by the values of each pixel, row by row.

2.2 Data distribution

There are several ways to test a uniform distribution of the dataset, we've decided to use a chi-squared test.
For the training set we can conclude that the observed proportions are not significantly different from an uniform distribution, with a p-value of 0.9999972.
For the test set we can conclude that the observed proportions are not significantly different from an uniform distribution, with a p-value of 0.9999935.
This could be useful for a *Naive Bayes* model and would simplify the calculation.

2.3 Digits analysis

First of all we investigate the correlation between each pixel of every images.
As expected there is a lot of positive correlation between near pixels. This local correlation can be exploited by a *Convolutional Neural Network* (CNN).

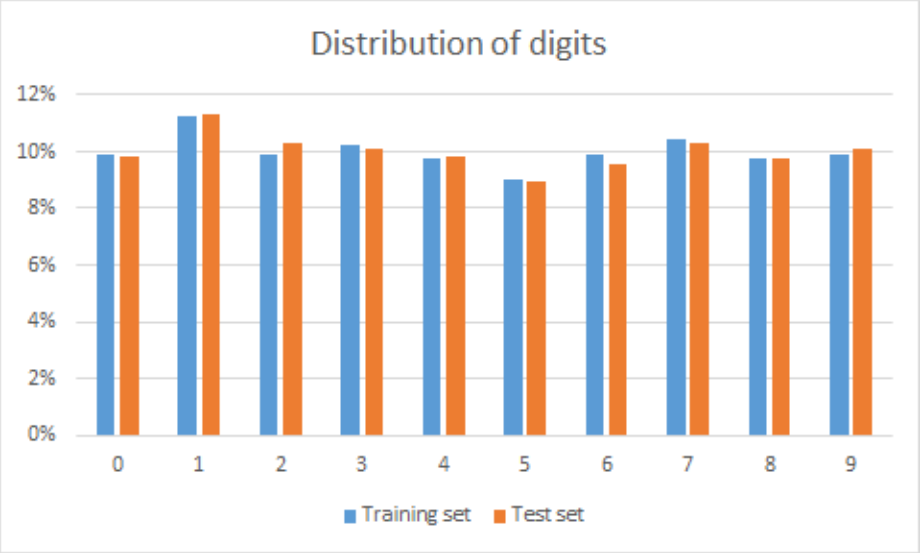

Figure 2.2: Graph of labels distribution in training set and test set.

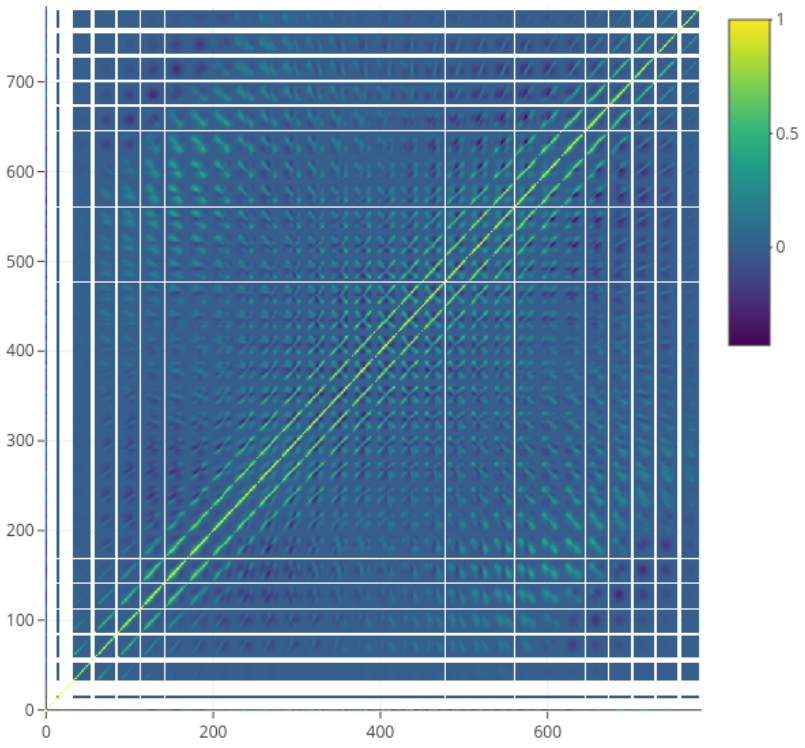

Figure 2.3: Correlation matrix between each pixel.

Instead of training a CNN, we've decided to proceed with a normal multi-layer perceptron model, even if it can lead to a low accuracy.
We have also studied the distribution of gray-scale color inside each digit.

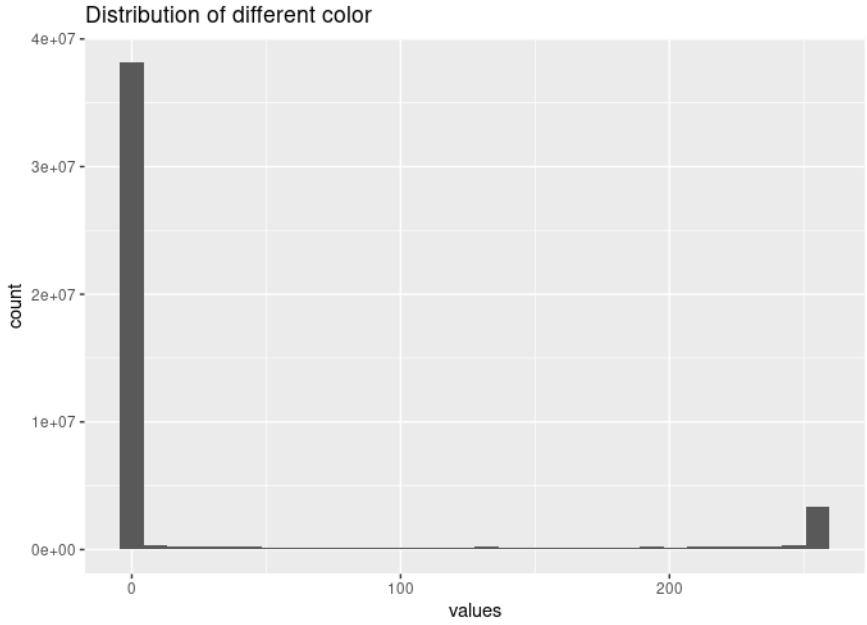

Figure 2.4: Graph of color distribution in training set.

Looking closely, you can see that pixels are mostly white, with a low percentage of black pixels. Therefore you could replace each pixel with a binary encoding, with a very little loss of information (probably null).
Even so, we decided to leave every sample as it is.
Moreover, we are interested in how much variability there is within each digit label. We decide to plot the mean color for each pixel of each label.

You can see that digits like 0s and 1s can be distinguished easily. 2s present some sort of recurrent artifacts compromising the classification.
For these reasons, we think that a well tuned Neural Network and a Naive Bayes can handle the variability of digits.

In order to understand better the way digits differ from each other we study the Euclidean distance of each digit from his centroid. Using a box-plot, we plot the variation of distances from each centroid.

As you can see, 1s have the lowest distance from their centroid and also have a low variation from every sample. This is due to the fact that a 1 is just a simple vertical straight line, so there aren't many ways to draw it. Otherwise, we can see that 0s and 8s are the most distant digits from their centroids. We assume that probably this is due to the fact that circles or wavy shapes can be really different from each other. You can also see

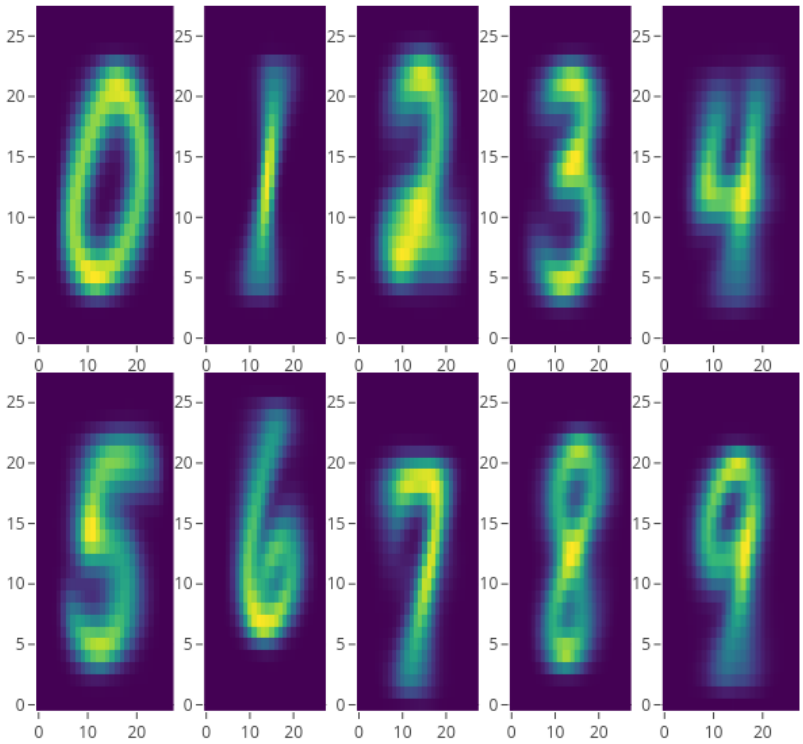

Figure 2.5: Average value of each pixel.

that there are a lot of outliers, so we decide to plot 5 of those for each class.

Surprisingly they are not that far from a well written digit, despite of those digits that look like doodles.

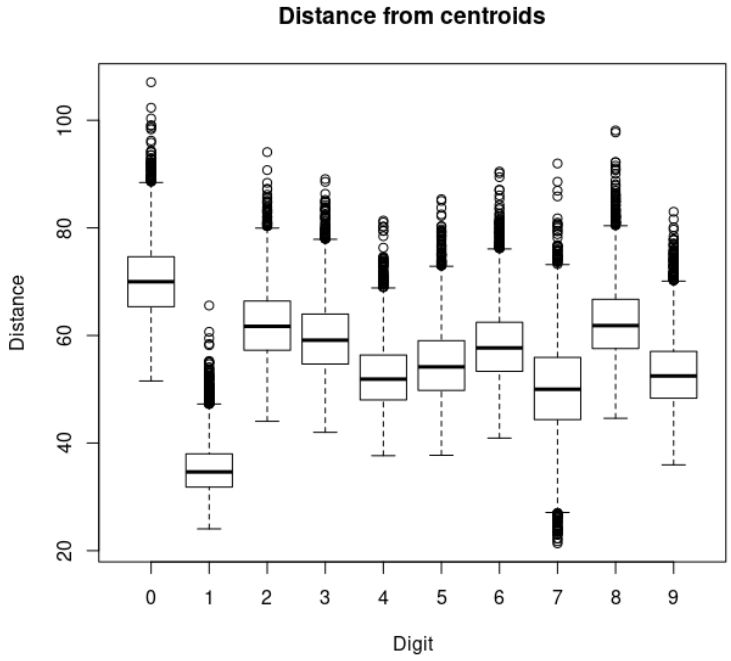

Figure 2.6: Boxplot of distances from all centroids.

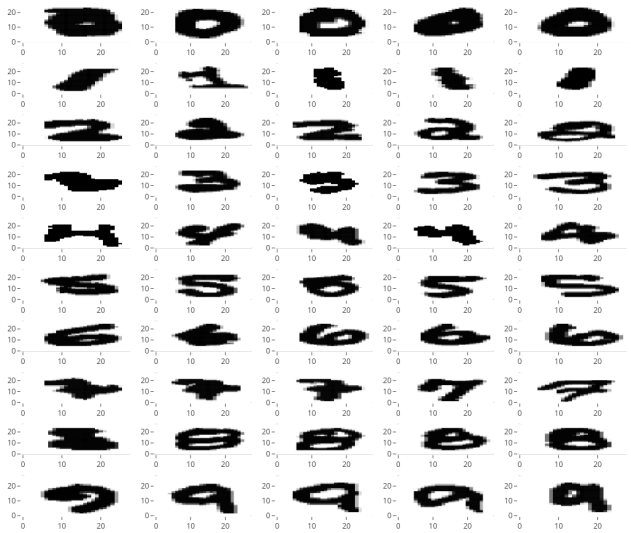

Figure 2.7: Five outliers for each digit.

2.4 Principal Components Analysis

In order to reduce the number of covariates inside the dataset, we use PCA.

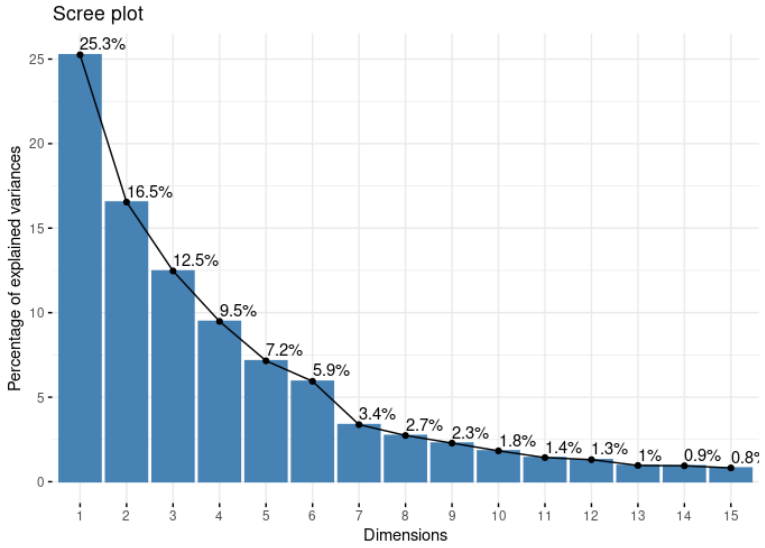

Figure 2.8: Scree plot.

With the scree plot (cropped at 15 PCs), you can see that the majority of variance is explained with the first 5 Principal Components. We've measured eigenvalues in order to decide how many components to keep but, as expected, their values were all less than 1. In order to decide how many PCs to keep, we use the quantity of cumulative explained variance.
We've tried different numbers of PCs. We've decided to use 20 PCs, explaining 95% of the variance, which is very good.
So we project our 784-dimension data to a reduced 20-dimension one.

Chapter 3

Naive Bayes

The first model is a *Naive Bayes* classifier, a supervised machine learning algorithm. This algorithm uses Bayes Theorem to generate a model.

We think that, since the images are centered and relatively simple we can say that, for example, if the center is not black the probability of the digit being 0 (zero) should be low. Following the same logic, if the center is white the probability of the digit being 1 (one) should be higher than the probability of the digit being 0 (zero).
Thinking in that way *Naive Bayes* model could work well also considering the fact that *Naive Bayes* assumes that all input variables are independent. If that assumption is not correct, then it could impact the accuracy of the *Naive Bayes* classifier.

To implement a *Naive Bayes* model in R we used the library *caret*, using the method *nb*.

3.1 Results

We've made a 10-fold cross validation on training set using the *Naive Bayes* model.

We've calculated:

- **Accuracy:** 0.8520
- **Precision:** 0.8556952
- **Recall:** 0.8546793
- **F-measure:** 0.8546174
- **AUC:** 0.920278

- **Accuracy of test:** 0.8621

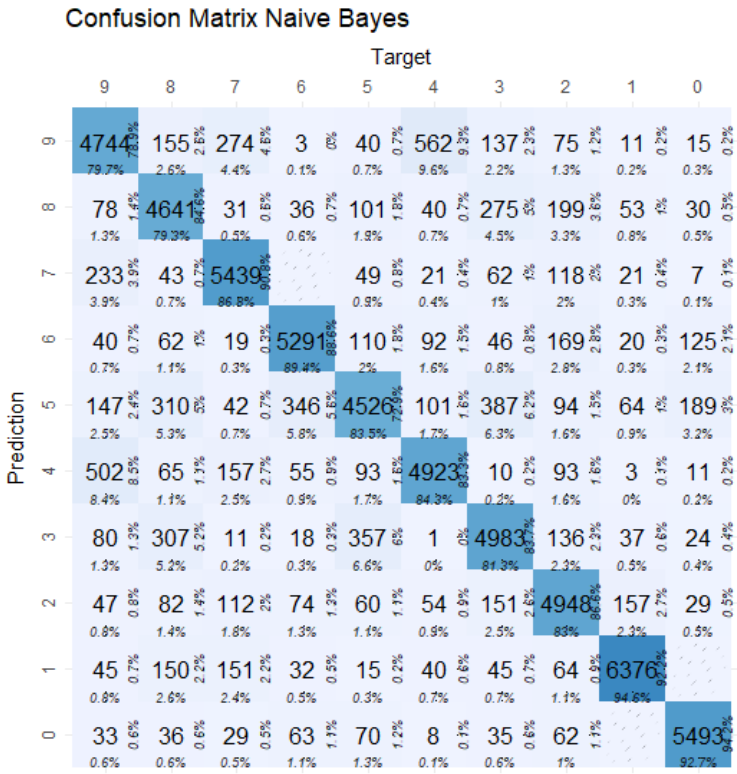

Figure 3.1: Confusion Matrix of *Naive Bayes* model.

The ROC plot and the AUC value show us how well the different classes are separated by the *Naive Bayes* model.

Label	Precision	Recall	F-measure	AUC
0	0.9423572	0.9274017	0.9348196	0.9599893
1	0.9216537	0.9457134	0.9335286	0.9669825
2	0.8659433	0.8304800	0.8478410	0.9100333
3	0.8369164	0.8127549	0.8246587	0.8994126
4	0.8327131	0.8426909	0.8376723	0.9132130
5	0.7292942	0.8349013	0.7785327	0.9017271
6	0.8856712	0.8940520	0.8898419	0.9404663
7	0.9075588	0.8681564	0.8874205	0.9296884
8	0.8462801	0.7931977	0.8188796	0.8906121
9	0.7885638	0.7974449	0.7929795	0.8906554

Table 3.1: Precision, Recall, F-measure and AUC per class.

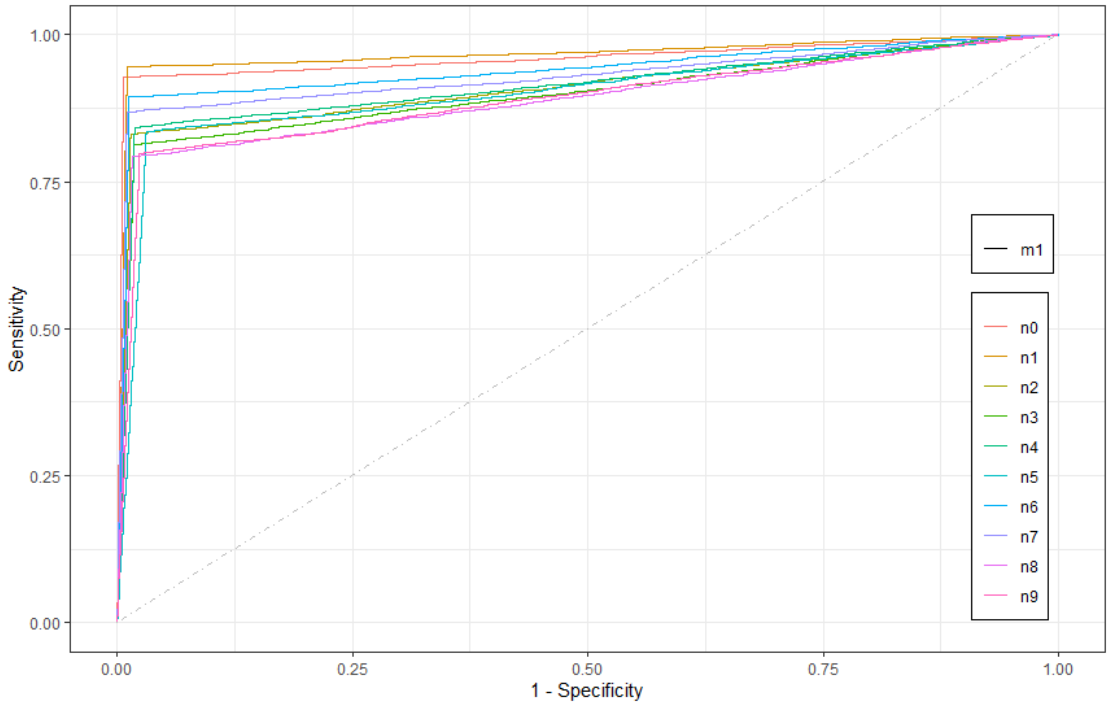

Figure 3.2: Multi-class ROC of *Naive Bayes*. **AUC:** 0,920278.

Chapter 4

Naural Network

As a second model we use a supervised model, *Neural Network*. This model is based on a collection of connected nodes called neurons, similar to a group of human neurons.
Each node has an associated weight, estimated during the training, and a threshold. Like synapses, which propagate an action potential, each neuron transmits a value (which is multiplied by the associated weight) in the *Neural Network*. If the output is above the threshold value, the node propagates the value on the next layer. Otherwise it doesn't pass any values.
During training, weights and thresholds are continually adjusted until training data with the same labels yields similar outputs.
We use this model, because it is robust to noisy data and it is reliable with tasks involving a very large number of features.
To be able to create a *Neural Network* we used the library *Caret*, using the method *mlpML* in order to model a multi-layer perceptron network.
The major problem with a multi-layer *Neural Network*, is to find the best number of neurons for each layer in order to reach the highest level of accuracy. Because there is no rule of thumb, we used the parameter *tuneGrid* which is available for the function *train*. Giving a vector of possible numbers of neurons per layer, it will find the optimal number with maximum Accuracy and Kappa.
In a first run we tried to tune the NN using 2 hidden layer, with 10, 15, 25, 30 and 45 neurons per layer, finding out that the best composition is 45 neurons for the first layer and 15 for the second one.
The accuracy reached is around 59%, which is not that bad.

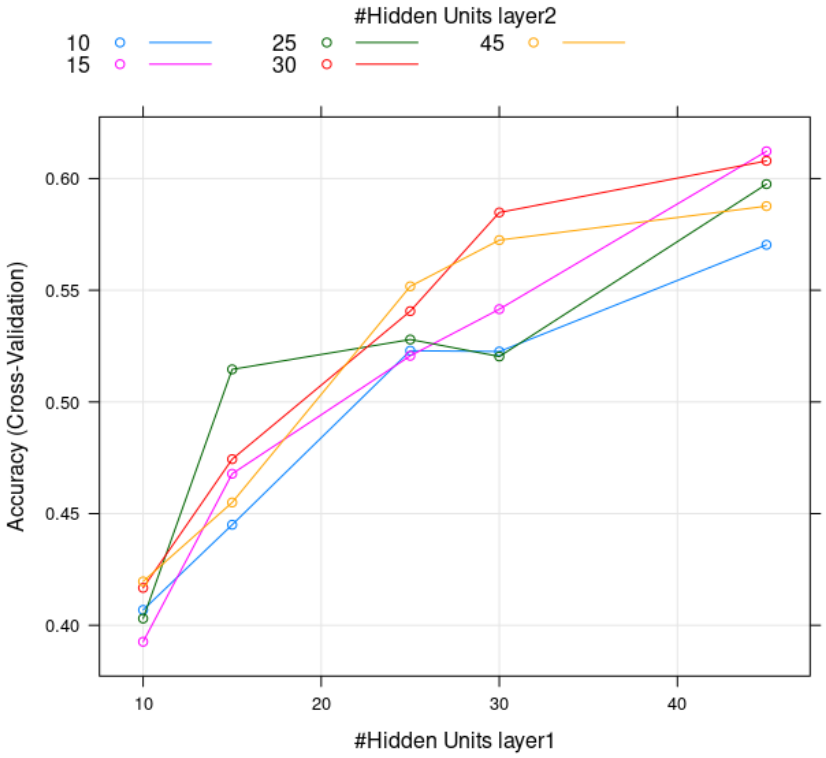

Figure 4.1: Plot of accuracy for different layouts of *Neural Network*.

Looking for better results we tried to tune again the network. For the first layer we test with 100, 200 and 300 neurons and for the second 10, 15, and 20 neurons. The algorithm shows us that the configuration with 300 neurons in the first layer and 20 neurons in the second layer gives an accuracy of 75%.

Increasing the number of total neurons we find higher accuracy. We think this could be due to the reduction of dimensions after the PCA. We decided to stop the tuning with this configuration in order to avoid overfitting and also because we didn't have enough computational resources.

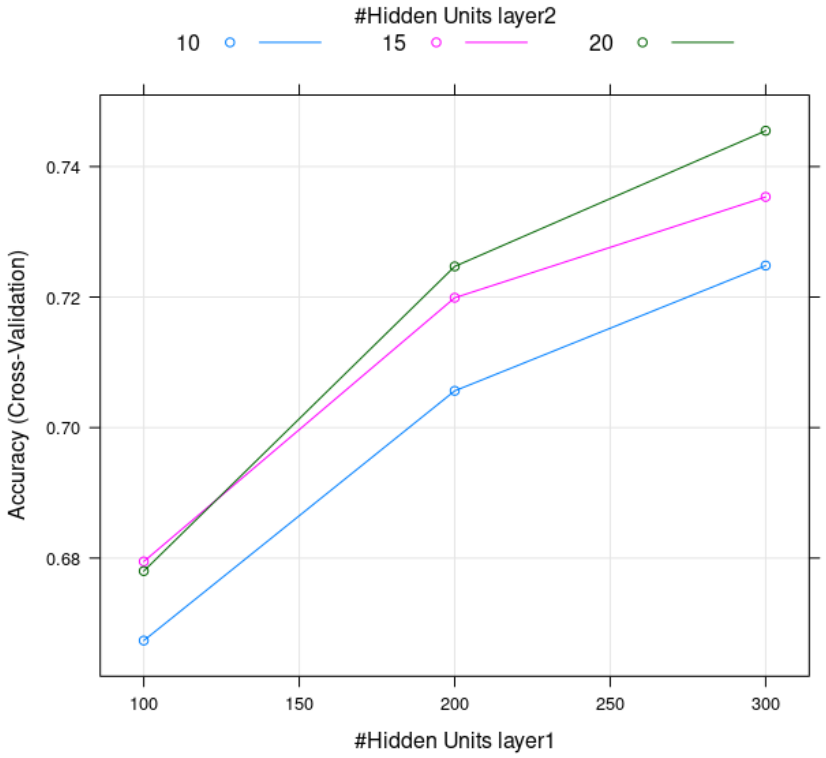

Figure 4.2: Plot of accuracy for different layouts of *Neural Network*.

4.1 Results

During training, we have done a 10-fold cross validation.
We have calculated:

- **Accuracy:** 0.7449164
- **Precision:** 0.742
- **Recall:** 0.72
- **F-measure:** 0.715
- **AUC:** 0.8453
- **Accuracy of test:** 0.7226

.

ROC curve and the Confusion Matrix show us some problems with the classificator. You can see how 9s are often classified as 4s. It could be improved, but we think we need more computational power.

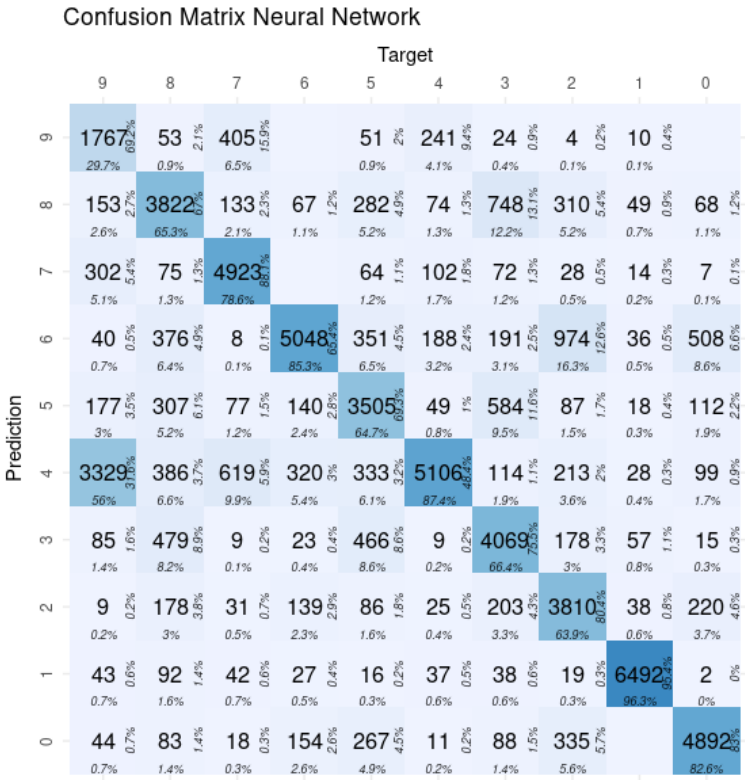

Figure 4.3: Confusion Matrix of *Neural Networ*k model.

Label	Precision	Recall	F-measure
0	0.8302783	0.8259328	0.8280999
1	0.9535840	0.9629190	0.9582288
2	0.8039671	0.6394763	0.7123493
3	0.7549165	0.6636764	0.7063623
4	0.4841187	0.8740157	0.6231009
5	0.6932358	0.6465597	0.6690847
6	0.6538860	0.8529909	0.7402845
7	0.8811527	0.7857941	0.8307459
8	0.6698212	0.6532217	0.6614173
9	0.6915851	0.2970247	0.4155691

Table 4.1: Precision, Recall and F-measure per class.

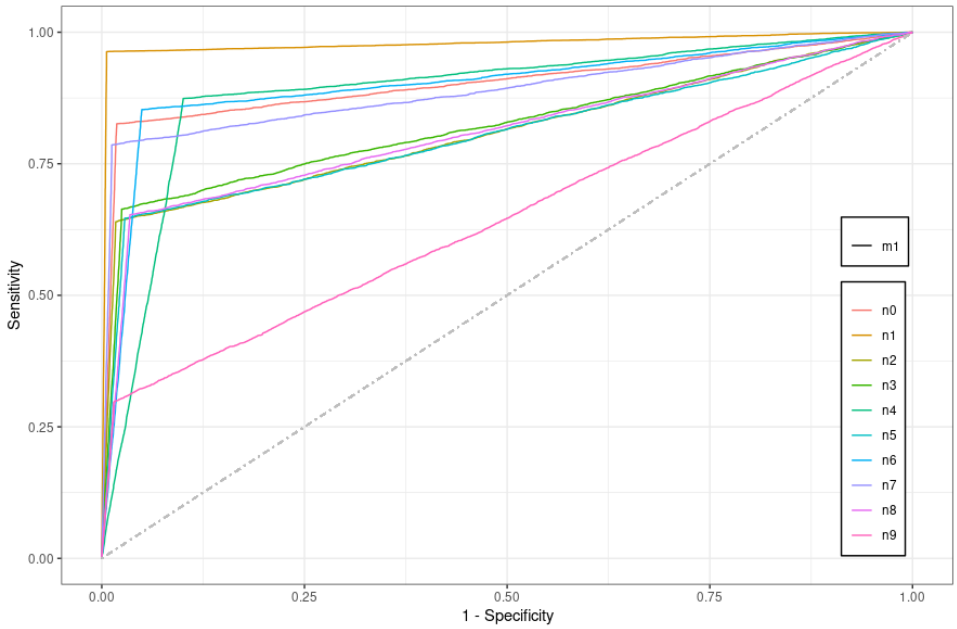

Figure 4.4: Multi-class ROC of *Neural Network* model. **AUC:** 0,845.

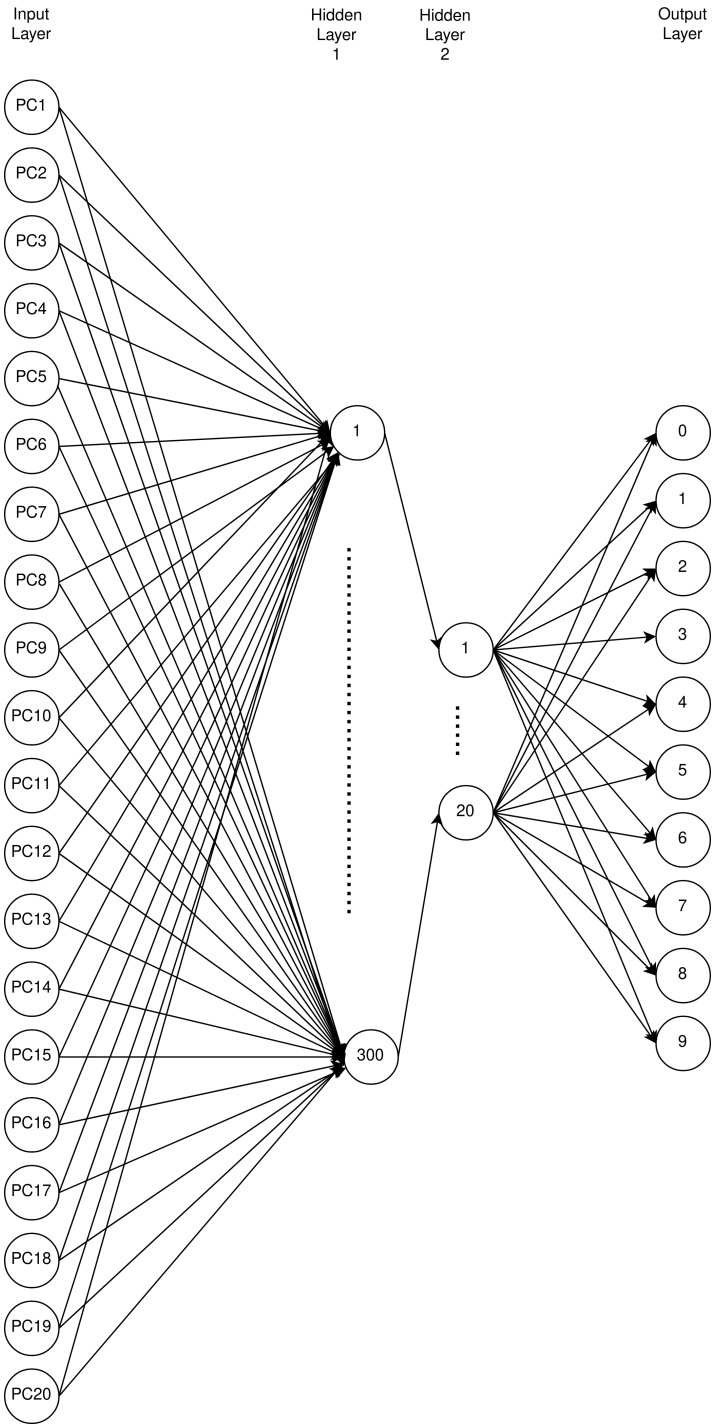

Figure 4.5: Final *Neural Network* layout.

Chapter 5

Results

After the execution of the two models we can see how, in *caret* package, the *Naive Bayes* model performs better than the *Neural Network* one. Anyway we should say that with a right tune of the *Neural Network*'s hyperparameters we could have better results.
Watching at the Accuracy, Precision, Recall, F-measure and AUC values we can state that our *Naive Bayes* model is superior to our *Neural Network*. In addition the t-test shows us how the two models are significantly different, with a p-value of 0.9999747.

	Naive Bayes	Neural Network
Accuracy (test)	0.8621	0.7226
Accuracy (10-fold cv)	0.8520	0.7449
Precision	0.8557	0.742
Recall	0.8547	0.72
F-measure	0.8546	0.715
AUC	0.9203	0.8453

Table 5.1: Accuracy and AUC for each model.

Naive Bayes and Neural Network have different performance characteristics with respect to the amount of training data they receive. The Naive Bayes classifier has been shown to perform surprisingly well with very small amounts of training data that most other classifiers, and especially Neural Network, would find significantly insufficient [Rish et al., 2001]. That could be a reason for the "bad" performance achieved from our Neural Network model. Maybe with more data it would have been better.
Furthermore, the complexities of the two models impact their tendencies to overfit. The simplicity of Naive Bayes prevents it from fitting its training data too closely. In contrast, due to their complexity Neural Networks can overfit very easily training data, especially when provided with large

data sets [Islam et al., 2009]. Therefore add more data to the dataset could lead to better results from Neural Network model in exchange for computational complexity and the risk of overfitting.
Another option is to add more hidden layers to let the Neural Network model fit better but we don't have the computational power to do this so we can't test it for now.

We've also tracked the different times for both models.
All tests are executed with an *AMD Ryzen 7 3700X (32 MB, 8 core, 16 thread, 4.4 GHz)* CPU. The train is executed using all the threads to reduce time, while the prediction are executed normally.

	Naive Bayes	Neural Network
Train (s)	1.37	272.79
Predict test (s)	42.79	0.4821
Predict train (s)	248.40	5.321

Table 5.2: Time to train the models and to predict.

As you can see from the data *Naive Bayes* is fast to train but slow to predict. On the other side *Neural Network* is fast to predict but slow to train.
We can say that the *Naive Bayes* final model has an higher computational complexity compared to the *Neural Network* one.

Chapter 6

Conclusions

In this project we have implemented *Naive Bayes* and *Neural Network* models, using R language and *caret* package, to classify handwritten digits from MNIST dataset.
It has been shown how our R implementation of *Naive Bayes* give better results than our *Neural Network* one, at the cost of an higher computational complexity.
We want to underline the fact that with a right tune of the *Neural Network*'s hyperparameters, maybe through a Bayesian Optimization, we could have better results.
We can, however, say that with a simple tune of the hyperparameters our *Neural Network* model doesn't perform as well as we could expect. On the other side our *Naive Bayes* model, without any tune, performs pretty well.
We suggest to use the *Naive Bayes* model if you want better results and the *Neural Network* one if you want fast results.

Bibliography

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

Md Saiful Islam, Shah Mostafa Khaled, Khalid Farhan, Md Abdur Rahman, and Joy Rahman. Modeling spammer behavior: naïve bayes vs. artificial neural networks. In *2009 International Conference on Information and Multimedia Technology*, pages 52–55. IEEE, 2009.