UNIVERSITÀ DEGLI STUDI
DI GENOVA

MACHINE LEARNING FOR ROBOTICS 1

LAB FINAL REPORT

ROBOTICS ENGINEERING

SUBMITTED BY

Ammar Iqbal, 5183355, Robotics

SUBMITTED TO

Prof. Stefano Rovetta

Abstract

This report is about the Labs proposed by Prof. Stefano Rovetta during the course Machine Learning For Robotics 1.There were four Labs purposed by the Professor each Lab was based on different approach Naive Bayes Classifier, Linear regression, K-Nearest neighbour classifier and Neural network, these approaches were used to do analysis using MATLAB as the tool for the data provided by the professor. The main purpose of this report is to show the execution of these labs and to show the results which ere obtained during this labs.The results were obtained in the form of graphs and matrix's so all the figures are attached in this report and every lab had some scenario and on the basis of the result of the analysis these scenario were answered.

Índice

1. LAB 1 Naive Bayes Classifier

1.1. Introduction

Naive Bayes is an algorithm for classification based on Bayes' Theorem with an assumption of independence among predictors. This is a MATLAB implementation of its behaviour. Laplace smoothing is applied. In statistics, classification is the problem of identifying the category of an observation, based on a training set of data with observations already classified. The classifiers implement a classification algorithm, that maps input data to a category y.

1.2. Naive Bayes Classifier

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}$$

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The classifier is based on the Bayes theorem:

H is the hypothesis

X is the experimental observation

P(H) is the a priori probability of hypothesis H P(X) is the marginal probability of X

P(X—H) is the likelihood of observing X when H is verified.

$$P(X) = \sum_i P(X|H_i)P(H_i)$$

Using Bayes theorem means finding the probability of H happening, given that X has occurred, assuming that the predictors/features are independent. That is that the presence of one particular feature does not affect the others. For this reason, gets the adjective of naive. The marginal probability of X, the probability of observing X in any case, is computed as: i indicates the classes.

$$\begin{aligned} g_i(\mathbf{x}) &= P(t_i)[P(x_1|t_i) \times P(x_2|t_i) \times \ldots \times P(x_d|t_i)] \\ &= P(t_i)\prod_{j=1}^{d} P(x_j|t_i) \end{aligned}$$

Now because the "naïve" assumption is made, the classifier has the following outlook: Where: x=[x1, x2 , . . . , xd] P(ti —x) P(x — ti)P(ti) = P(x1, x2, x3, . . . , xd — ti) P(ti) So the naive assumption is Pr(x1, . . . , xd — ti) = Pr(x1 — ti) Pr(x2 — ti) · · · Pr(xd — ti)

1.3. Implementation

1.3.1. Data preprocessing

The data was downloaded from the Weather data set and the Weather data description. For use with Matlab it is convenient to convert all attribute values into integers¿=1, conversion done by plain text editor.

The attributes are

Outlook (overcast, rainy, sunny)=[1,2,3] Temperature (hot, cool, mild)=[1,2,3] Humidity (high, normal)=[1,2]

Windy (false, true)=[1,2] Classes:

Play (yes,no)=[1,2]

After the conversion the data file could be found in the DataSet.txt and uploaded directly in the Main.m function and divided in training set (training) and test set (test).

The training set will be composed of 10 random observation and a target set of data while the test set will be composed of the 4 remaining observation.

1.3.2. Training phase

The training task was implemented in the function ('Naive-trainer.m') that implements also a Laplace smoothing improvement. Due to this additive smoothing a priori information is needed: in the data preparation set, the information about the number of levels must be added. This means that for each data column the number of possible different values for that column is added; then to compute probabilities, Laplace smoothing is introduced in the formulas by adding terms that consider your prior belief. Since the classifier don't know anything, the prior belief is that all values are equally probable.

1.3.3. Testing phase

Once the Naive Bayes classifier has been trained it can accept the testing set and classify all the observation by computing the Bayes theorem. Two posteriori probability are computed ('Naive-classifier.m') and the higher one is chosen to complete the classification algorithm.

1.3.4. Error calculation

The testing set is a file composed, as well as the training set, of as many rows as the observation and as many columns as the attribute plus one more last target column for the class which can be used to validate the classifier. Once the classifier finishes the classification an error is computed to check the validity.

```
This is the average of error rate:
    0.4475
```

1.4. Conclusions

The error rate appears to float between 0.5 and 1, reaching the minor value of 0. The error rate is demonstrated to be acceptable, confirming the efficiency of this kind of classification, even if is not that effective with a small set of data. Thanks to the smoothing algorithm the results are improved.

2. Lab 2 Linear regression

2.1. Introduction

Linear regression, in statistic, is a model that describes the relationship between two variables and the influence they have on each other. The goal of this process is to find a linear relationship between a target (dependent variable) and one or more predictors based on the measured data (independent variable).

It's one of the most important supervised algorithms, used in many fields to analyze behaviors and efficiency of something.

Here will follow a brief description of the laboratory and the mathematical functions used.

2.1.1. Laboratory tasks

The dataset on which the linear regression algorithm needed to be applied were:

- The Turkish stock exchange data can be downloaded from the U.C.I. Machine Learning Repository
- The MT cars data are available as the command "mtcarsïn the (open source) R statistical/data analysis language and environment,

The first task was to get them with a function that would make it readable by MATLAB.

On the data obtained, the fit to the different linear regression models seen in class which are:

1. One-dimensional problem without intercept on the Turkish stock exchange data
2. Compare graphically the solution obtained on different random subsets (10
3. One-dimensional problem with intercept on the Motor Trends car data, using columns mpg and weight
4. Multi-dimensional problem on the complete MTcars data, using all four columns (predict mpg with the other three columns)

The last task is a test on the regression model, re-running points 1,3 and 4 from the second task using the square error loss (MSE) as objective function on different sizes of the dataset.

2.1.2. Linear regression problem

The linear regression is seen as an optimization problem in which the functional dependency between measured data is approximated. The approximation is applied by the parameter w on the model that predicts the target t with a given observation x, good for every point. It forecasts one continuous variable using observations (= one or more other variables) related to it. The generic goal is to minimize the mean value of the loss over the whole data set:

$$J = \frac{1}{N}\sum_{l=1}^{N}\lambda(y_l, t_l)\,.$$

J is the objective function or cost function,
N is the number of observations,
is the loss function,
tl measured target value,
yl is the inferred target value.
In this laboratory the particular square error loss function was used.

$$J_{\text{MSE}} = \frac{1}{N}\sum_{l=1}^{N}(y_l - t_l)^2$$

The goal is to minimize the mean square error objective with respect to fixed data.
1. One-dimensional problem without intercept This is the value of w obtained

$$w = \frac{\sum_{l=1}^{N} x_l t_l}{\sum_{l=1}^{N} x_l^2}$$

2. One-dimensional problem with intercept The solution in this case can be found by centering around the mean of and of .

$$\bar{x} = \frac{1}{N}\sum_{l=1}^{N} x_l \qquad \bar{t} = \frac{1}{N}\sum_{l=1}^{N} t_l$$

Here we switch from a linear to an affine model = 0 + 1

$$w_1 = \frac{\sum_{l=1}^{N}(x_l - \bar{x})(t_l - \bar{t})}{\sum_{l=1}^{N}(x_l - \bar{x})^2}$$

$$w_0 = \bar{t} - w_1\bar{x}$$

With the following gain and intercept In which 1= slope; gain 0= intercept, offset; bias.

$$\mathbf{w} = (X^T X)^{-1} X^T \mathbf{t}$$

3. Multi-dimensional problem The data is now composed of d-dimensional vectors, stored into a N x d matrix, in which: N is the number of observations, d is the number of features observed. Since the data are now d-dimensional, we have d parameters in a d-dimensional vector. The linear model now results like = where:

2.2. Implementation

2.2.1. Task 1: get the data

As already said here, the data upload was asked suggesting 2 functions: “load” or “csvread”, but because of the mixed structure of the data with numbers and strings the “readmatrix” function was used instead.

2.2.2. Task 2: fit a linear regression model

Here the results of each regression model will be shown.

a- One-dimensional problem without intercept

The function, explained in the previous paragraph at the subsection dedicated to this problem, was implemented on the “linearRegression.m”. This function will calculate the value of w and plot it with the following result.

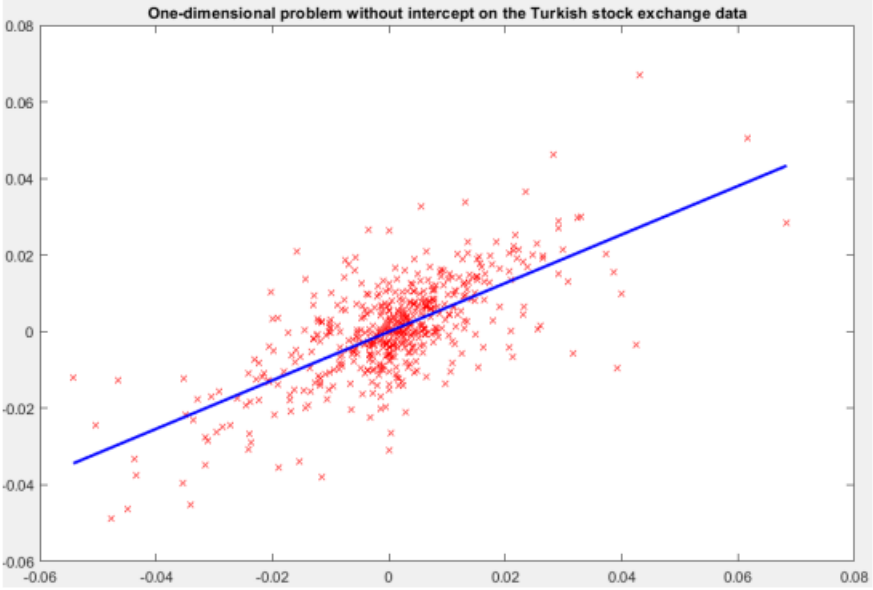


Fig 1D linear regression without intercept

b- Compare graphically the solution

The comparison should be executed on the 10 % of the whole dataset of the stock market. With the function “compare.m” the dataset is split in 9 subsets as shown in Figure 2:

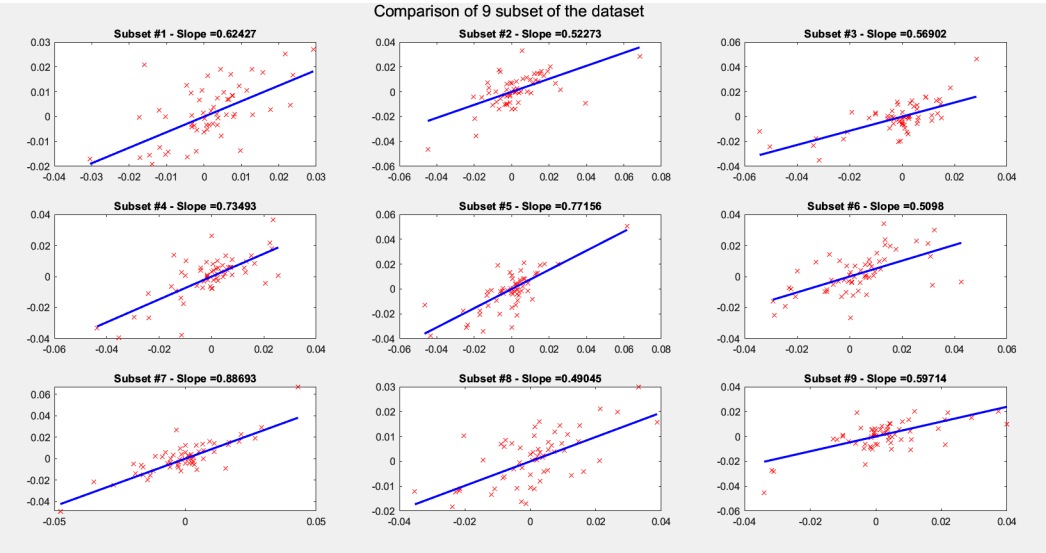


Fig Comparison of different subset of stock market data

c- One-dimensional problem with intercept To predict the mpg and weights features from the motor car trends data. The implementation of the formulas described before in the subsection dedicated to this task is in the "linearReg-offset.m". The result is in figure 3. d- Multi-dimensional problem The prediction of the mpg with all the other feature is done. It was needed to implement the equation of the regression problem in the function "linearReg-mD.m" which will print a table with the dataset values inferred and not (Figure 4)

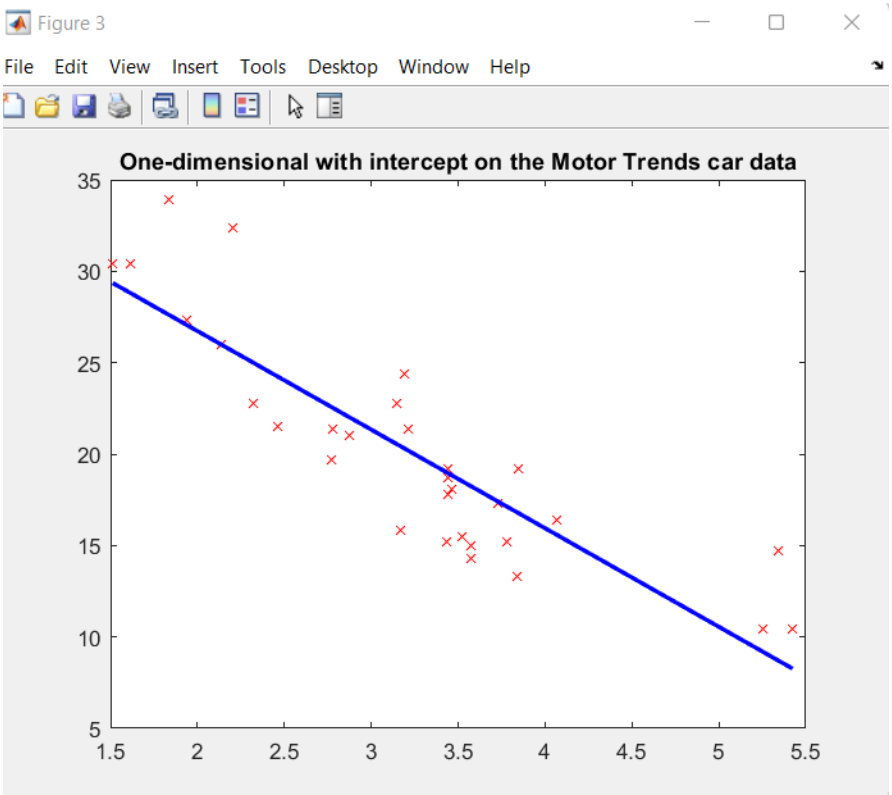


Fig 1D linear regression with intercept

	disp	hp	weight	dataset mpg	Predicted mpg
1	160	110	2.8750	21	20.6889
2	108	93	2.3200	22.8000	19.3273
3	258	110	3.2150	21.4000	13.4010
4	360	175	3.4400	18.7000	7.1052
5	225	105	3.4600	18.1000	19.9438
6	360	245	3.5700	14.3000	11.7141
7	146.7000	62	3.1900	24.4000	23.9120
8	140.8000	95	3.1500	22.8000	25.5554
9	167.6000	123	3.4400	19.2000	27.1433
10	167.6000	123	3.4400	17.8000	27.1433
11	275.8000	180	4.0700	16.4000	24.6319
12	275.8000	180	3.7300	17.3000	20.5587
13	275.8000	180	3.7800	15.2000	21.1577
14	472	205	5.2500	10.4000	17.1127
15	460	215	5.4240	10.4000	21.0243
16	440	230	5.3450	14.7000	23.0503
17	78.7000	66	2.2000	32.4000	20.1094
18	75.7000	52	1.6150	30.4000	12.8386
19	71.1000	65	1.8350	33.9000	16.5742
20	120.1000	97	2.4650	21.5000	19.8360
21	318	150	3.5200	15.5000	11.8429

Fig Multi-dimensional regression

2.2.3. Task 3: Test a regression model

As already described in the introduction paragraph, in this task the MSE was computed on the training set (5 % of the whole data) and apply the model obtained on the remaining 95 %.

	Dataset	Percentage	MSE
1	Train set	4.8507	9.5895e-05
2	Test set	94.9627	9.4647e-05

	Dataset	Percentage	MSE
1	Train set	3.2258	NaN
2	Test set	93.5484	NaN

	Dataset	Percentage	MSE
1	Train set	3.2258	2.3350e-29
2	Test set	93.5484	1.9777e+03

2.3. Conclusion

Some of the results obtained are pretty clear. In Figure 1 we have just a representation of simple linear regression algorithm working on the dataset.

In Figure 2 we can see that the slope obtained for each subset is different from the others, because of the reduced dimensions of the subsets.

Figure 3, as figure 1, is just the representation of the model used.

Figure 4 is the representation of the data calculated by the equation defined in the introduction paragraph.

Figure 5,6,7 display how the MSE on the 95 % of the dataset is higher than the MSE on the train set. That's because the model overfits the data, so the model can'

3. Lab 3 K-NN classifier

3.1. Introduction

k-Nearest neighbours classifier is a non-parametric classification method which inputs are the k closest training examples of a data set, while the output is a class membership. The goal of this laboratory is to implement this classifier in MATLAB on the specified data sets.

3.1.1. K-Nearest neighbour classifier

The KNN algorithm is a simple and easy to implement supervised machine learning algorithm. It's non parametric so it makes no assumption about the probability distribution building a discrimination rule directly from the data.

To implement this algorithm is needed a:

- Training set X=x1, . . . , xl, . . . , xn and a query point
- A value k

Then given the set for training with n examples, it firstly identifies the k-nn training example of the new instance and then assigns the class label with the highest number of neighbours of the new instance.

$$\{n_1, \ldots, n_k\} = \text{top-}k\|\mathbf{x}_l - \bar{\mathbf{x}}\|$$
$$y = \text{mode}\{t_{n_1}, \ldots, t_{n_k}\}$$

3.1.2. Task description

The aim of this laboratory is to develop a k-nn classifier using the mnist data set: it contains a training set, the labels of this set, a test set and the labels of this set. Three functions are also given:

"loadMNISTLabels.m" to load the label,

"loadMNISTImages.m" to load the images,

"loadMNIST.m" to flexibly load the data.

The first two function are anyway useless for this implementation. The data represent handwritten digits in 28x28 greyscale images, representing the numbers from 0 to 9, the training set contains 60000 examples while the test set contains 10000. Loading the data, for each set (training and test) the following elements returns:

1- A matrix composed by the number of examples as rows and by 784 columns (representing the number of image's pixels 28x28);

2- A column vector composed by the number of examples: it contains the labels for 1 to 10, which represents the number of digit showed in the image;

3.2. Implementation

3.2.1. Building the classifier

The function kNN.m is used to implement the classifier. It takes as inputs:
- Training set,
- Test set and its label
- The k values that are chosen are k=[1,2,3,4,5,10,15,20,30,40,50]

And due to the long computational time, the number of observations for both training and test set where educed, namely at 6000 and 1000.

The output is the classification and the error rate, calculated comparing the predicted label with the given ones.

The code does some checks:
- on the number of arguments received (nargin) equals at least the number of mandatory arguments.
- the number of columns of the test matrix should be at least equals the number of columns of the training matrix.
- Check that k¿0 and k¡=cardinality of the training set (number of observations, referred to as n).

After that, considering as query points each observation in the test set, the classification of the test set is done. The Euclidean distance is computed using a MATLAB function "pdist2()", computes the distance using the metric specified by distance (D) and returns the K smallest pairwise distances to observations in X for each observation in Y in ascending order. Then, using "mode()" function, the most frequent value in an array is returned (classification). Having available also the label of the test set as input of the function, the error rate may be computed.

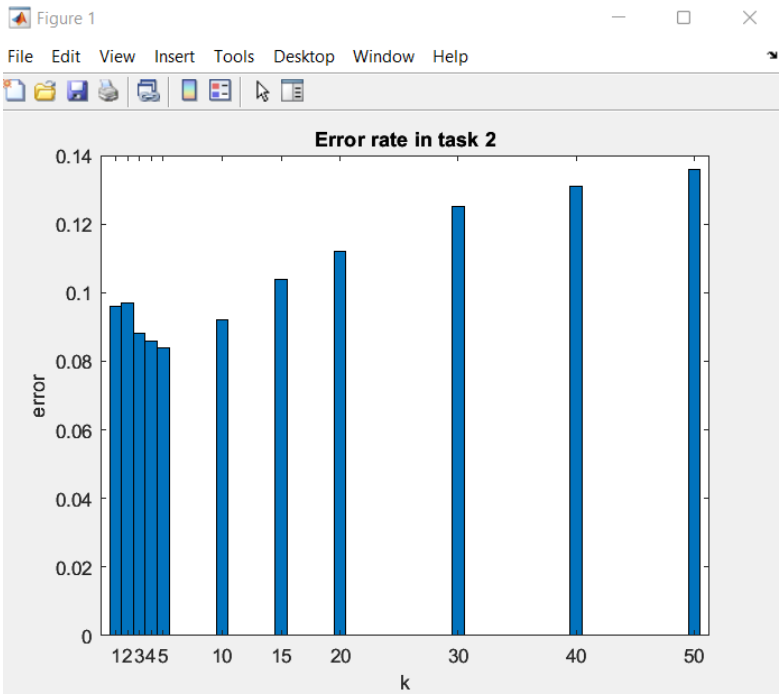
3.2.2. Test the kNN classifier

In this task is asked to use the MNIST character recognition data and to compute the accuracy on the test set in two ways:
- On 10 tasks: each digit vs the remaining 9
- for several values of k, e.g., k=[1,2,3,4,5,10,15,20,30,40,50]

According to what it is required, the results are provided for any combination of these parameter.

3.3. Results

The total results are a lot and contained in the "/results" folder. Here are analysed for explanation just some of them.



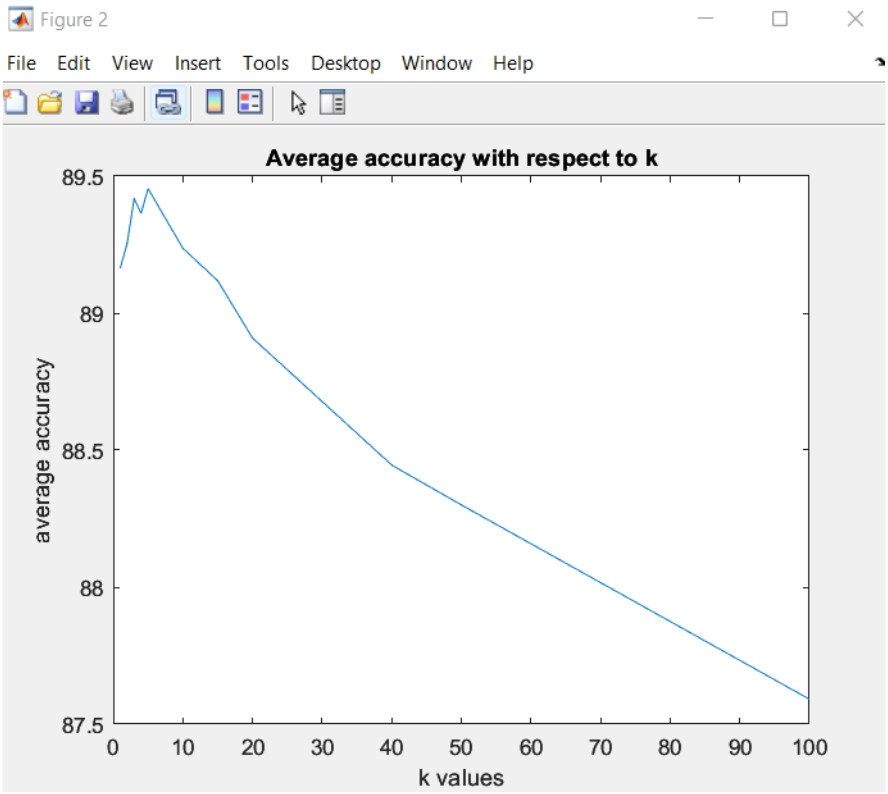


Figure 2
File Edit View Insert Tools Desktop Window Help
Average accuracy with respect to k
average accuracy
k values
89.5
89
88.5
88
87.5
0 10 20 30 40 50 60 70 80 90 100

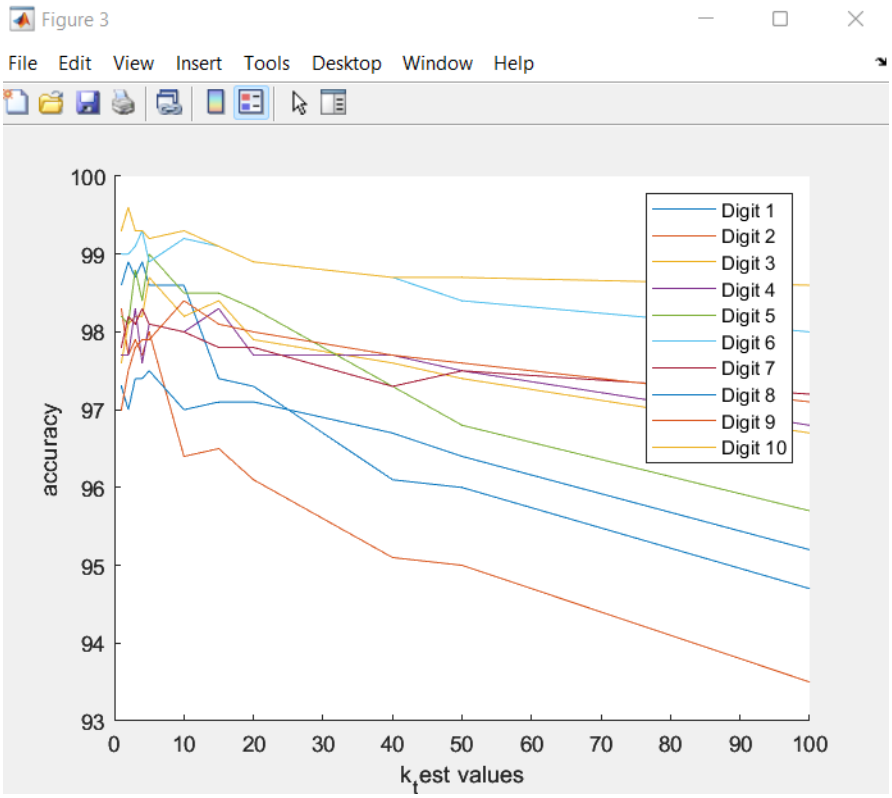


Figure 3
File Edit View Insert Tools Desktop Window Help
accuracy
k_test values
Digit 1
Digit 2
Digit 3
Digit 4
Digit 5
Digit 6
Digit 7
Digit 8
Digit 9
Digit 10
100
99
98
97
96
95
94
93
0 10 20 30 40 50 60 70 80 90 100

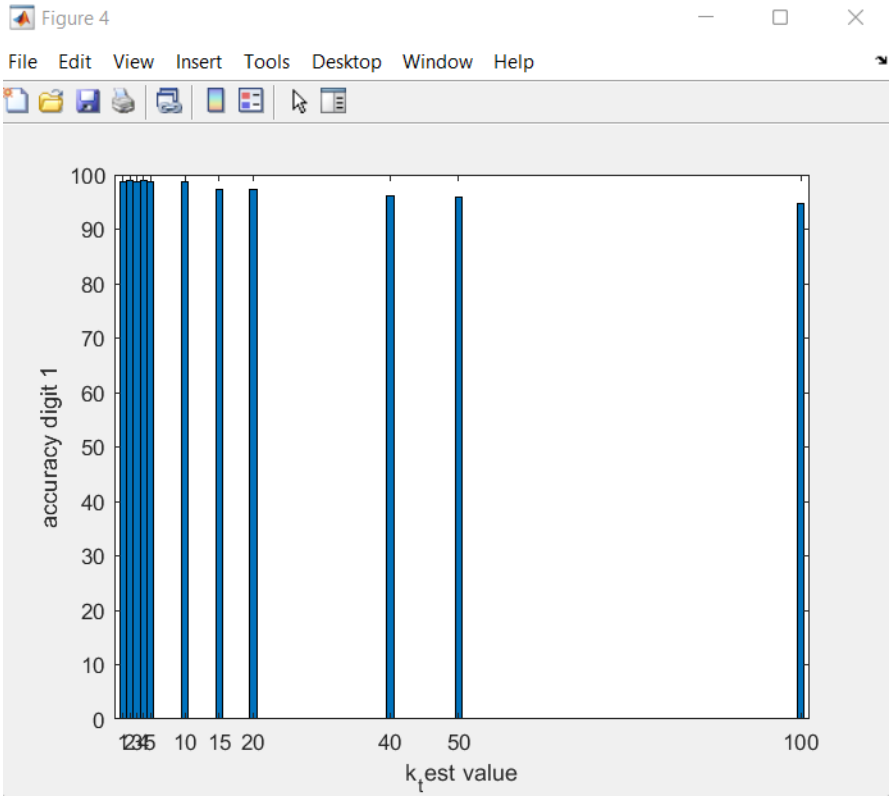


Figure 4
File Edit View Insert Tools Desktop Window Help
accuracy digit 1
100
90
80
70
60
50
40
30
20
10
0
1 2 3 4 5 10 15 20 40 50 100
k_test value

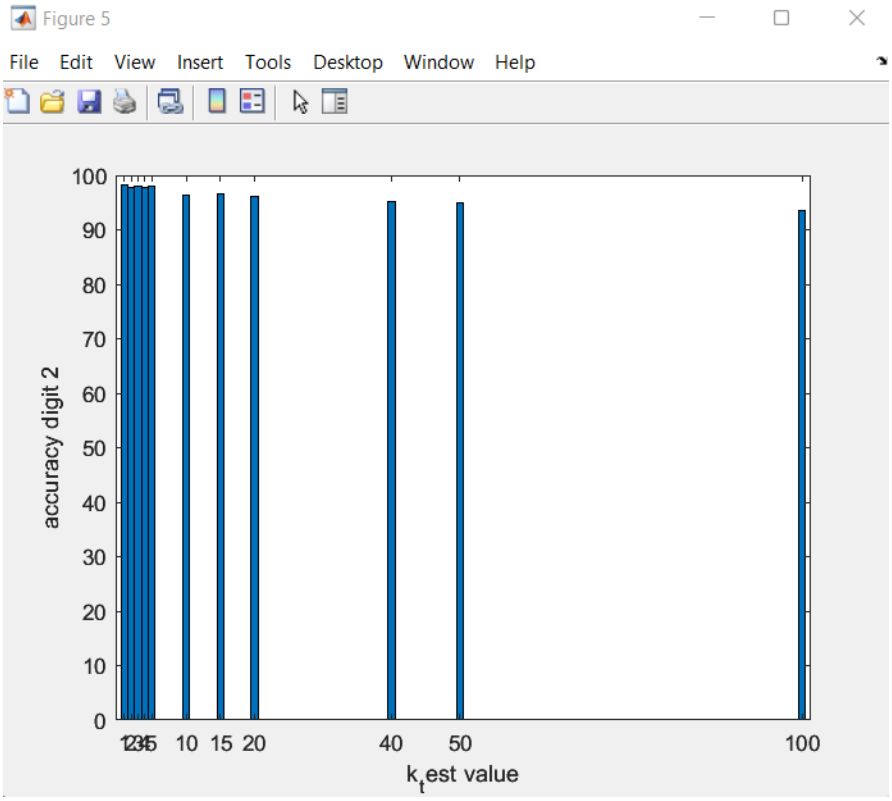


Figure 5
File Edit View Insert Tools Desktop Window Help
accuracy digit 2
100
90
80
70
60
50
40
30
20
10
0
1 2 3 4 5 10 15 20 40 50 100
k_test value

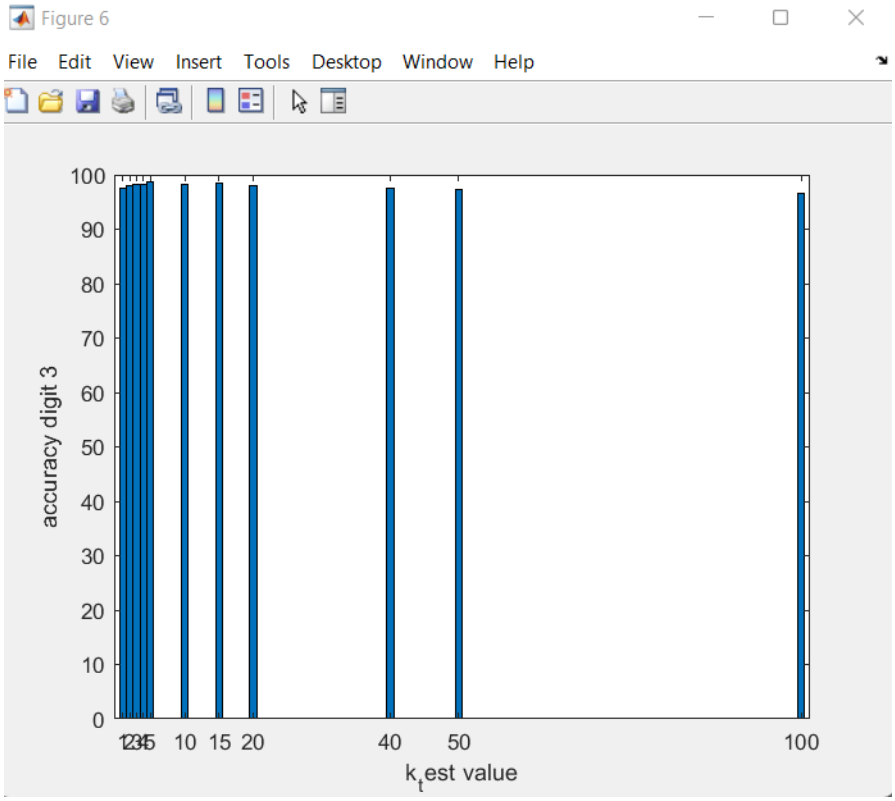


Figure 6
File Edit View Insert Tools Desktop Window Help
accuracy digit 3
k_test value
0 10 20 30 40 50 60 70 80 90 100
1 2 3 4 5 10 15 20 40 50 100

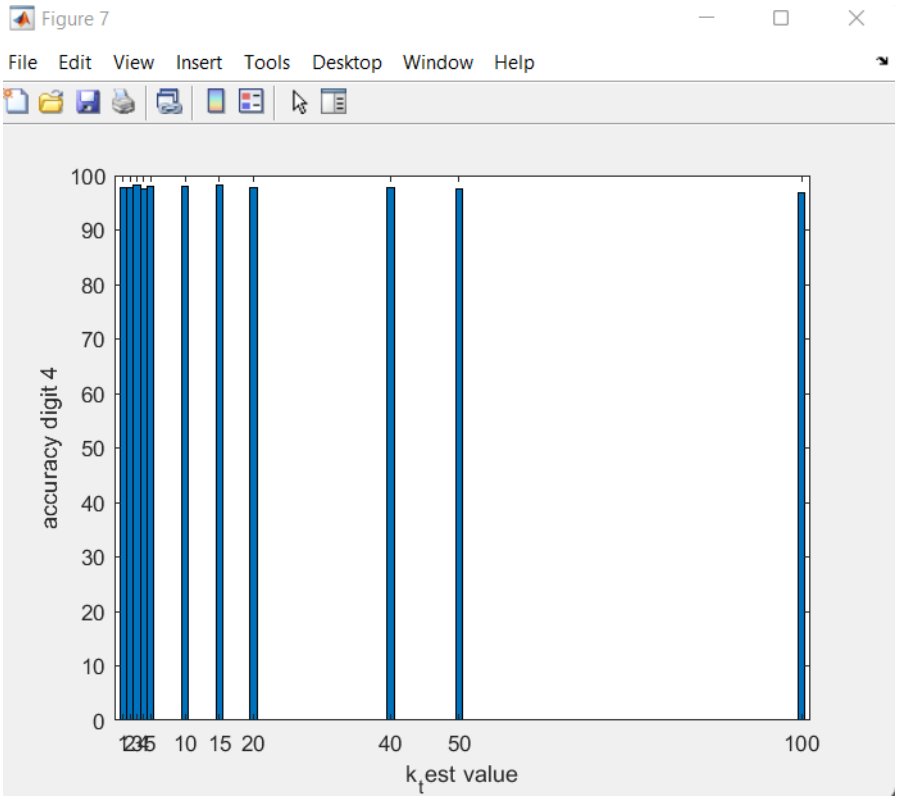
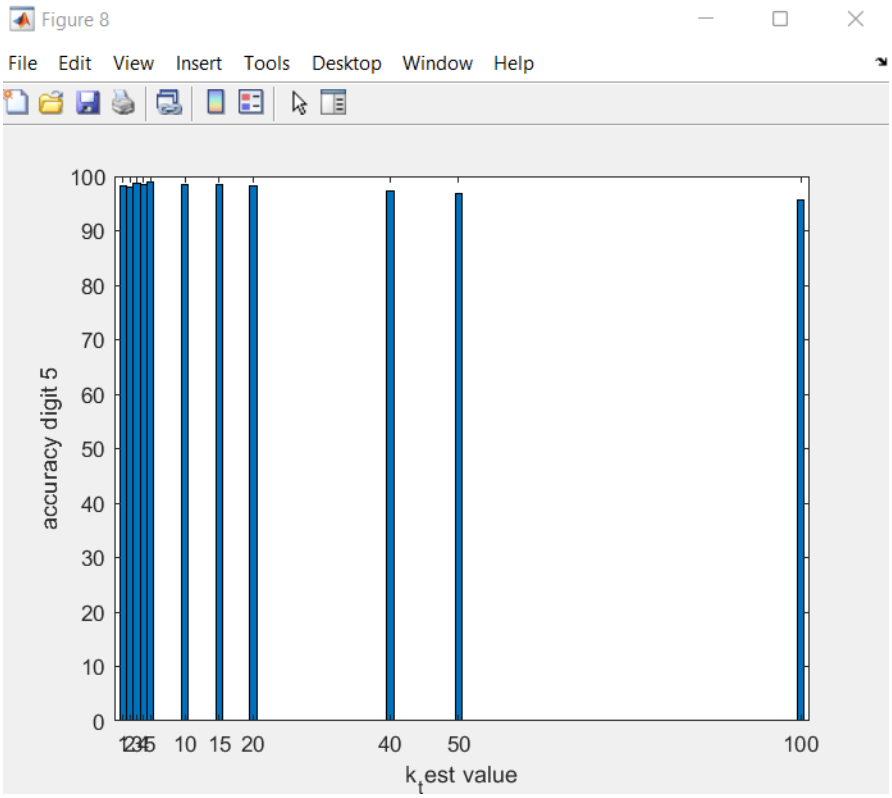
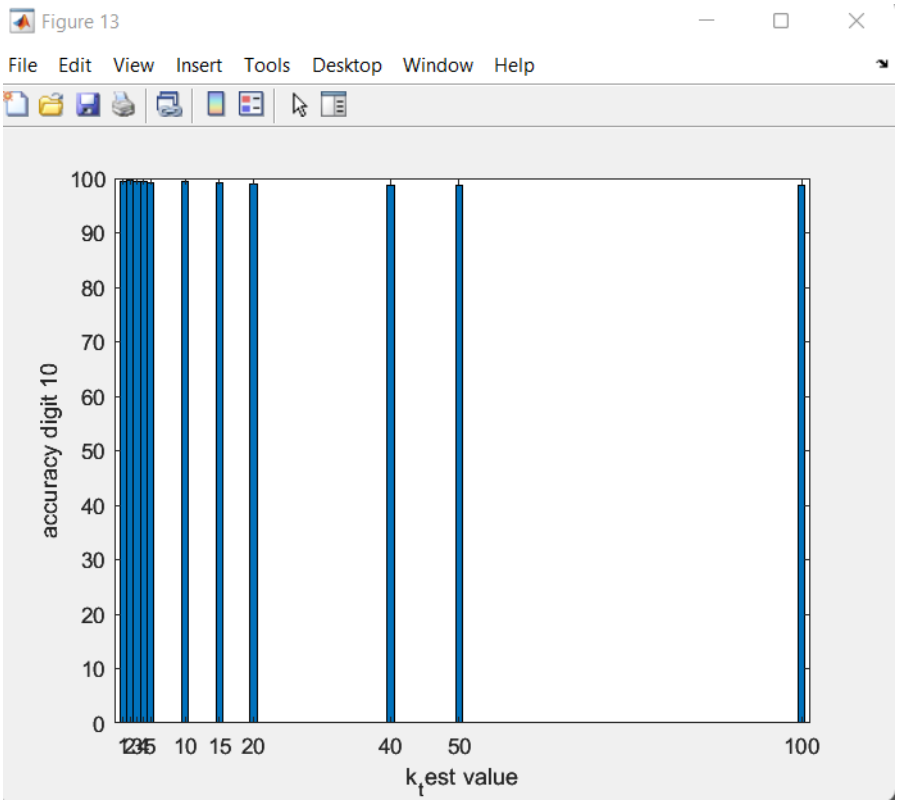


Figure 7
File Edit View Insert Tools Desktop Window Help
accuracy digit 4
k_test value
0 10 20 30 40 50 60 70 80 90 100
1 2 3 4 5 10 15 20 40 50 100





3.4. Conclusion

The accuracy is computed for each digit, but only the bar graph of the digit 1 is showed. According to the bar graph, the accuracy of each digit with respect all value of k is very high (usually between 97/100 per cent)

4. Lab 4 Neural Network

4.1. Introduction

(Artificial) neural networks are networks of several interconnected units (multi-layer) with a simple behaviour used to classify things and make predictions. They're usually trained iteratively. In this tasks execution MATLAB's neural network tools will be used.

a. Task description The aim of this laboratory is to test the different functions of the Neural Network Tool using the dataset available in different suggested repositories like: • UCI Machine Learning Repository https://archive.ics.uci.edu/ml/inc
;

• NIST handwritten digits data set https://www.nist.gov/srd/nist-special-database-19 which contains the already used MNIST handwritten character database.

• Kaggle https://www.kaggle.com/datasets hosts machine learning competitions and has many datasets available, including links to UCI

The first two tasks were two tutorials to get started using the MATLAB tool, using "shallow networks".

The third one is the implementation of the simplest autoencoder network, as example of unsupervised training, since the target set is the input pattern itself. Is required to use the MNIST data set, known from the previous work, in particular the 10-class MNIST digits problem.

The workflow of the experiment is:

- Split the data into subsets of different classes x1, x2, ... , x10 - Create a training set with only 2 classes - Train an autoencoder on the new, reduced training set - Encode the different classes using the encoder obtained - Plot the data using the "plotcl"function given.

4.2. Implementation

4.2.1. Task0: Fitting task

This problem can be approached in 2 ways: - with the neural net fitting app - or the command line functions as suggested by the tutorial I started using the app and then generated the command-line scripts.

The "Body fat data set", which structure is shown in the model summary section (Figure 1.a), is the dataset chosen for this tutorial from the sample dataset provided by the tool.

To define the fitting problem, we need to arrange a set of inputs as a vector (predictors), which will be the columns in a matrix; then arrange a set of responses in a second matrix.

```
predictors = [0 1 0 1; 0 0 1 1];
responses = [0 0 0 1];
```

Once this was defined, training and plotting phase started and the result of the different outputs will be shown in the result section.

4.2.2. Task1: Feedforward multi-layer networks (multi-layer perceptron)

In this section a pattern recognition task is implemented. Here too we have two ways to solve the problem and one of them is using the Neural Net Pattern Recognition app. To define a pattern recognition problem, arrange a set of input vectors (predictors) as columns in a matrix. Then arrange another set of response vectors indicating the classes to which the observations are assigned. When there are only two classes, each response has two elements, 0 and 1, indicating which class the corresponding observation belongs to. After opening the nprtool is possible to start the task choosing dataset, trying to choose two different dimensions of them, in this case will be the Glass Data set (Figure 1) and the Thyroid Data Set (Figure 2). The thyroid can be used to create a neural network that classifies patient's thyroid as normal, hyperfunctioning or subnormal functioning

In the results section we can find the different scenario generated by the plotting functions.

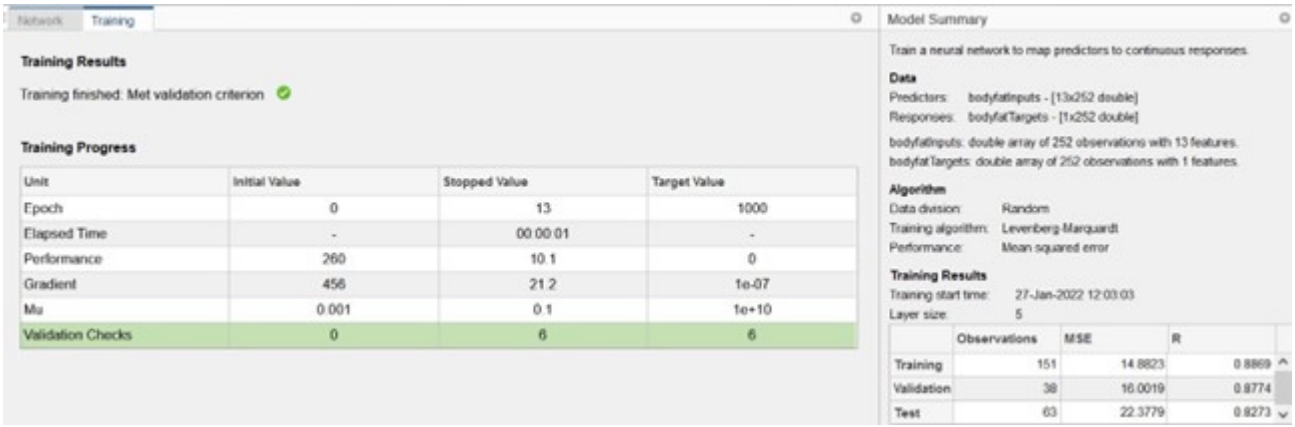
4.2.3. Task2: Autoencoder

Train a multilayer perceptron as an autoencoder for the MNIST data. Following the workflow explained above (section a.), due to the massive amount of data a subset of 2 digits has been chosen, taking 500 random samples for each of them. It was interesting to chose different digits com compare the load of work in learning them for the autoencoder algorithm. The training and encoding phase are completed thanks to the given functions "trainAutoencoder()" and "encode()". Afterwards the output will be plotted in a 2-dimensional plot given by the 2 units of the hidden layer; also the plotting function was given by the project "plotcl()".

4.3. Results

4.3.1. Fitting task results

As specified before these results are for the body fat dataset.
I chose as hidden layer size 5, a test data of 25 and a validation data of 15 the training progress result was:



This table shows what happens at the beginning, what happened when the training stopped (after 13 epochs) and what would have happened at the actual end of the training process.

The information given are:

- The epoch is the number of times the process passed the whole training set - Performance is the accuracy of the training - Gradient value will never be 0 but we'll see it decrease during the process. - Validation checks how many times during the process the validation is calculated.

After that is possible to evaluate the performance plot (Figure 2.a):

Here is possible to notice how the train and test accuracy are a bit distant from each other which means that the random split generated two set with some consistent statistical differences.

For the validation line we can say that the circled point is the minimum value, after that the validation plot is raising and after a bit stopping.

4.3.2. Pattern Recognition results

A comparison between the two-dataset chosen will be developed here. Starting from training the two sets. The same settings for both will be used. Staring with: - Split data in training, validation, and test set, namely 65- Layer size 15 and then 5.

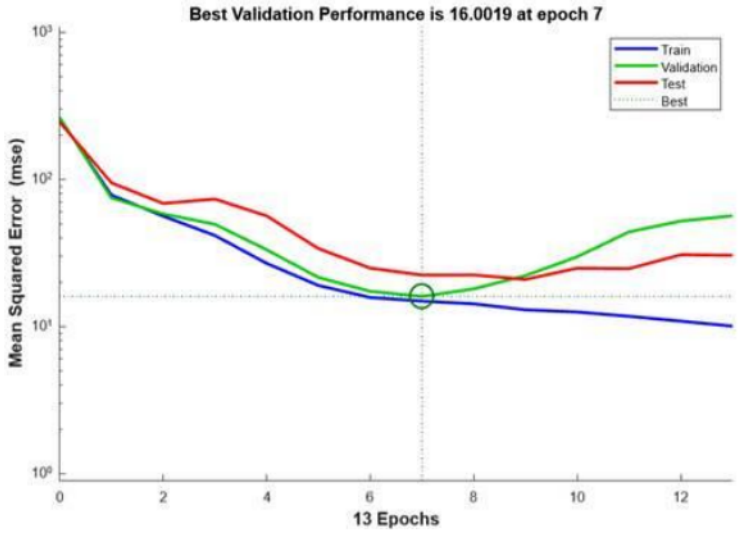
The evaluation of this set will be done on the confusion matrices, to show the classification results for training, testing and validation sets and the results of those data combined, shown in Figure 1.b and Figure 2.b for the Thyroid dataset, and in Figure 3.b and Figure 4.c for the Glass dataset. the accuracy of the outputs can be evaluated by the number of correct classifications

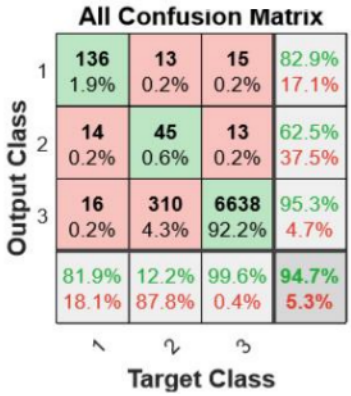
4.3.3. Autoencoder Results

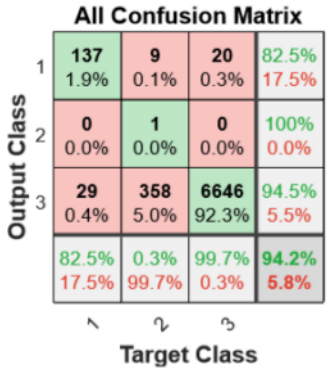
In the plots we see the representation of the two digits problem from MNIST database, and the different colors represent the different neurons (2).

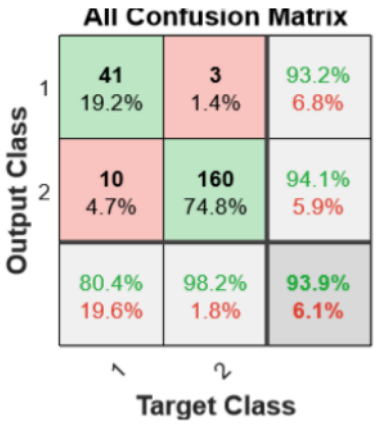
As we can see, the bigger is the difference between the shapes of the digits the easier will the be the linear separation (for instance 1-8 and Figure 2.d), otherwise, the points will result mixed, and to hardly separable (for instance 4-9 and 3-8)

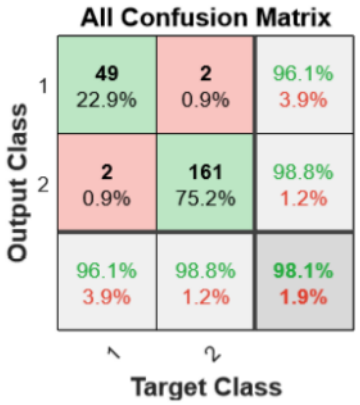
IMAGES











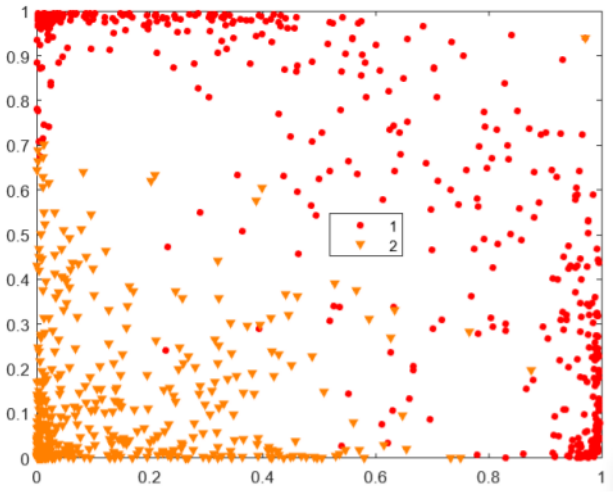


Fig Autoencoder classes 1 and 8

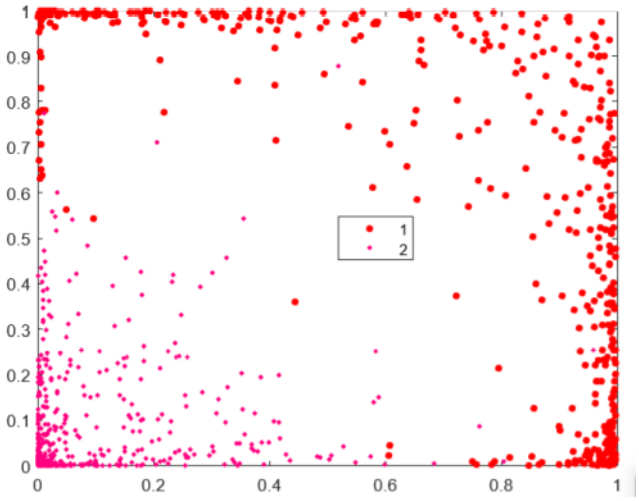


Fig Autoencoder of class 1 and 4

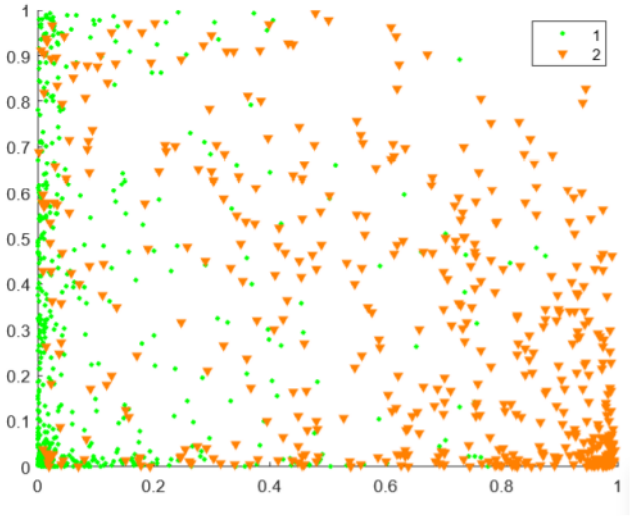


Fig Autoencoder between 3 and 8

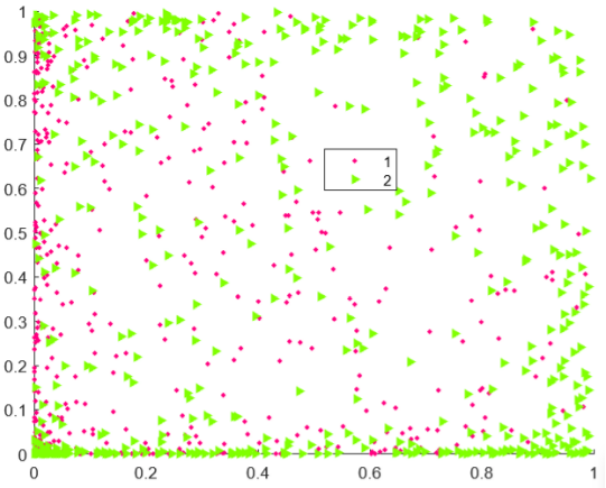


Fig Autoencoder between class 4 and 9