Report for the Machine Learning Course

Ebru Baglan, *2021-2022 JEMARO Master's Student, UniGe*

***Abstract*—This paper presents the results of the laboratory sessions conducted under Machine Learning for Robotics course which is the part of Robotics Engineering Master Program along with JEMARO program offered by University of Genova. During the semester five laboratory sessions have taken place with the subjects of Naive Bayes classifier, linear regression, kNN classifier, and lastly neural networks. After an introductory part, the laboratory session is presented with some intermediate results, and demanded graphs/metrics.**

I. Lab1: Naive Bayes Classifier

A. Introduction

Classification is a decision problem with where there is no actual decision to take, only the state(class) of the nature is being recognized. The accuracy/error evaluation of a classifier a loss function should be defined for wrong decisions. One example of this is zero-one loss. Zero-one loss corresponds to minimum-error-rate classification and can be defined as

$$\lambda(y|t) = \begin{cases} 0 & y = t \\ 1 & y \neq t \end{cases} \tag{1}$$

When the decision equals the state of nature, it is not penalized, and when a wrong classification is made, it is penalized with one. Notice that all types of error has the same cost, correct classification does not have a cost. In this case the risk equals the expected probability of error R(y(x)—x). Because risk is defined as

$$R(y_i) = \sum_{j=1}^{c} \lambda(y_i|\omega_j)P(\omega_j) \tag{2}$$

When all the wrong decisions have penalty of 1, there are only the probability of the wrong observations left. Notice that this not-differentiating between types of error attitude is not realistic, but when not given the importance of decisions, it can be used, as for in lab case.

A classifier is a rule $y()$ that receives an observation x an outputs a class $w = y(x)$. The Bayes decision criterion states that y should minimize the expected risk. While building Bayes classifier, c number of blocks (decision/state of nature) are defined that compute the risk of every decision. And among them, the one with minimum risk is decided to be the state of nature for given case.

Naive Bayes is one of the most popular classifiers due to its simplicity. The name 'naive' comes from the fact that it makes naive assumption of input variables being all independent of each other as

$$Pr(x_1, ..., x_d|t_i) = Pr(x_1|t_i)Pr(x_2|t_i)...Pr(x_d|t_i) \tag{3}$$

```
#Outlook   Temperature  Humidity  Windy  Play
overcast   hot          high      FALSE  yes
overcast   cool         normal    TRUE   yes
overcast   mild         high      TRUE   yes
overcast   hot          normal    FALSE  yes
rainy      mild         high      FALSE  yes
rainy      cool         normal    FALSE  yes
rainy      cool         normal    TRUE   no
rainy      mild         normal    FALSE  yes
rainy      mild         high      TRUE   no
sunny      hot          high      FALSE  no
sunny      hot          high      TRUE   no
sunny      mild         high      FALSE  no
sunny      cool         normal    FALSE  yes
sunny      mild         normal    TRUE   yes
```

Fig. 1. Weather data set showing the decision of whether to go play outdoor (e.g., tennis) or not given a description of the weather.

To learn $Pr(x_1)$, how often each value of x_k occurs in the class t_i in the training set is calculated as

$$P(x_k = true|t_i) = \frac{numberoftimesx_k = trueinclasst_i}{numberofobservationsinclasst_i} \tag{4}$$

$$P(x_k = true|t_i) = \frac{N_{true,t_i}}{N_{t_i}} \tag{5}$$

And prior probability of classes can be calculated as

$$P(t_i) = \frac{numberofobservationsinclasst_i}{numberofobservationsinthetrainingset} \tag{6}$$

B. Lab Session and Results

For the lab session, weather data set is used which includes the decision of whether to go play outdoor (e.g., tennis) or not given a description of the weather as can be seen in Fig. 1. It has 14 observations, 4 attributes, and 2 classes. Attributes have either 2 or 3 levels. Data set is pre-processed, and a Naive Bayes classifier is built, that has been improved with Laplace (additive) smoothing later on.

Play	Outlook	Temperature	Humidity	Windy
YES	overcast	hot	high	FALSE
	rainy	cold		
	sunny	mild	normal	TRUE
NO	overcast	hot	high	FALSE
	rainy	cold		
	sunny	mild	normal	TRUE

Fig. 2. The illustration of the cell structure to keep probabilities for each target class.

While calculating the probabilities, cell structure of Matlab is highly used to accommodate varying size of matrices. Mainly, the probabilities are kept in a structure similar to the one seen in Fig. 2.

As indicated in lab manual, with such a small data set, some combinations that appear in the test set were not encountered in the training set, so their probability is assumed to be zero. When multiplying many terms, just one zero will make the overall result zero. In general, it may be the case that some values of some attribute do not appear in the training set, however the number of levels are known in advance. An example is binary attributes (e.g., true/false, present/absent...). It is known that attribute x can be either true or false, but in the training set there are only observations with x = false. This does not necessarily mean that the probability of x = true is zero. To deal with this case, a kind of prior information, which is called additive smoothing or Laplace smoothing should be introduced. This can be thought as a correction to Naive-Bayes classifier to deal with zero possibilities. When N experiments performed and value i occurs n_i times. Then, Without smoothing (simple frequency) the probability of observing value i is given by

$$P(x = i) = \frac{n_i}{N} \tag{7}$$

With Laplace smoothing, the probability of observing value i is given by

$$P(x = i) = \frac{n_i + a}{N + av} \tag{8}$$

where v is the number of values of attribute x, and a is a coefficient with $a > 1$ values meaning trusting to prior belief more than the data, and $a < 1$ values meaning trusting data more than prior belief. Changing a values creates a bias towards the class which has more data in training set according to a StackOverFlow question/answer pair [1], in case of an unbalanced dataset.

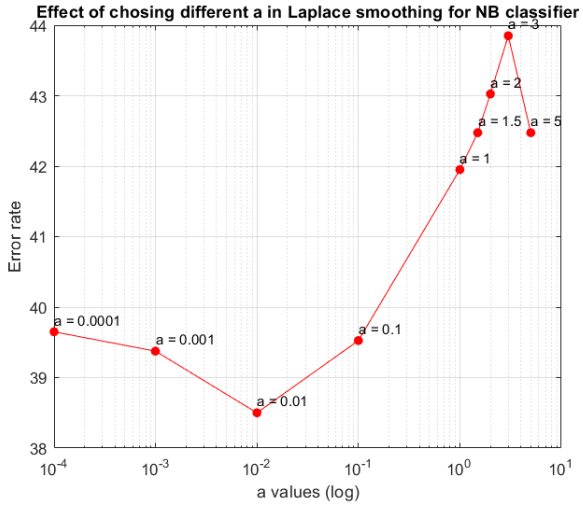

Fig. 3. The effect of choosing a values for Laplace Smoothing for NB classifier.

The effect of the smoothing is investigated with a values in a range of 0.0001, 0.001, 0.01, 0.1, 1, 1.5, 2, 3, 5 and each value is run for 1000 times. The error rate is found to be as in Fig. 3 with an average of 42.5%. Notice that this value is 36.4% without smoothing. So Laplace smoothing causes increase in error value. This can be due to introducing something that was not initially within the data set, however, this correction should be made even for the cost for a small drop in accuracy. From Fig. 3, we can deduce that for this particular dataset of weather data, we should trust data more than our prior belief, and should take $a = 0.01$.

II. Lab2: Linear Regression

A. Introduction

Regression means approximating a functional dependency based on measured data, and is a typical supervised problem. A given data set is divided into two categories, observations (denoted as x_i) and target (denoted as t_i), having the same number of data. A linear model $y(x)$ that predicts t given x with relationship $t = y$ (approximate equation) where $y = wx$. For instance $t_1 = y_1$ where $y_1 = wx_1$ or $t_2 = y_2$ where $y_2 = wx_2$. And we want $y(x)$ to be similar to $t(x)$ for any x.

As clearly seen, w is an average over all data, and we cannot expect this to be good for all data points. What we can expect is to have a value which is 'not so bad for the most of the points'. This leads us to define an error measure to see how good our average value w is. This error measure is called 'loss function', and our aim is to make this as small as possible on average. Loss function is defined as a positive contribution and should also be differentiable (to find maxima/minima). Among possible choices, square error is chosen, as it penalizes deviations more than a linear one, and it produces positive output. Square error loss is defined as $\lambda(y, t) = (y - t)^2$. To define this over the whole data set, we define an objective(or cost) function as

$$J_{MSE} = \frac{1}{N} \sum \lambda(y, t) \tag{9}$$

and for Square Mean Error it becomes,

$$J_{MSE} = \frac{1}{N} \sum (y - t)^2 \tag{10}$$

This can be used over 2 situations. If it is processed on the data set (building a model = training), where the targets (t_1,t_2...t_n) are fixed, the objective only depends on the model parameters. Whereas for a given model (using a model = inference), this can be applied to various data sets, which makes the objective dependant only on the fixed model parameters(data).

The objective is to minimize J_{MSE}, which is known to be a parabola. The unique solution is to find minimum is

$$\frac{d}{dw} J_{MSE} = 0 \tag{11}$$

For one observation it becomes

$$\frac{d}{dw} \lambda_{SE}(t, y) = 2x^2 w - 2xt \tag{12}$$

For overall data set,

$$w = \frac{\sum_{i=1}^{N} x_l t_l}{\sum_{i=1}^{N} x_l^2} \tag{13}$$

Above equation gives the least squares solution to the linear regression problem.

Notice that the current model of $y = wx$ is not very flexible and has only one variable. A more flexible model would be obtained just by introducing an offset, which would turn the model into $y = w_1 x + w_0$. The solution (13) in this case would transform into

$$w_1 = \frac{\sum_{l}^{N} (x_l - \bar{x})(t_l - \bar{t})}{\sum_{l}^{N} (x_l - \bar{x})^2} \tag{14}$$

$$w_0 = \bar{t} - w_1 \bar{x} \tag{15}$$

where w_1 is the slope and w_0 is the intercept or offset. Now it is switched from a linear model to an affine model. A side note, an affine transformation or map means a linear transformation followed by a translation, which describes the case.

B. Lab Session and Results

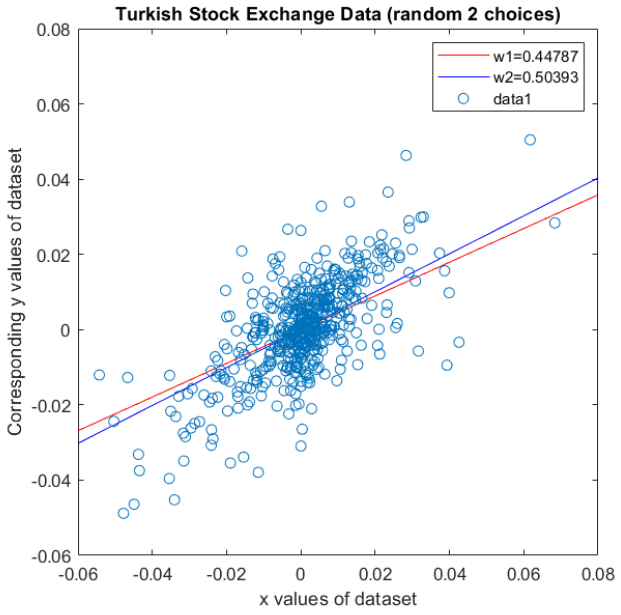

Fig. 4. One-dimensional problem without intercept on the Turkish stock exchange data, with randomly chosen data set.

In the lab session, provided datasets of Turkish stock exchange and Motor Trends car data (32 rows and 4 columns data set) are used. First, the linear regression on stock data without intercept is implemented, which is then repeated for a randomized 10% of data. After this step, a linear regression with

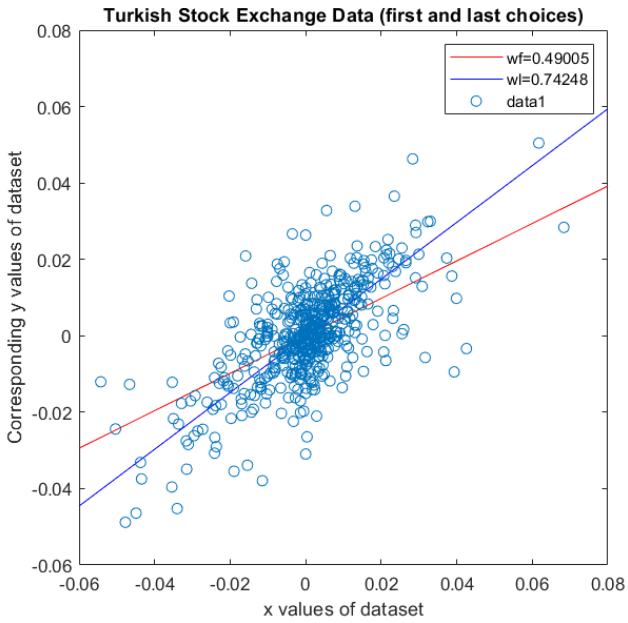

Fig. 5. One-dimensional problem without intercept on the Turkish stock exchange data, with data chosen among first 10 percent and last 10 percent of data set.

intercept is implemented on two columns(mpg and weight) of Motor Cars data set is applied. And finally, the multi-dimensional problem of Motor Cars data set is proceeded for all columns of the set.

For first two parts of the last task, consider the figure with randomly chosen data shown in Fig. 4. As the random choice tends to pick data over the whole data set, the difference of the w values might not be huge. To make differences more visible, it might be a good idea to choose some data from the beginning and the end of the data set (as suggested in the class) as can be seen in Fig. 5.

For third part of the task, the intercept is introduced and is used for mpg and weight columns of Motor Trends data. The result is obtained as seen in Fig. 6.

Finally, multi dimensional linear regression is made for Motor Trends car data set. The only difference from one-dimensional case is to introduce 2 other columns of data set(disp and hp) to fit a line to corresponding mpg values. An offset is also introduced to fit the line. The result can be seen in Fig. 7.

Choosing the test set out of 5% of total data and repeating the regression for 10 run-time resulted in the objective (mean square error) of the values which can be seen through Fig. 8 and Fig. 9. As it is clearly seen, with this very low percentage of training data and then applying this rule on the remaining data set causes huge errors, and should be avoided.

III. Lab3: kNN Classifier

A. Introduction

Statistics is a fundamental part of machine learning. Some even claim machine learning is statistics. In most real-life

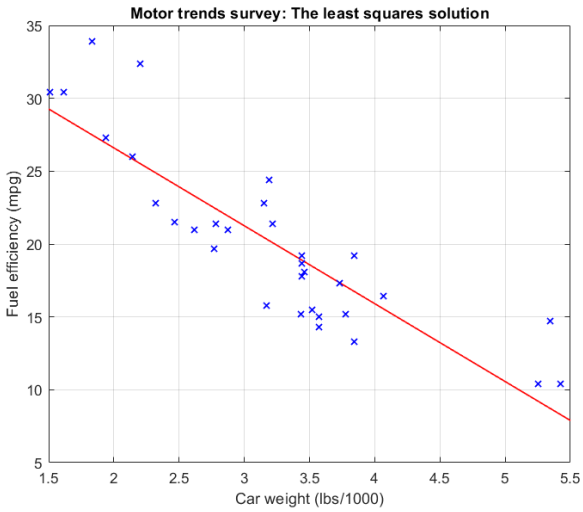

Fig. 6. One-dimensional problem with intercept on the two columns on the Motor Trends car data, mpg and weight.

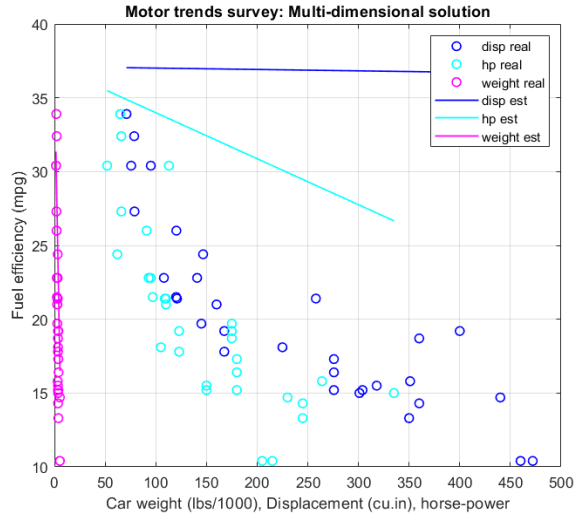

Fig. 7. Motor Trends data with multi-dimensional linear regression.

Run #	J_{MSE} for chosen 5% of data set	J_{MSE} for remaining 95% of data set
1	8.73×10^{-5}	9.02×10^{-5}
2	5.35×10^{-5}	9.07×10^{-5}
3	10.1×10^{-5}	8.96×10^{-5}
4	3.91×10^{-5}	11.2×10^{-5}
5	10.8×10^{-5}	8.78×10^{-5}
6	8.91×10^{-5}	8.91×10^{-5}
7	10.3×10^{-5}	9.64×10^{-5}
8	9.52×10^{-5}	10.2×10^{-5}
9	5.01×10^{-5}	9.72×10^{-5}
10	6.31×10^{-5}	11.1×10^{-5}
Average	7.91×10^{-5}	9.69×10^{-5}

Fig. 8. The effect of choosing different random data sets using 0.05 of data for Turkish stock exchange data.

Run #	J_{MSE} for chosen 5% of data set	J_{MSE} for remaining 95% of data set
1	4.42	147
2	5.56	71.8
3	2.14	65.7
4	4.63	143
5	2.84	60.1
6	0.26	54.6
7	2.07	63.3
8	20.1	50.9
9	0.02	78.7
10	17.5	52.9
Average	5.95	79.9

Fig. 9. The effect of choosing different random data sets using 0.05 of data for Turkish stock exchange data.

scenarios, where probability density function is not available to foresee future outcomes, and only data is available, this limited number of experiences can be used to infer something in general. That is where statistics, the science of inducing from experience, comes in handy to make generalization. Induction is the task of learning a function for every possible input, given only a finite set of example inputs and it creates new knowledge, which is impossible. In this case, learning can take place only if we have an inductive bias (apriori knowledge or assumption). For a classification problem, the bias is that the points in the same class are close to each other, whereas it is that nearby points do not have very different outputs for a regression problem. Clustering assumes the existence of k compact clusters in the data, and lastly it is assumed for a mapping problem that the data are organized in a low-dimensional shape within the d-dimensional data space.

The learning models can be built in two types, parametric models and non-parametric models. A problem in parametric form makes an assumption about probability distributions while the non-parametric form does not. Non-parametric models also do not have a fixed number of complexity (model size, number of parameters) which depend on data. An example of a non-parametric classifier that has been implemented in the lab session is nearest-neighbour(NN) classifier. Due to its simple nature, some call it rather an algorithm. NN classifier does not explicitly implement a model with parameters but directly builds a discrimination rule from data.

The kNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other as in the old saying ”Birds of a feather flock together”. The algorithm works by finding the distances between a query and all the examples in the data, selecting the specified number examples (k) closest to the query, then votes for the most frequent label in the case of classification. It is easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows as in our case [2] The code for the lab gives results for 100 test samples in a few minutes, whereas it had to be processed all night long for it to classify all 10,000 test data. That is why, test data can be windowed to a specific interval of test set by uncommenting some lines in the code.

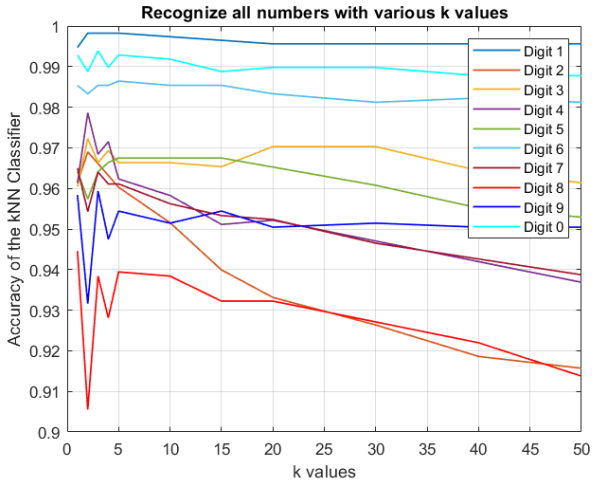

Fig. 10. Recognizing digits with various k values displays 'generally' a decreasing trend with increasing k values.

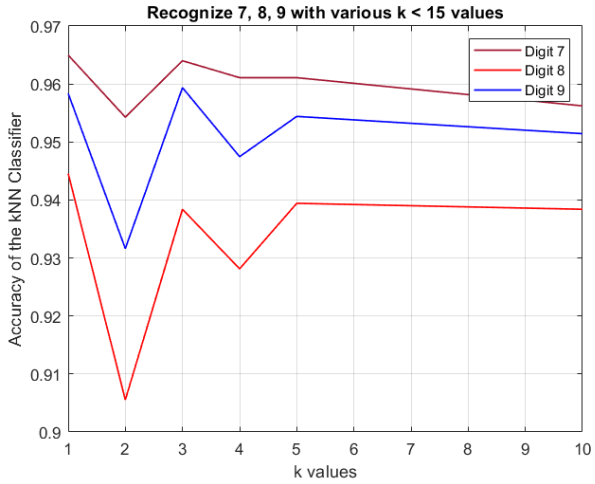

Fig. 11. Recognizing digit '7', '8' and '9' with various k values displays a reduced accuracy for even values compared to adjacent odd values of k.

B. Lab Session and Results

The lab is divided into three tasks as usual, with the first one being data reading, which is followed by the implementation of a kNN classifier, and lastly the testing of it. MNIST data archive has been used, which contains in seperate files a training set of 60,000 samples, its corresponding labels, a test set of 10,000 samples, its corresponding labels, two Matlab functions to load data and labels, and one flexible function to load the data named loadMNIST. The data are in the form of 1x784 row vectors which represent handwritten digits in 28x28 greyscale images, and are a standard benchmark for machine learning tasks.

Several k values have been used as suggested in the Aulaweb page, which are 1, 2, 3, 4, 5, 10, 15, 20, 30, 40, 50 respectively. The accuracy of recognizing each digit with these k values have been calculated and and the accuracy results for recognition of each individual number has been plotted in Fig. 10. As can be seen from the figure, with the increasing k values the accuracy either stayed the same or showed a reducing trend. For identification of 2, 4, 7, 8, however, k values higher than 20 showed a clear recession in accuracy.

Another interesting behaviour has been observed for the recognition of digits 7, 8 and 9 with $k < 15$ values. As can be seen in Fig. 11, these numbers display a similar tendency for same k values. Accuracy for k = 2 and k = 4 are lower than it is for $k = 1$, $k = 3$, and $k = 5$ values. This decrease for even numbers can be due to having equal numbers of neighbours for two different classes, and thereby making a 'majority' decision between equal numbered classes. This random choice can result in decrease in accuracy, that is why odd numbers for k should be preferred over even numbers to prevent this. A close look-up to recognition of digit 8 with 15 and 20 neighbour points give the same result as depicted in Fig. 12.

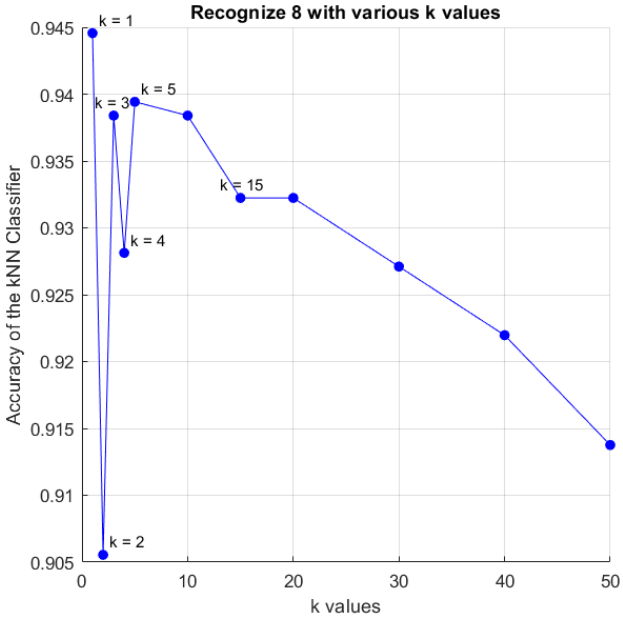

Fig. 12. Recognizing digit '8' with various k values.

IV. Lab4: Neural Networks

A. Introduction

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network. [3]

The Perceptron is a linear machine learning algorithm for binary classification tasks.

It may be considered one of the first and one of the simplest types of artificial neural networks. It is definitely not “deep” learning but is an important building block.

Like logistic regression, it can quickly learn a linear separation in feature space for two-class classification tasks, although unlike logistic regression, it learns using the stochastic gradient descent optimization algorithm and does not predict calibrated probabilities. [4]

Autoencoder is an unsupervised artificial neural network that learns how to efficiently compress and encode data then learns how to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original input as possible. [5]

The simplest autoencoder network is a multi-layer perceptron neural network which has one input layer, one hidden layer (n.hidden units¡n.inputs), and one output layer (n.output units=n.inputs).

It is trained using the same pattern as both the input and the target. So for instance input pattern $x(l,:)$ has target $t(l,:) = x(l,:)$ (the same as the input).

Note that in this case there are not any classes or other mapping to learn. This is a special case of unsupervised training. In fact, it is sometimes called ”self-supervised”, since the target used is the input pattern itself.

B. Lab Session and Results

The lab is as usual divided into three tasks, with the first one getting acquainted with Neural Network Toolbox of Matlab, which is then followed by implementation of feedforward multi-layer networks (multi-layer perceptrons) applied on two datasets, and lastly implementing an autoencoder.

Neural Network Toolbox is used with GUI to work on Matlab’s intrinsic body fat estimation dataset. Using this data set, one can estimate the body fat percentage of a person described by thirteen physical attributes [6], and it has 252 samples. Data set is divided as 70% for training, 15% for validation and 15% for test data and a total of 10 hidden layers is employed. The results can be seen in Fig. 13

For the second part of the lab, feedforward multi-layer networks (multi-layer perceptrons) are asked to implement using Neural Net Pattern Recognition App of Matlab. For this purpose glass identification data set [7] is used. This data set is used to identify a glass to be window glass or not, with given refractory index, chemical composition etc.

Glass data set is trained with 70% of data, 15% validation data and 15% test data, with 10 hidden layers. The results can be seen in Fig. 14. Searching for ideal number of hidden layers, Heaton [8] states that 1 hidden layer can approximate any function that contains a continuous mapping from one finite space to another, and 2 hidden layers can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy. No other numbers are indicated, but still the effects of this metric is wondered and tried with 1, 2, 5 and 10 hidden layers. As can be seen from Fig. 15, even though training and validation errors are lower for 1 hidden layer, test set error is found to be 6.2% percent, whereas this is

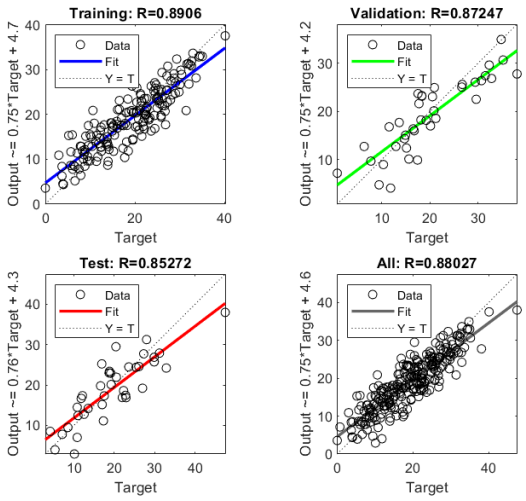

Fig. 13. The result for body fat estimation data set training with 70% of samples, 15% for validation and 15% for test data.

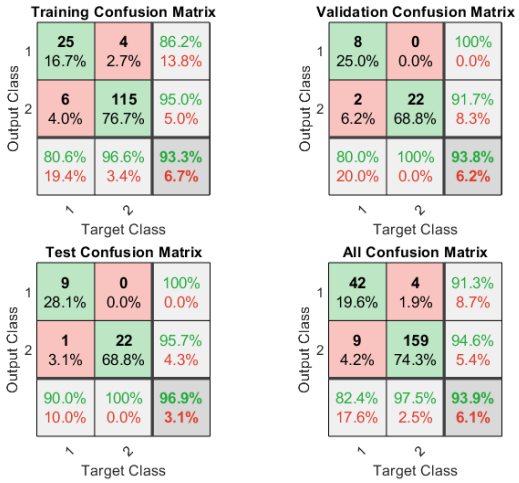

Fig. 14. The confusion matrices for 10 hidden layer for the glass data set.

3.1% for 10 hidden layer case(and it is the lowest among all cases). The procedures are carried out for 2 hidden layers and 5 hidden layers later on, and the results can be seen in Fig. 16 and Fig. 17, respectively. From the results, overall error rate is minimum with 4.7% percent for 1 hidden layer usage, however, it is minimum for test error when using 10 hidden layers.

For the autoencoder part of the lab, a multilayer perceptron as an autoencoder for the MNIST data (of 1000 training sample) is trained, hoping that it will learn the internal, compressed representation for the data and that similar patterns will have similar representations; for instance, we hope that images representing a ”1” will correspond to very similar representations, and quite similar to ”7” but different from ”0” or ”8”. In other words, that the network will learn the classes. As suggested, the classes of 0,1,7 and 8 are used. The encoder is implemented using the given functions of

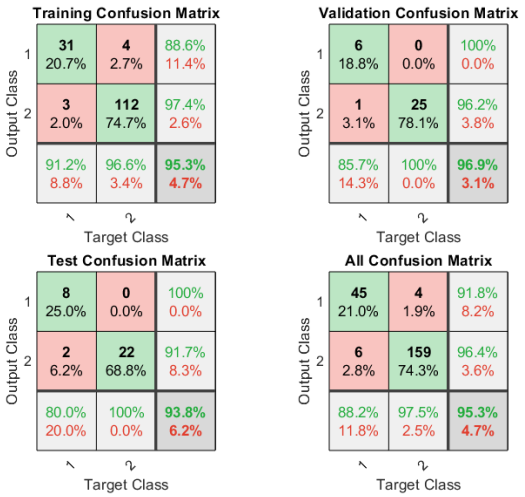

Fig. 15. The confusion matrices for 1 hidden layer for the glass data set.

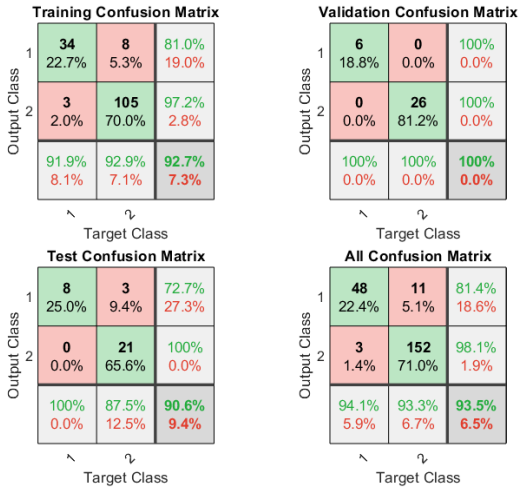

Fig. 16. The confusion matrices for 2 hidden layer for the glass data set.

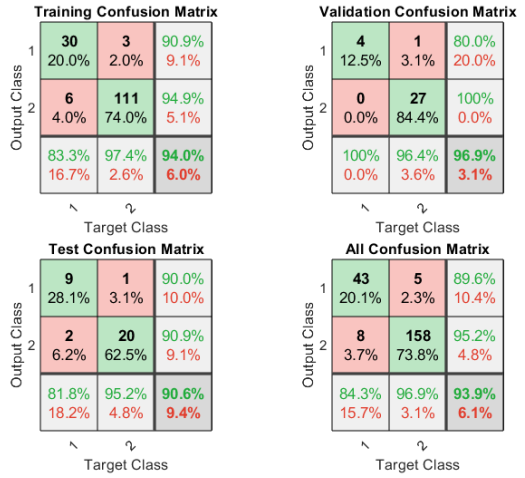

Fig. 17. The confusion matrices for 5 hidden layer for the glass data set.

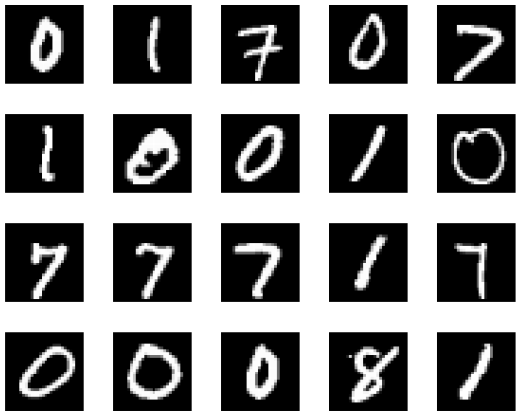

Fig. 18. Randomly chosen handwritten digits from class 0, 1, 7, and 8 of MNIST data set for classification.

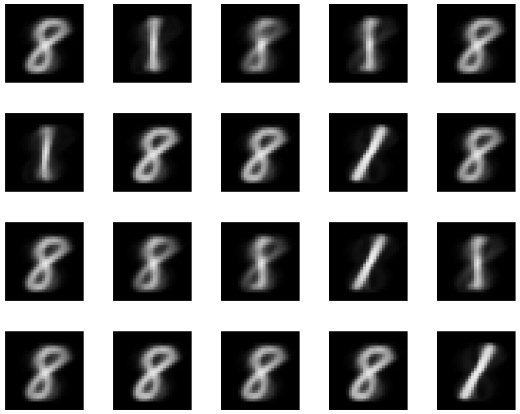

Fig. 19. Outputs for chosen digits of the encoder which has trained with class 0 and 8 of the MNIST data set.

$trainAutoencoder()$ and $encode()$, and the training set only consisted of classes of '1's and '8's. As suggested in class, the hidden layer number is chosen to be 2. Then, this encoder is used to encode a mixed choice of 20 numbers, consisting of 0, 1, 7 and 8, the results are obtained as in the Fig. 18 and Fig. 19.

Using $plotcl$ to represent results gives the Fig. 20, where 1 stands for class of '0's, and 2 is for class of '7'. There is a tendency for both classes to clutter around 0, with class '0' being a little more spread in the graph.

V. Interesting Information

Stratified random sampling is a method of sampling that involves dividing a population into smaller groups–called strata. The groups or strata are organized based on the shared characteristics or attributes of the members in the group. The process of classifying the population into groups is called stratification.

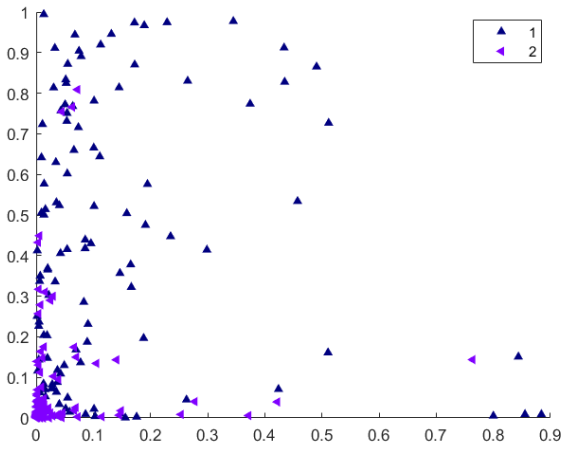

Fig. 20. Outputs for class of '0'(denoted as 1), and class of '7'(denoted as 2) of the encoder which has trained with class 1 and 8 of the MNIST data set.

Stratified random sampling is also known as quota random sampling and proportional random sampling. Stratified random sampling has numerous applications and benefits, such as studying population demographics and life expectancy.

This methodology used in Lab3: kNN Classifier to reduce training set and test set.

REFERENCES

[1] StackOverFlow, "Naive Bayes accuracy increasing in the alpha value," online, accessed 7/12/21, 2021, https://stackoverflow.com/questions/52319703/naive-bayes-accuracy-increasing-as-increasing-in-the-alpha-value.

[2] TowardsDataScience, "k Nearest Neighbour Algorithm," online, accessed 9/12/21, 2021, https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

[3] IBM Cloud, "Neural Networks," online, accessed 27/1/22, 2021, https://www.ibm.com/cloud/learn/neural-networks.

[4] Machine Learning Mastery, "Perceptron Algorithm for Classification in Python," online, accessed 27/01/22, Mar 2011, https://machinelearningmastery.com/perceptron-algorithm-for-classification-in-python/.

[5] TowardsDataScience, "Auto-Encoder," online, accessed 27/01/22, 2021, https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726.

[6] MATLAB, "Body Fat Estimation," online, accessed 22/01/22, 2021, https://www.mathworks.com/help/deeplearning/ug/body-fat-estimation.html.

[7] UCI Center for Machine Learning and Intelligent Systems, "Glass Identification Data Set," online, accessed 22/01/22, 2021, https://archive.ics.uci.edu/ml/datasets/glass+identification.

[8] J. Heaton, *Introduction to Neural Networks with Java*. Heaton Research, Inc, 2008, p. 439.