

Università degli Studi di Genova

MASTER DEGREE IN ROBOTICS ENGINEERING
MACHINE LEARNING



UNIVERSITÀ DEGLI STUDI
DI GENOVA

2020 / 2021

Machine Learning Final Report

Presented by

Isabella-Sole Bisio

On Date 02/02/2021

List of Figures

1.1	Translated data	5
2.1	One dimensional Linear Regression without intercept	8
2.2	Comparison of different subsets	9
2.3	One dimensional Linear Regression with intercept	10
2.4	Multi dimensional Linear Regression	10
2.5	From left to right: One-dimensional problem without intercept, One-dimensional problem with intercept, Multi-dimensional problem	11
3.1	Test 1	14
3.2	Test 2	14
4.1	Perceptron Rule	16
4.2	Adaline Rule	17
5.1	Fitting Results	18
5.2	Vintage Wine Dataset Confusion Matrix	19
5.3	Vintage Wine Dataset Receiver Operating Characteristic	19
5.4	Heart Failure Dataset Confusion Matrix	20
5.5	Heart Failure Dataset Receiver Operating Characteristic	20
6.1	Autoencoder Scheme	21
6.2	Autoencoder Architecture	22
6.3	Neural Network Training in Matlab	23
6.4	Similar Digits	24
6.5	Different Digits	24
7.1	Mouse Recognized Using AlexNet	25
7.2	Bell Pepper Classification	26
7.3	Top 5 Bell Pepper Classification	26
7.4	Orangutan Classification	26
7.5	Top 5 Orangutan Classification	26
7.6	Feature Extraction Example	27
7.7	Deep Neural Network Training	28
7.8	Training Results	28

Contents

1	Assignment 1: Naive Bayes Classifier	4
1.1	Introduction	4
1.2	Experiments	4
1.3	Conclusions	6
2	Assignment 2: Linear Regression	7
2.1	Introduction	7
2.2	Experiments	7
2.2.1	Implement Models	7
2.2.2	Testing the implemented models	11
2.3	Conclusions	11
3	Assignment 3: kNN Classifier	12
3.1	Introduction	12
3.2	Experiments	12
3.2.1	Implementation of the classifier	12
3.2.2	Testing of the classifier	13
3.3	Conclusions	13
4	Assignment 4a: Single-unit Neural Networks	15
4.1	Introduction	15
4.2	Experiments	16
5	Assignment 4b: Neural Networks	18
5.1	Contents and Results	18
6	Assignment 5a	21
6.1	Introduction	21
6.2	Experiments	22
6.3	Conclusions	23
7	Assignment 5b: Practice with Deep Learning	25
7.1	Contents and Results	25
7.1.1	Image Classification	25
7.1.2	Image Feature Extraction	27
7.1.3	Depp Learning Training	27

Assignment 1: Naive Bayes Classifier

Abstract

The first assignment deals with the implementation of the Naive Bayes classifier, a probabilistic machine learning model used for a classification task.

It is called "naive" since it operates on the assumption of independence of the features, that is not always true.

1.1 Introduction

The core of the *Naive Bayes classifier* is based on the *Bayes Theorem*; for the *Bayes Rule*, the a posteriori probability of a given class for an input X , is proportional to the likelihood of the class input multiplied by the P probability of that class.

$$P(H|X) = \frac{P(X|H)P(H)}{\sum_i P(X|H_i)P(H_i)} \quad (1.1)$$

The probability of the vector X is the probability of its single components, but in general the probability of two general events is not given by the multiplication of the probability of these events. With the *Naive assumption* we presume instead that this fact is always true [1].

$$P(X|H) = P(x_1, x_2, \dots, x_n|H) = P(x_1|H) \dots P(x_n|H) \quad (1.2)$$

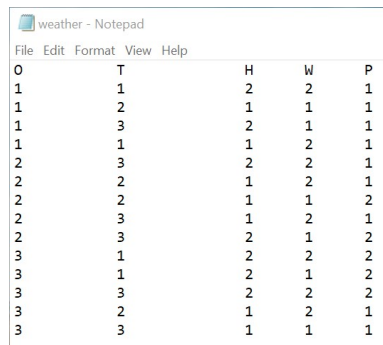
In other words, we are pretending that the input variables are all independent from each other. The goal of this laboratory is to implement the algorithm as a Matlab function and then improve it using a *Laplace smoothing*.

1.2 Experiments

Task 1: Data preprocessing

The goal is to apply the *Naive Bayes Classifier* to understand whether or not a particular climatic situation is suitable to play outdoor. To begin the laboratory, two files were provided: a data file and a description file of the data. Regarding the data, it is possible to determinate a first analysis; the problem to be solved is a two-class problem, the data are multivariate, meaning that I have more than one input variable, and the task is supervised since it is a

classification. The number of instances is 14 (rows of the table), whereas the number of attributes is 4 (columns of the table). The features used here are categorical; this means that I can enumerate the values that they have. In the first line of my table, I have associated meaning for each column: from 1 to 4 they represent input feature, the fifth column is the target class. The latter represents also the correct classification for the data of the data set. I have employed this column for those lines of the data set that I have used as a training set (exploiting the supervised information class for the training set), but I didn't use it for the testing set. Moreover, inside the table, there are 3 levels of outlook, 3 levels of temperature, 2 levels of humidity, and the windy information, that can be only set to true or false. The first thing I had to do with these data sets was to convert their attribute values into integers, to index matrices: 1, 2, or 3 were used for outlook and temperature, 1 and 2 for humidity, windy and play information, as it is possible to observe from the image 1.1



	T	H	W	P
0				
1	1	2	2	1
1	2	1	1	1
1	3	2	1	1
1	1	1	2	1
2	3	2	2	1
2	2	1	2	1
2	2	1	1	2
2	3	1	2	1
2	3	2	1	2
3	1	2	2	2
3	1	2	1	2
3	3	2	2	2
3	2	1	2	1
3	3	1	1	1

Figure 1.1: Translated data

For what regards the algorithms implemented to fulfill this first task, the correspondent Matlab table was loaded. Then, to split the data set into two parts (one part will be used for training and the other for testing), a vector of indexes with a random number between 1 and n=14 was built. In this way, a data set with 10 random data set rows and all the columns were created as a training set, whereas the remaining data set rows and d-1 columns were stacked as a test set. Furthermore, a target variable was created from the last column of the data set.

Task 2: Build a Naive Bayes classifier

To fulfill this second task, the function named *nbc* was implemented in Matlab: it has as inputs the matrices of the training set, test set, and target, while creating as output the classification and the error rate. Firstly, the function performs a dimensional check, to verify if the test set has one column less than the training one. Secondly, it investigates that all the values inside matrices are greater or equal than 1. Finally, it evaluates all the probabilities needed to train the classifier on the training set.

- The a priori probability of hypothesis H is computed as the ratio of the number of times the H class appears and the total number of observations in the training set. There will be an a priori probability for each class, as shown below.

$$P(\text{play} = 1) = \frac{9}{14} \quad (1.3)$$

$$P(\text{play} = 2) = \frac{5}{14} \quad (1.4)$$

- The likelihood of observing X when H holds, is computed as the ratio of the number of times when X appears while H is true and the number of time the H class appears.

$$P(outlook = 2|play = 1) = \frac{3}{9} \quad (1.5)$$

$$P(temperature = 1|play = 2) = \frac{2}{5} \quad (1.6)$$

Once the training is finished, the function will classify the test set; the most probable class value will be returned for each observation. Then, according to *Bayes Theorem*, the a posteriori probability should be computed for all the observations and all class values. The function will now compute which a posteriori probability is the highest, and its respective class values will become the classification. If there is the target input then also the error rate is computed: in fact, it is the ratio between the number of times in which the classification is different from the target and the total number of observations in the test set.

Task 3: Improve the classifier with Laplace Smoothing

To increase the performances of the classifier, it is possible to consider only some particular situations, for instance, when are present only some values in the test set: in this case, the value does not appear in the training set, so my classifier is unable to classify and it will return an error. In these specific situations the new likelihood is computed as follow:

$$P(X|H) = \frac{NumberOfTimesThatXAppearsWhenHIsTrue + a}{NumberOfTimesHClassAppear + an} \quad (1.7)$$

To solve this problem, a *Laplace Smoothing* has been implemented during the evaluations of the probabilities. It is possible to do so since I am adding something that considers the prior belief, and since nothing is known about the prior belief, then all data are equally probable. To behave like this, the *nbcSmoothed* function was implemented.

1.3 Conclusions

In conclusion, it was possible to observe that, even running several times the functions, the last improvement is not so efficient: this can happens if the input training set has all the possible values of each feature, adding a *Laplacian Smoothing*, that means distorting the real probabilities. On the other hand, if the input training set has some missing values that instead appear in the test set, the result with the Laplacian is better than the result without it.

Assignment 2: Linear Regression

Abstract

In the Statistics field, Linear Regression points out a model able to outline associations between a pair of variables and how they influence each other. The goal of this procedure is to identify a linear relationship between a target and one or more predictors based on measured data. Regarding the Artificial Intelligence and Machine Learning field, Linear Regression is recognized as one of the fundamental supervised algorithms; basically, it displays how the changes in the target, dependent variable, can be grabbed by changes in the observed and measured data, independent variables [2].

2.1 Introduction

In this second assignment, the goal is to develop *Linear Regression* examples and evaluate their execution through graphics and *Mean Square Error*. Two dataset were provided, *turkish-se-SP500vsMSCI* and *mtcarsdata-4features*, in order to perform the tests. Firstly, it was required to manage the datasets to make them usable by the Matlab software and then implement a *Linear Regression* problem, carrying out the following tasks:

1. One-dimensional problem without intercept on the Turkish stock exchange data
2. Compare graphically the solution obtained on different random subsets (10%) of the whole data set
3. One-dimensional problem with an 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

2.2 Experiments

2.2.1 Implement Models

To implement the various steps, the lecture slides have been followed, and all the details can be retrieved from them. Essentially, what we want to perform is an *optimisation problem*, where w approximates the model which predicts the target t in the observation x . Using the Objective function J , the main assignment is to obtain a minimised mean value of the loss over the whole

dataset. This can be explicated with the below formula, where λ is a loss function, n the observations, t the target and y the inferred target value.

$$J = \sum_{i=0}^n \lambda(t_i, y_i) \quad (2.1)$$

Furthermore, for the specific requirements of the assignments, the variant of the loss function for the *Mean Square Error* was used:

$$J_{MSE} = \sum_{i=0}^n \lambda(t_i, y_i)^2 \quad (2.2)$$

keeping into account that the goal is to minimise that function, which is synonym of obtain its variation with respect to w equal to zero. The one pointed out until now, was the general overview of the assignment; at this point, it is possible to go through the four tasks cited in the introduction.

One-dimensional problem without intercept

For this task, the Turkish stock exchange dataset was used. In this specific case, the below formula was used to retrieve the w value:

$$w = \frac{\sum_{i=1}^n (x_i t_i)}{\sum_{i=1}^n (x_i)^2} \quad (2.3)$$

Finally, the model is implemented as follows:

$$y = wx \quad (2.4)$$

Using the information above-calculated, it is possible to plot the graph of this representation 2.1:

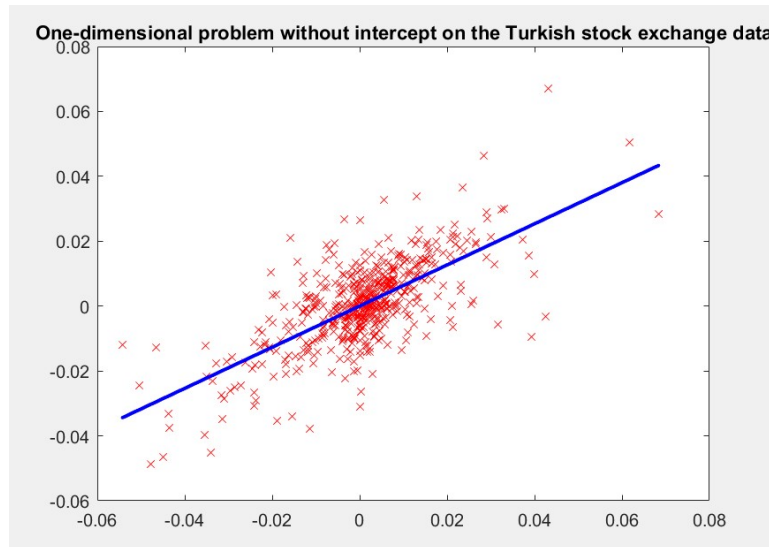


Figure 2.1: One dimensional Linear Regression without intercept

Compare graphically the solution obtained on different random subsets (10%) of the whole data set

For this task different random subsets (10%) of the whole dataset were created to produce a comparison. The following graph shows 9 different subsets 2.2 using various slopes for the line:

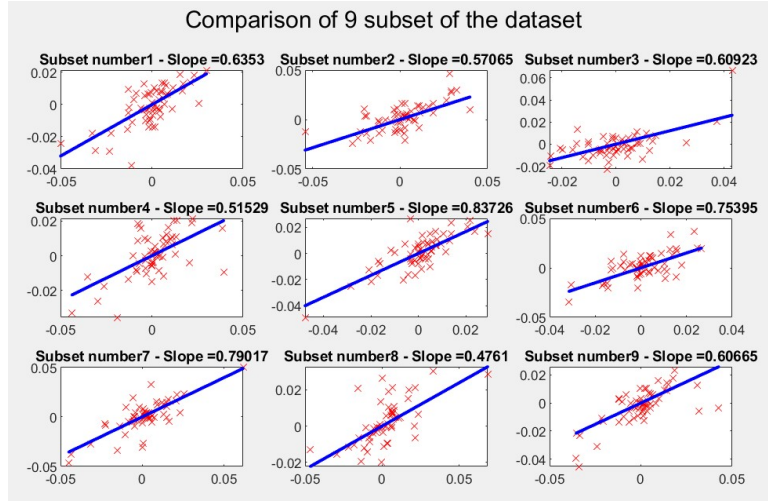


Figure 2.2: Comparison of different subsets

One-dimensional problem with intercept

This third task was performed on the Motor Trends car data, using columns mpg and weight. For this specific case some new evaluations should be performed in order to obtain the desired requirements. The mean of x and t have to be calculated as follows:

$$\bar{x} = \sum_{i=1}^n (x_i) \quad (2.5)$$

$$\bar{t} = \sum_{i=1}^n (t_i) \quad (2.6)$$

The actual model will then be:

$$y = w_1 x + w_0 \quad (2.7)$$

with specific gain equals to:

$$w_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(t_i - \bar{t})}{\sum_{i=1}^n (x_i - \bar{x})^2} \quad (2.8)$$

with intercept:

$$w_0 = \bar{t} - w_1 \bar{x} \quad (2.9)$$

The plot of this case is the following 2.3:

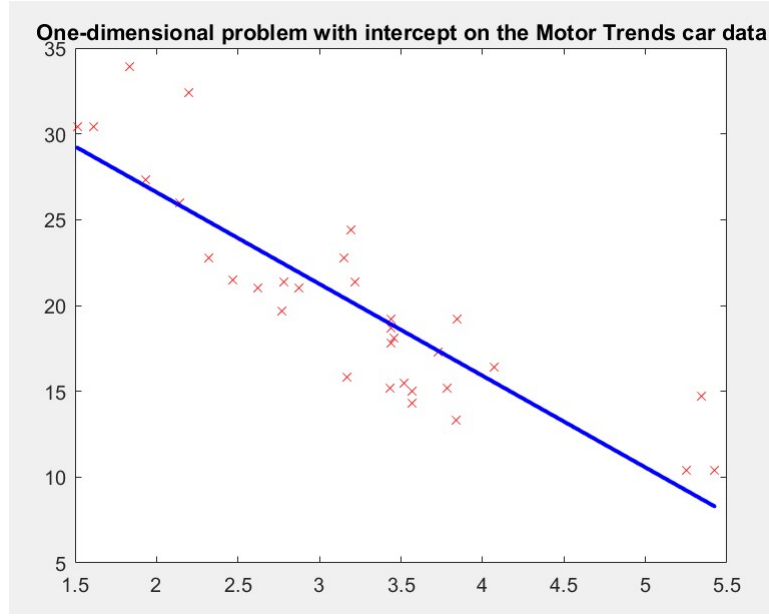


Figure 2.3: One dimensional Linear Regression with intercept

Multi-dimensional problem

This last task was carried out on the complete MTcars data, using all four columns, predicting mpg with the other three columns. Since the data is now d -dimensional, w turns into a vector of dimension $1 \times d$ and t becomes a vector of dimension $1 \times d$. This time w is evaluated as follows:

$$w = (X^T X)^{-1} X^T t \quad (2.10)$$

and the linear model is:

$$y = Xw \quad (2.11)$$

The developed function performs the above-calculations and displays a table with the dataset values and the inferred ones, as shown below 2.4:

	disp	hp	weight	dataset mpg	Predicted mpg
1	160	110	2.6200	21	17.7300
2	160	110	2.8750	21	20.8063
3	108	93	2.3200	22.8000	19.4427
4	258	110	3.2150	21.4000	13.4556
5	360	175	3.4400	18.7000	7.0986
6	225	105	3.4600	18.1000	20.0486
7	360	245	3.5700	14.3000	11.7344
8	146.7000	62	3.1900	24.4000	24.0573
9	140.8000	95	3.1500	22.8000	25.7103
10	167.6000	123	3.4400	19.2000	27.3040
11	167.6000	123	3.4400	17.8000	27.3040
12	275.8000	180	4.0700	16.4000	24.7577
13	275.8000	180	3.7300	17.3000	20.6560
14	275.8000	180	3.7800	15.2000	21.2592
15	472	205	5.2500	10.4000	17.1604
16	460	215	5.4240	10.4000	21.1001
17	440	230	5.3450	14.7000	23.1416
18	78.7000	66	2.2000	32.4000	20.2359
19	75.7000	52	1.6150	30.4000	12.9155
20	71.1000	65	1.8350	33.9000	16.6769
21	120.1000	97	2.4650	21.5000	19.9532
22	318	150	3.5200	15.5000	11.8763

	disp	hp	weight	dataset mpg	Predicted mpg
11	167.6000	123	3.4400	17.8000	27.3040
12	275.8000	180	4.0700	16.4000	24.7577
13	275.8000	180	3.7300	17.3000	20.6560
14	275.8000	180	3.7800	15.2000	21.2592
15	472	205	5.2500	10.4000	17.1604
16	460	215	5.4240	10.4000	21.1001
17	440	230	5.3450	14.7000	23.1416
18	78.7000	66	2.2000	32.4000	20.2359
19	75.7000	52	1.6150	30.4000	12.9155
20	71.1000	65	1.8350	33.9000	16.6769
21	120.1000	97	2.4650	21.5000	19.9532
22	318	150	3.5200	15.5000	11.8763
23	304	150	3.4350	15.2000	12.4870
24	350	245	3.8400	13.3000	16.1603
25	400	175	3.8450	19.2000	7.3100
26	79	66	1.9350	27.3000	17.0039
27	120.3000	91	2.1400	26	15.7461
28	95.1000	113	1.5130	30.4000	12.0911
29	351	264	3.1700	15.8000	8.7933
30	145	175	2.7700	19.7000	24.1410
31	301	335	3.5700	15	22.5733
32	121	109	2.7800	21.4000	24.1740

Figure 2.4: Multi dimensional Linear Regression

2.2.2 Testing the implemented models

In this section it was required to Re-run 1,3 and 4 of the introduction's bullet list, using only 5% of the data. The required work was:

- Compute the objective (mean square error) on the training data;
- Compute the objective of the same models on the remaining 95% of the data;
- Repeat for different training-test random splits.

The tables below 2.5 show the results for a split of 95% and 5%, as the assignment required.

	Dataset	Percentage	MSE
1	Training set	3.1250	NaN
2	Test set	93.7500	NaN

	Dataset	Percentage	MSE
1	Training set	4.8507	8.2455e-05
2	Test set	94.9627	9.6593e-05

	Dataset	Percentage	MSE
1	Training set	3.1250	2.2928e-29
2	Test set	93.7500	1.2921e+03

Figure 2.5: From left to right: One-dimensional problem without intercept, One-dimensional problem with intercept, Multi-dimensional problem

2.3 Conclusions

To summarize the work done in this assignment, it can be said that *Linear Regression* is a simple example of a learning problem. It is possible to make predictions on one variable using observations made on another variable related to the first one. As it has been possible to observe from the result of the assignment, there are various limits for this method; the main one is the simplicity of the algorithm, which leads to the need of having a linear relationship between the variables taken into the examination. The tables result shown that, while using a percentage for the training set of 5% of the whole dataset, the model overfits the data and the MSE obtained on the rest of the dataset is rather higher concerning the one obtained on the train set. Moreover, it is expectable that increasing the training set, the gap between the two MSE reduces because the model, thanks to the bigger quantity of observation available, manages to generalize more.

Assignment 3: kNN Classifier

Abstract

Non-parametric models have a non-pre-defined complexity but it depends on specific data: this is because non-parametric statistics make no assumptions with regards to probability distributions. For this assignment, the k-nearest neighbors' algorithm (kNN) will be used; it is a straightforward method, categorized as a supervised machine learning algorithm based on a non-parametric method, which is useful for regression and classification issues [3].

3.1 Introduction

The guideline of this algorithm consists of considering that similar things are always in proximity to each other, in fact, the *kNN algorithm* integrates the idea of an analogy between objects evaluating the distance among points; in the assignment situation, it will be used for classification purposes. The classifier will output a class label, achieved through the majority voting from the k nearest neighbors. The goal is to develop and test a *kNN classifier*, computing the accuracy on the test set for several values of k and among different digits. The *MNIST dataset* was provided for this laboratory, which consists of a store of handwritten digits and it has a considerable size. Firstly, the data has to be loaded on the Matlab software. Secondly, by keeping into account the guidelines provided, a classifier has to be implemented by primarily perform a dimensional check. The classifier function would have to return the classification obtained. Finally, the *kNN classifier* has been tested computing the accuracy of the test set in the following scenarios; on 10 tasks with each digit vs the remaining 9 and for several values of k .

3.2 Experiments

3.2.1 Implementation of the classifier

All the following work was carried out using the slides guidelines: more accurate details regarding the methodology used can be found in the lecture notes. To implement a *kNN classifier algorithm* the below steps should be developed:

1. Compute the distances between two sets of points, the query point and the training set.
2. Take out the k smallest distance with relative class labels.

3. Compute the mode among those k values, the result will be the classifier's prediction.

A brief analysis of the used methodology will be now provided. After having uploaded the data in Matlab, the function which creates the classifier was built: it is noteworthy the fact that this function takes as parameters:

1. Training set matrix
2. Test set matrix
3. Nearest neighbors' number to evaluate
4. Ground truth label vector

To ensure the consistency of inputs parameters, a dimensional check was added. Regarding the classification, the function computes for each element of the test set, the Euclidean distance from each element of the training set; is quite challenging computational-wise because it requires to iterate multiple times along the dataset: it's necessary to compute 60'000 distances for each one of the 10'000 elements of the test set, given a total number of iteration of 600'000'000. Then the k smaller value was derived, to be able to predict the class label. A new check for the fourth parameters is needed at this point, and, in the case, it is required it computes the classifier's error rate by comparing the inferred classification with the ground truth vector and returns this value along with the prediction vector.

3.2.2 Testing of the classifier

The implemented classifier was tested in two different ways:

1. on 10 tasks: each digit vs the remaining 9 (i.e., recognize whether the observation is a 1 or not; recognize whether it is a 2 or not; ...; recognize whether it is a 0 or not)
2. for several values of k , e.g., $k=1,2,3,4,5,10,15,20,30,40,50$ (you can use these, or a subset of these, or add more numbers, and you can also implement the rule: "k should not be divisible by the number of classes," to avoid ties)

The recommended assignment guidelines require to evaluate the distance each time a new sample is made. I have implemented only two scripts for the entire testing phase, to avoid redundancy in the code; the two programs are identical to the *kNN function*, an exception for the fact that they compute the matrix of distances using *pdist2* only once while evaluating the error rates for several k values, extracted using the *mink* Matlab function.

3.3 Conclusions

In this section it possible to analyze the accuracy (defined as the number of correct classification over the total number of samples) of each digit vs the remaining nine (the first test) 3.1. From these results, it is possible to retrieve the fact that while increasing the value of k the accuracy decreases, since too many neighbors are taken into account, and values that are different from the one in the evaluation are considered. On the other hand, if k is too small this can leads to inaccuracies and errors when considering an observation in which the proximity assumption of similar elements does not hold. The last figure provided the result for the second test 3.2, where the accuracy drops down as k increases. Finally, it is possible to say that the main drawback of using a *kNN classifier* is that the evaluations become slower as the volume of data increases. This happens because of the absence of a real training phase during which the model is developed, so it is not convenient in cases where a fast prediction is required. On the

other hand, this lack can be considered an advantage since there is no need to tune several parameters. In conclusion, it is possible to affirm that the kNN classifier is mostly useful to solving problems with solutions that depend on identifying similar objects [4].

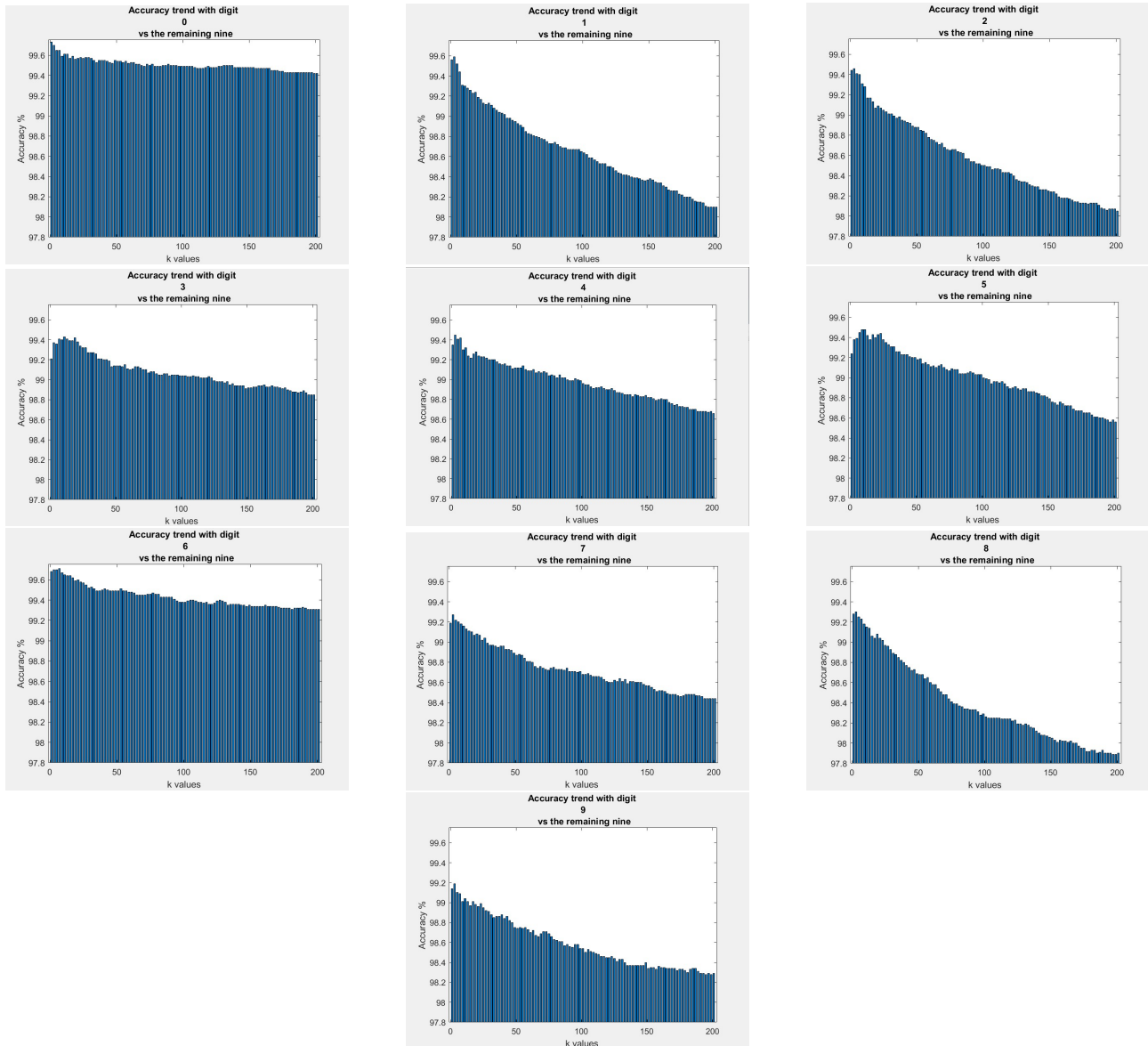


Figure 3.1: Test 1

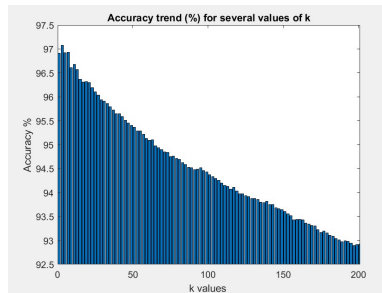


Figure 3.2: Test 2

Assignment 4a: Single-unit Neural Networks

Please notice that unfortunately the delivered assignment was not 100% completed. This is why the *Result* section is missing in this report.

Abstract

Neural networks, are a branch of machine learning and can be defined as the core of deep learning algorithms. As the name suggests, they have an architecture inspired by the human brain structure, in fact, they try to emulate how neuronal cells communicate through each other. The structure at the base is made up of several layers including nodes; each node, which can be identified as a single neuron, is connected to others. Based on various parameters that determine its activation, each node may or may not be able to communicate with those connected to it, to make information flow. Neural networks rely on training data to learn and improve their accuracy over time. In this fourth assignment, the goal is to develop and test two single-unit neural network called Perceptron and Adaline.

4.1 Introduction

For this laboratory was requested to implement two Matlab functions, *Perceptron* and *Adaline* which take as arguments:

1. a set of n data, as a $n \times (d+1)$ matrix, observations in rows
2. a scalar h with a positive value (mandatory) smaller than 1 (suggested)
3. an integer scalar k

The two functions should perform the following tasks:

- Split the data according to k
- Perform training and test, building a confusion matrix or an average confusion matrix
- At the end, output the confusion matrix as the return value of the function

Moreover, a testing part was expected, using the *Iris* and *MNIST* datasets. The overall characteristics of *Perceptron* and *Adaline* can be easily found in many websites, scientific articles,

and books, so I will avoid describing them, focusing on the main characteristic that differentiates them. *Adaline* and *Perceptron* are both linear classifiers when dealing with individual units. They both take an input and based on a threshold, output e.g. either a 0 or a 1. The biggest difference between the two is that the former utilizes a continuous response value to update the weights, whereas the latter takes that binary response and computes an error used to update the weights [5]. The fact that the Adaline does this, allows its updates to be more representative of the actual error, before it is thresholded, which in turn allows a model to converge more quickly.

4.2 Experiments

Firstly, the required datasets were uploaded in a Matlab script: since it was not a unique one, at the beginning of the code it is needed to select which dataset is going to be used. As above-mentioned in the *Introduction* section, the chosen database should be passed to one of the two functions, along-with a k parameter. Specifically, this parameter is necessary to determine the splitting methodology of the dataset. In fact:

- $k = 2$: splits into one training set and one test set of equal size
- $k = n$: splits into n training sets in the n possible ways (perform leave-one-out cross-validation)
- $2 \leq k \leq n$: performs k -fold cross-validation

Following these guidelines, the learning algorithms are applied once for each sample, considered the unique test set, and using the remaining ones as a training set.

Perceptron function

The implemented function takes as input the required dataset, previously modified, and the parameters h and k . The attention is focused on reducing the *Computational complexity* of the algorithm; it is possible to decrease it considering the mean absolute error. A stopping point is needed to block the computations, otherwise, the algorithm will continue to infinity. A certain error should be chosen as a stopping point for evaluations. This control is performed when a training set search is completed, along with a check on the iterations number. The function produces as output the training set, the test set, the optimal value, and the iterations number.

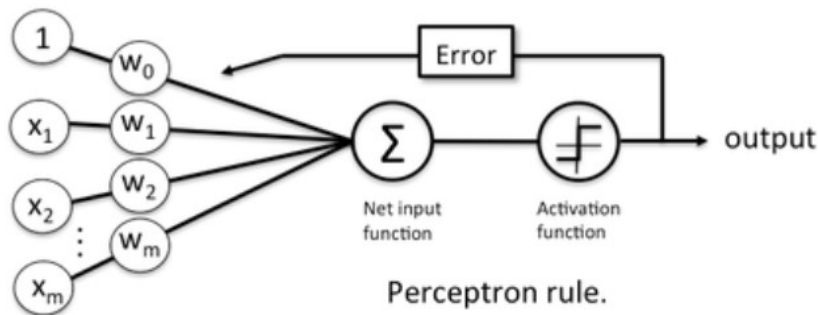


Figure 4.1: Perceptron Rule

Adaline function

As above-mentioned, the *Adaline algorithm* has many common aspects with the previous one, as it is also possible to see from the code. The main difference regards the stopping condition: *Adaline algorithm* will train until the updated weight becomes a small length (Euclidean Norm), and will not be associated anymore with the error rate and accuracy.

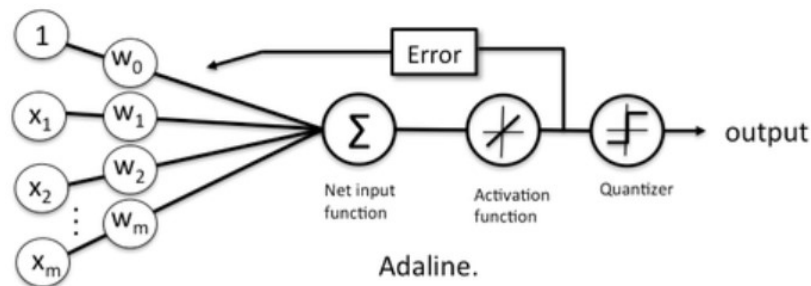


Figure 4.2: Adaline Rule

Assignment 4b: Neural Networks

Abstract

Neural network (NN) models are inspired by the biological nervous system. The most basic elements are neurons, which all act at the same time to accomplish a specific task, including pattern recognition, identification, classification, speech, vision, and control systems.

This assignment aims to become acquainted with these learning models' structure and understand the ideal set-up to obtain performing results.

5.1 Contents and Results

Neural Networks are made of elements (neurons) that are trained to perform a particular function by adjusting the internal neuron weights. Typically, neural networks are adjusted, or trained, so that a particular input leads to the specific target output [6]. A first introduction to Neural Networks via Matlab is provided. The steps to create a complete NN are: Collect Data, Network Creation, Network Configuration, Weight and Biases Initialization, Network Training, Network Validation, Using the Network.

Following the Data Fit Tutorial at [Data Fit](#) we first get acquainted with the MATLAB tools. A data fitting neural network of 20 hidden neurons, with 70% of the data used as testing, 20% as validation, and 10% as testing and the Levenberg-Marquardt algorithm was used. Its result and structures are here shown, where we can see the particular and overall fits. The tutorial is very insightful as it shows how it is possible to plot *Histogram Errors*, *Regression Results*, *Network Training State* and *Network Performance*. Furthermore, it also shows how it is possible to generate code, function, and Simulink models from the easy-to-use GUI that MATLAB provides. Lastly, it is possible to save the results to analyze later. One remark is that the data has to be input and properly organized with the same amounts of rows and columns, according to the convention.

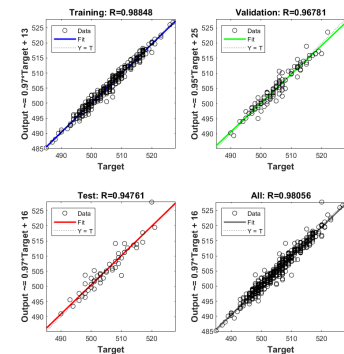


Figure 5.1: Fitting Results

Then, the classification pattern tutorial was followed at [Pattern Classification](#). Here it is furthermore possible to plot *Confusion Matrices* and *Receiver Operating Characteristics*.

First, the *Wine Vintage* provided by Matlab was used, with a NN of 100 neurons. Using 70%

of the data as training, 20% for validation, and 10% for testing, the following results were obtained:

Training Confusion Matrix				
Output Class	1	2	3	
1	38 30.6%	0 0.0%	0 0.0%	100% 0.0%
2	0 0.0%	51 41.1%	0 0.0%	100% 0.0%
3	0 0.0%	0 0.0%	35 28.2%	100% 0.0%
	100% 0.0%	100% 0.0%	100% 0.0%	100% 0.0%
Validation Confusion Matrix				
Output Class	1	2	3	
1	12 33.3%	1 2.8%	0 0.0%	92.3% 7.7%
2	0 0.0%	13 38.1%	0 0.0%	100% 0.0%
3	0 0.0%	0 0.0%	10 27.8%	100% 0.0%
	100% 0.0%	92.9% 7.1%	100% 0.0%	97.2% 2.8%
Test Confusion Matrix				
Output Class	1	2	3	
1	9 50.0%	0 0.0%	0 0.0%	100% 0.0%
2	0 0.0%	6 33.3%	0 0.0%	100% 0.0%
3	0 0.0%	0 0.0%	3 16.7%	100% 0.0%
	100% 0.0%	100% 0.0%	100% 0.0%	100% 0.0%
All Confusion Matrix				
Output Class	1	2	3	
1	59 33.1%	1 0.6%	0 0.0%	98.3% 1.7%
2	0 0.0%	70 39.3%	0 0.0%	100% 0.0%
3	0 0.0%	0 0.0%	48 27.0%	100% 0.0%
	100% 0.0%	98.6% 1.4%	100% 0.0%	99.4% 0.6%

Figure 5.2: Vintage Wine Dataset Confusion Matrix

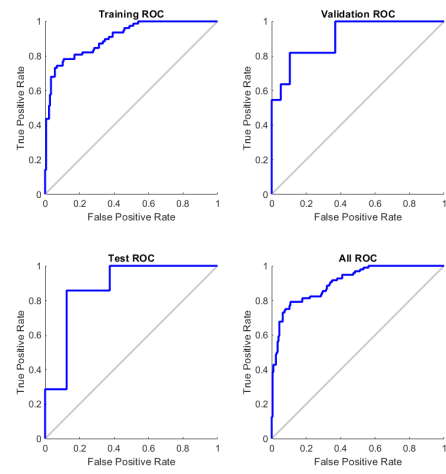


Figure 5.3: Vintage Wine Dataset Receiver Operating Characteristic

Another dataset on heart failure was also used (retrieved at:). By using 12 characteristics and 299 measurement instances (after a minimal data manipulation) it is possible to predict whether a patient would die (1) or survive (0). The 12 characteristics are used as inputs, while the death/survive data vector is used as output. The Neural Network was built with 85% of the data for training, 10% for validation and 5% for testing. Furthermore, the number of neurons was changed, obtaining the following results:

- 10 hidden neurons - 87% overall accuracy.
- 100 hidden neurons - 86% overall accuracy.
- 1000 hidden neurons - 84.9% overall accuracy.
- 1000 hidden neurons - 83.3% overall accuracy.

The results for the 10 neuron pattern classification follow.

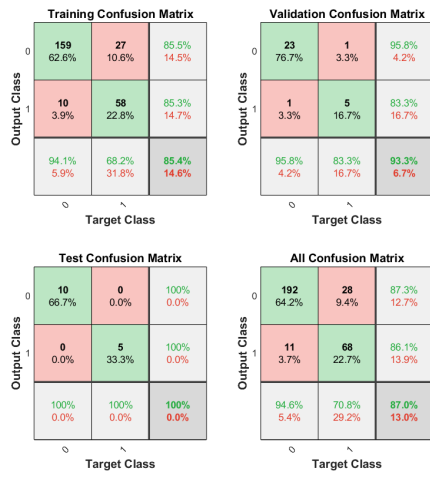


Figure 5.4: Heart Failure Dataset Confusion Matrix

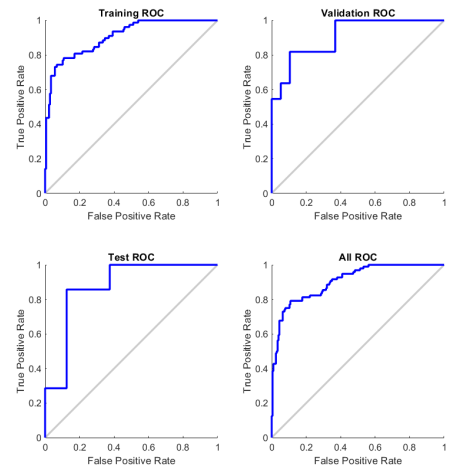


Figure 5.5: Heart Failure Dataset Receiver Operating Characteristic

Assignment 5a

Abstract

Autoencoders are a specific type of unsupervised artificial neural networks, where the input is the same as the output. They compress the input into a lower-dimensional code and then reconstruct the output from this representation. The result is a code which is a compact summary of the input. This assignment aims to train an autoencoder using the MNIST dataset on MATLAB software, to verify the capacity of the network to learn various classes.

6.1 Introduction

Three components are needed to define an autoencoder, which are encoder, code, and decoder. At the beginning of the process, the encoder is in charge of compress the inputs, producing the code. Then, the decoder reassembles the input only using this code[7].

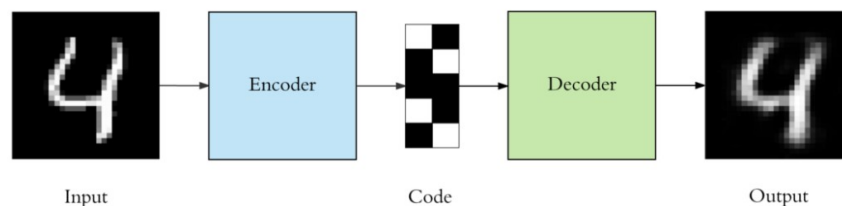


Figure 6.1: Autoencoder Scheme

Properties described below represent the main peculiarity of autoencoders:

- **Data-specific:** autoencoders are only able to squash data similar to what they have been trained on
- **Lossy:** autoencoder outputs will not be exactly the same as the input but they will be a degraded version of the input one.
- **Unsupervised:** autoencoders are considered an unsupervised learning technique since they don't need explicit labels to train on

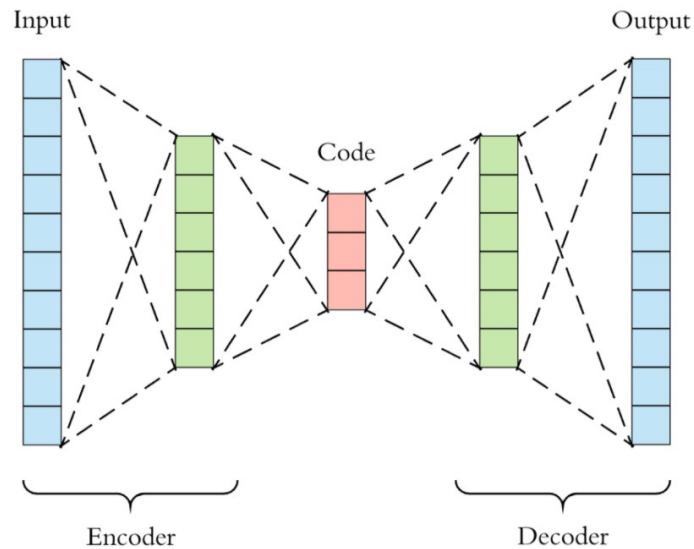


Figure 6.2: Autoencoder Architecture

6.2 Experiments

For this laboratory it was required to perform the following tasks:

1. Split the data into subsets of different classes x_1, x_2, \dots, x_{10}
2. Create a training set with only 2 classes and experiment with different combinations
3. Train an autoencoder on the new, reduced training set
4. Encode the different classes using the encoder obtained
5. Plot the data using the provided "plotcl" function
- 6.

The first part of the code is dedicated to loading the *MNIST* dataset into a Matlab script. It is a noteworthy fact that for each example, 784 features are present since they represent the value of every single pixel of a 28x28 image. After this, a training set was created using the two subsets just provided and applying the suggested function *trainAutoencoder* to train an autoencoder on the new, reduced training set. Then, different classes were encoded using the *encode* function and finally, thanks to the *plotcl* function, a graph is displayed to compare the encoding similarities between two digits through a geometrical distance.

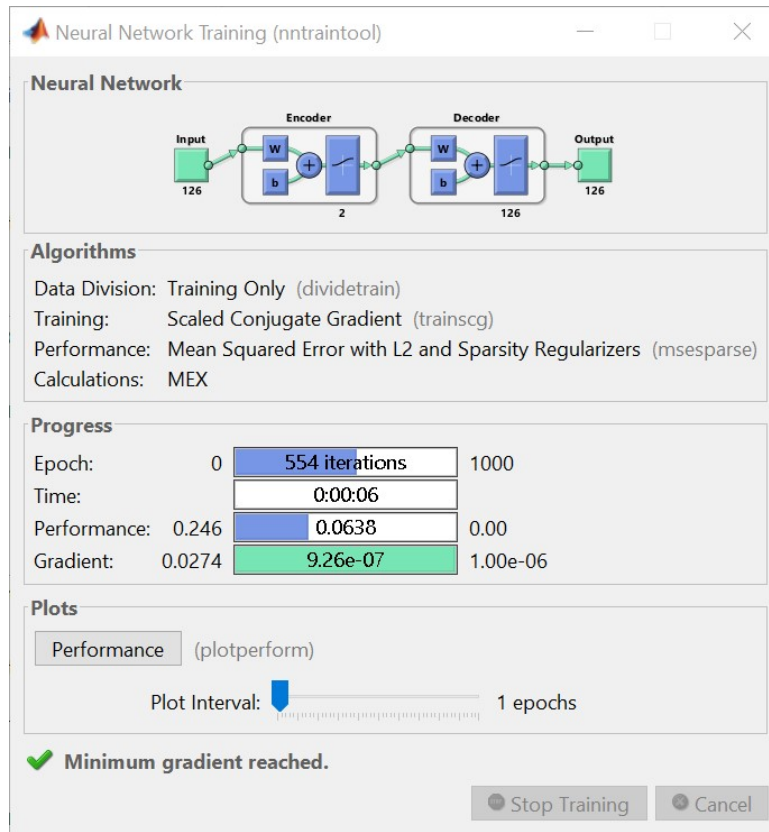


Figure 6.3: Neural Network Training in Matlab

6.3 Conclusions

In conclusion, it is possible to affirm that the forecast result was achieved: in fact, it was expected that similar patterns will have similar representations. As it is possible to observe in the following images, each point represented with a particular color and style is an output of one of the two neurons of the hidden layer, and so a different digit. From the figures, it is noticeable that, on one hand, if there is a high discrepancy in the shape of two digits, (e.g. 2 and 7, 1 and 8, 2 and 9) 6.5, the distance between the two points is greater than $0.7 * 10^{-3}$, and so, they will be linearly separable. On the other hand, if there is a similarity in the shape of two digits, (e.g. 1 and 7, 6 and 5, 8 and 9) 6.4, the distance between the two points is less than $0.3 * 10^{-3}$, leading to a more difficult linear separation.

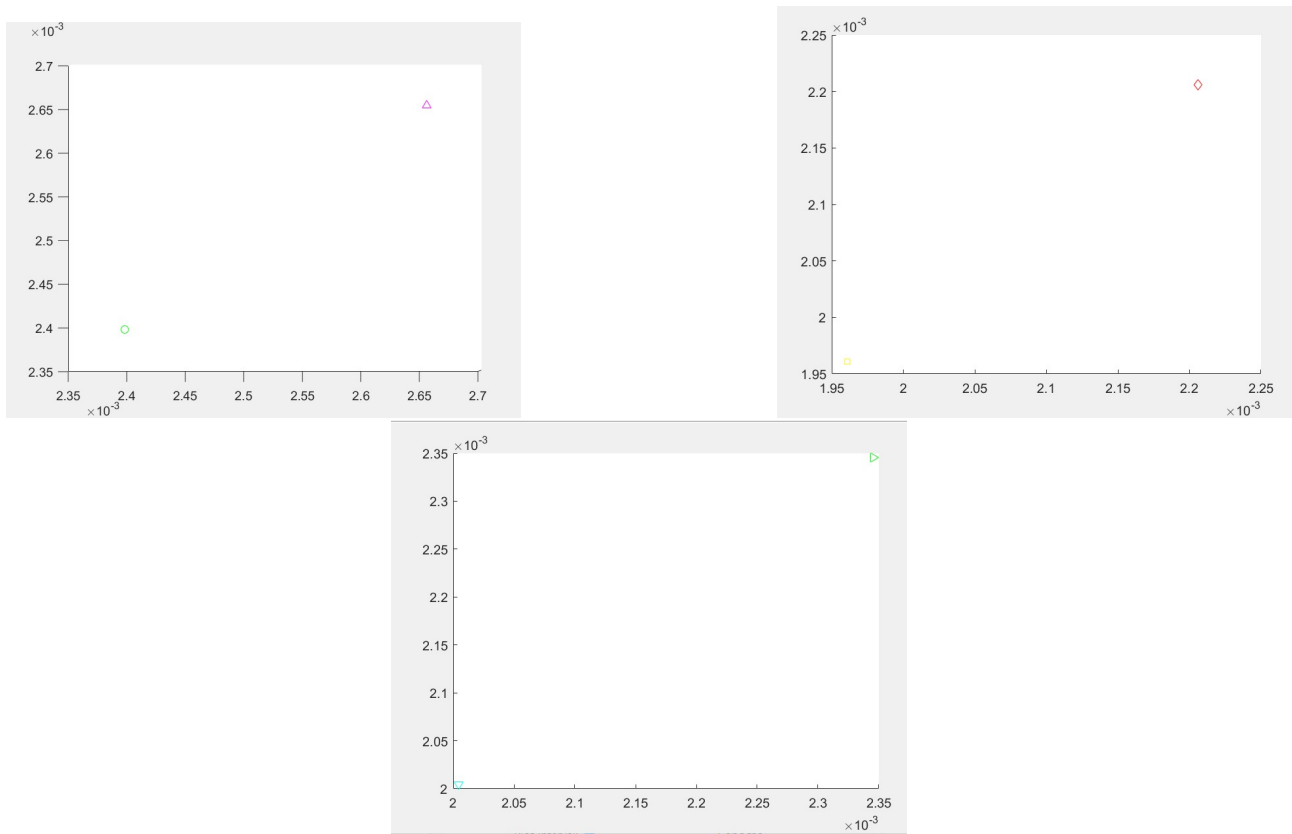


Figure 6.4: Similar Digits

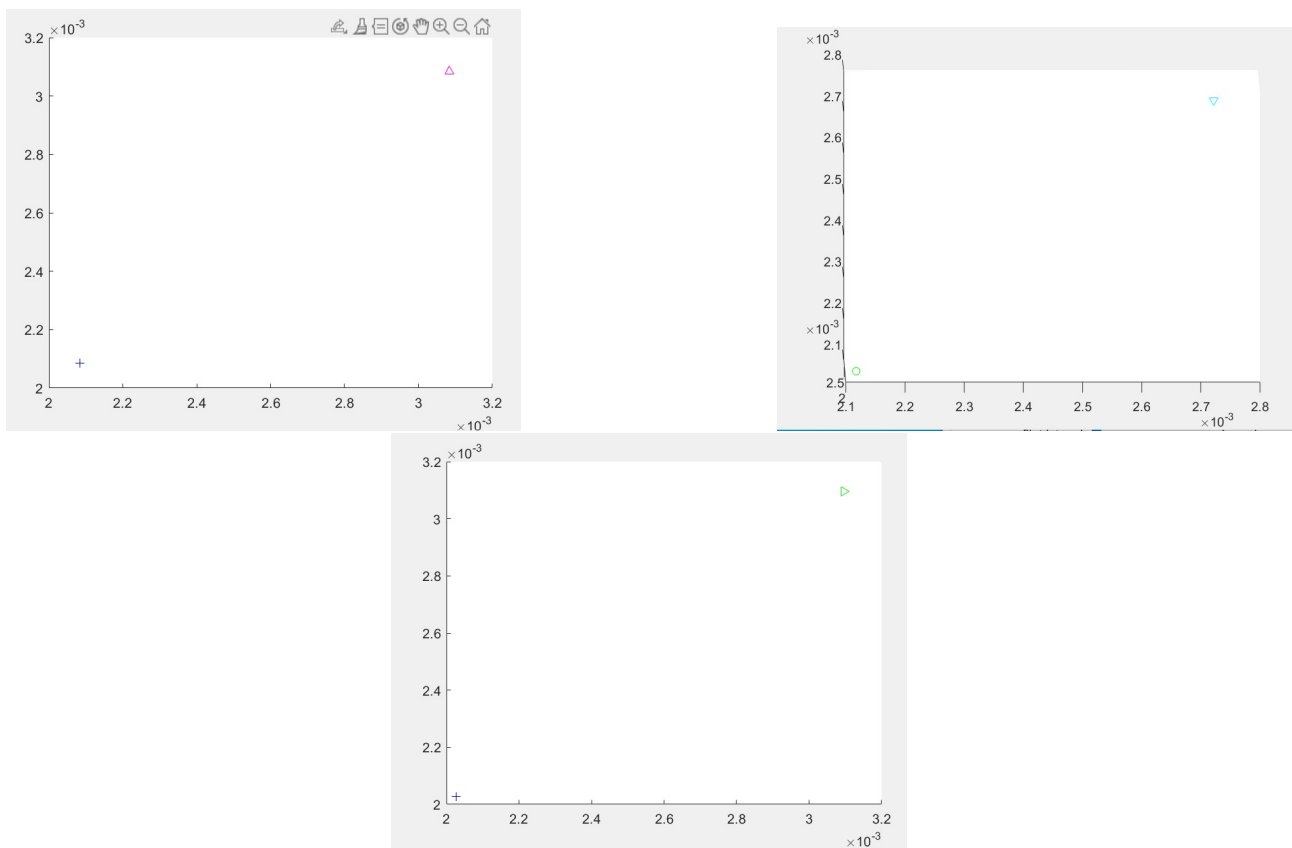


Figure 6.5: Different Digits

Assignment 5b: Practice with Deep Learning

Abstract

The Assignment aims to familiarize with Deep Learning, especially with Deep Neural Networks. Their approaches are based on pre-trained neural networks, which brings some advantages since the Neural Networks training from scratch could be avoided, saving computational efforts. Furthermore, NNs are usually more reliable, being continuously tested and updated by users and developers.

7.1 Contents and Results

The goal of this laboratory is to get acquainted with Deep Learning and Convolutional Neural Networks, using MATLAB's large suite of tools and examples. The first example ([Deep Learning in 10 lines](#)) shows us how with just a few lines of code is possible to recognize a mouse (see Figure) by using *AlexNet*, a pre-trained network that can recognize up to 1000 objects. Next, a thorough description, classification of pre-trained Deep Neural Networks, and their main characteristics are provided at [Pretrained CNNs](#). Using pre-trained networks saves a lot of time and is more effective for tasks such as *Classification*, *Extract Features*, *Transfer Learning*. Pretrained networks vary according to speed, accuracy, and size; the article shows how various networks perform, describing how, in general, networks are benchmarked against the publicly available **ImageNet** dataset, a reference point in image classification. Lastly, the article describes what are the main networks available on MATLAB, what they are used for, how to re-use and improve them, and how to import and interface networks with external models (*Keras*, *Caffe*, *ONNX*).

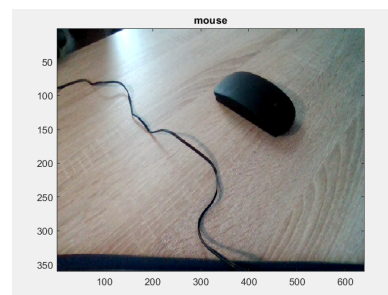


Figure 7.1: Mouse Recognized Using AlexNet

7.1.1 Image Classification

Following the tutorial at [Image Classification](#), it is possible to use the (pre-trained) *GoogLeNet* network to classify images. The classes available in this case are: 'papillon', 'eggnog', 'jackfruit', 'castle', 'sleeping bag', 'redshank', 'Band-Aid', 'wok', 'seat belt', 'orange'. Then, we load an image and resize it to fit *GoogLeNet*'s size of 244x244x3 and correctly classify a bell pepper.

Finally, by running the last instance of code, it's possible to retrieve the top 5 predictions of the algorithm. The results from the example follow.



Figure 7.2: Bell Pepper Classification

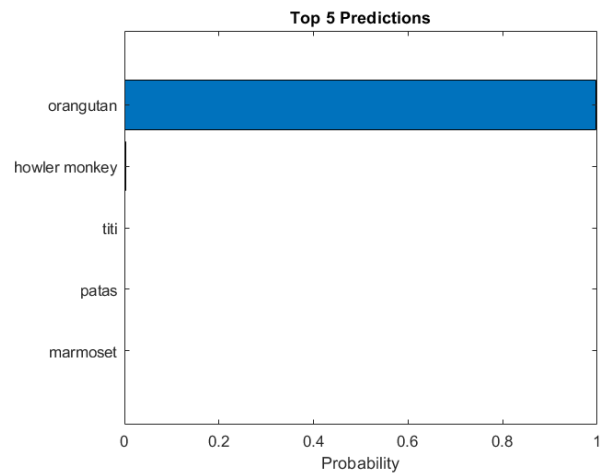


Figure 7.3: top 5 Bell Pepper Classification

By using other networks, it is possible to see the different classes they possess, for instance *squeezenet* has slug, suspension bride, orangutan or cairn. By repeating the procedure described in the tutorial, it is possible to correctly recognize an *orangutan* by using *squeezenet* (and resizing the image), as it is possible to see:



Figure 7.4: Orangutan Classification

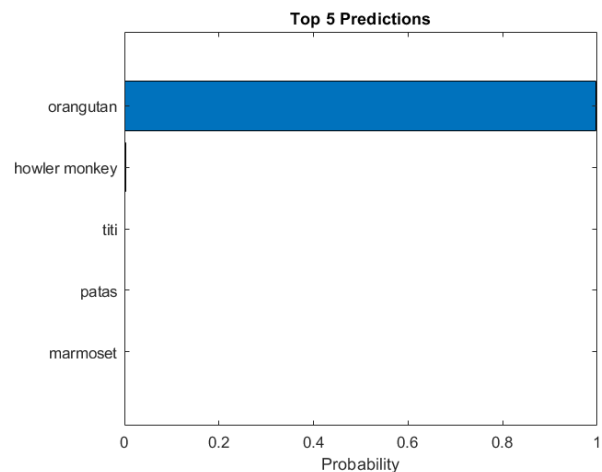


Figure 7.5: Top 5 Orangutan Classification

7.1.2 Image Feature Extraction

The next tutorial at [Featyre Extraction](#) shows how to extract image features from a pre-trained CNN. Here we use *resnet18*, a CNN with 244x244x3 dimension. Then, by using the *analyzeNetwork()* function, it's possible to see the overall structure and layers of the network. Here, we load a set of MATLAB merch images and get feature representation of the training and test images at the global pooling layer, *pool5*. Then, the tutorial shows how to classify the images (see Figure) and does so with an accuracy of 95% in this case. Shallower features can also be extracted and used to train a classifier; these features are earlier in the network, and in this case, we use the *res3b_relu* layer, that uses a ReLu function, and that can be retrieved via the *analyzeNetwork()* function in the network. Since we are earlier in the network, we need to average the activations in the spatial locations, and then train a Support Vector Machine, that yields again a 95% accuracy.

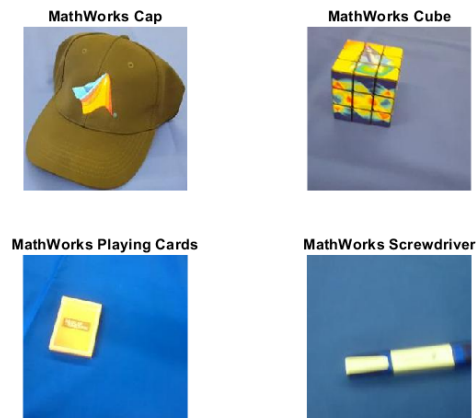


Figure 7.6: Feature Extraction Example

7.1.3 Deep Learning Training

In this last example [Deep Learning Classification Network](#), a deep NN is trained to recognize new images. As before, MATLAB merch images are loaded and the *GoogLeNet* network is used. However, differently from before, now a pre-trained NN is reused and the final layers are substituted, to suit it to the desired needs. The last 2 layers are substituted to better suit the used dataset: in fact, the tutorial explains how different NN use different final layers, and how it is up to the designer to choose and implement the correct one for the proper classification that needs to be done. In this case, we replace the classification layers with the new classes that we need. Furthermore, layers can be "frozen", i.e. the weights are not updated, to speed up the learning. Then, after a small image processing to improve learning, the dataset is trained, with a 90% accuracy. Lastly, the finely tuned network is used for the classification of the used images. The training results and processes are shown in the following pictures.

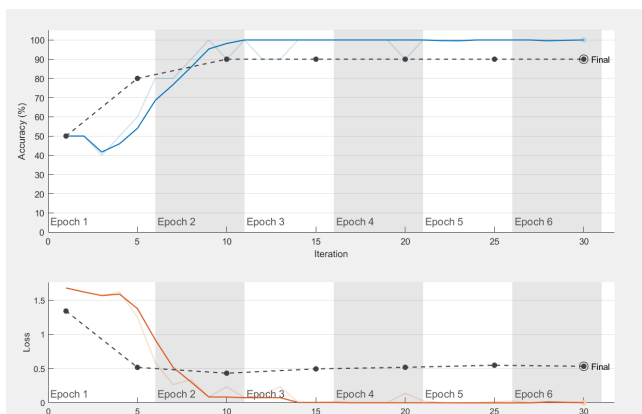


Figure 7.7: Deep Neural Network Training

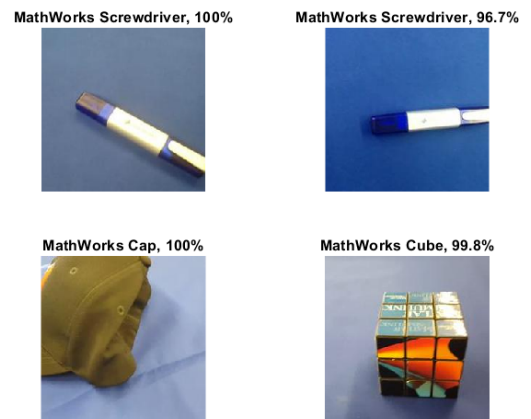


Figure 7.8: Training Results

Bibliography

- [1] S. Mukherjee and N. Sharma, “Intrusion Detection using Naive Bayes Classifier with Feature Reduction,” *Procedia Technology*, vol. 4, pp. 119–128, Jan. 2012. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S2212017312002964>
- [2] C. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. New York: Springer-Verlag, 2006. [Online]. Available: <https://www.springer.com/gp/book/9780387310732>
- [3] “Machine Learning Basics with the K-Nearest Neighbors Algorithm | by Onel Harrison | Towards Data Science.” [Online]. Available: <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>
- [4] Y. Jiang and Z.-H. Zhou, “Editing Training Data for kNN Classifiers with Neural Network Ensemble,” in *Advances in Neural Networks – ISNN 2004*, ser. Lecture Notes in Computer Science, F.-L. Yin, J. Wang, and C. Guo, Eds. Berlin, Heidelberg: Springer, 2004, pp. 356–361.
- [5] “What is the difference between a Perceptron, Adaline, and neural network model?” Jan. 2021. [Online]. Available: <https://sebastianraschka.com/faq/docs/diff-perceptron-adaline-neuralnet.html>
- [6] “Cos’è una rete neurale?” [Online]. Available: <https://it.mathworks.com/discovery/neural-network.html>
- [7] A. Dertat, “Applied Deep Learning - Part 3: Autoencoders,” Oct. 2017. [Online]. Available: <https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798>
- [8] “Try Deep Learning in 10 Lines of MATLAB Code - MATLAB & Simulink - MathWorks Italia.” [Online]. Available: <https://it.mathworks.com/help/deeplearning/gs/try-deep-learning-in-10-lines-of-matlab-code.html>