NTNU

elsys
ELEKTRONISK SYSTEMDESIGN & INNOVASJON

Classification

Classification of Iris Flowers and Handwritten Numbers

Sif Anna Darling Lyken og Live Standal Myklebust

Version: 1.0 | Date: 04.05.22

Innhold

---

1 Summary

This rapport is a two-parted classification project in the course of TTT4275 Estimation, Detection and Classification. The project reflects on and tests different types of classifiers. The projects first task is to classify three species of the Iris, and a Linear Discriminant Classifier is designed for this. With this classifier we manage to determine the flower with an error rate of 66.7%. For the second part we wish to classify handwritten numbers and design an Template Based Classifier. We implement both the Nearest Neighbour algorithm and the more advanced K-Nearest Neighbour algorithm to compare the training to the test data set.

Sadly, testing of the algorithms was not possible due to difficulties with data-fetching. This would be a improvement in future work.

2 Introduction

Classification breaks down broad subjects into smaller more specific part, and is a efficient way of understanding the information we have at hand. The field of classification have been in a large growth, and with the development of powerful technology and efficient implementations, its area of use is grater than ever. By undertaking this project we aspire to get a better understanding of the benefits of the implementation of a classifier, as well as getting insight challenges that it presents. To ensure the proper insight to the tasks at hand, we will firstly introduce the necessary theory. Further, the implementation of the two classifiers and the results will be presented. A discussion of our finds will end in a final conclusion.

3 Theory

In this section we will undertake the theory needed to understand the tasks at hand, as well as establish a basis of argument for our design. Humans are trained to look for traits in their surroundings, distinguish the objects from each other and place them into their appropriate group. By experience and observing we are able to separate women from men, dogs from cats and desserts from dinner. The classification-method used in this project has similarities to this, and is called supervised learning[3]. By feeding the model inputs, the weights are adjusted until a fitted model. The use of classification allows for processing large sets of data, and is a useful tool in machine learning.

3.1 Linear discriminant classifier

A linear discriminant classifier(LDC) uses a linear model for classification and is most commonly used for feature extraction for pattern classification problems. The LDC is detonated by the equation

$$g(x) = w_i^T x + w_i o, \quad i = 1...C.$$

C is the number of classes. For $C > 2$ the equation can be written as $g = Wx + w_0$, where g and w_0 are of dimensions equal to C and W is a CxD matrix(D is number of features). This expression can be rewritten as the $g = [W w_0][x^T 1]^T$. The column of ones added in the latter matrix is needed to obtain the right dimensions.

Now that we got the discriminant vector g we can use it to train our classifier. To do this we need a cost function to optimize the weights W. A popular cost function is the “Minimum Square Error"(MSE). The equation for MSE is

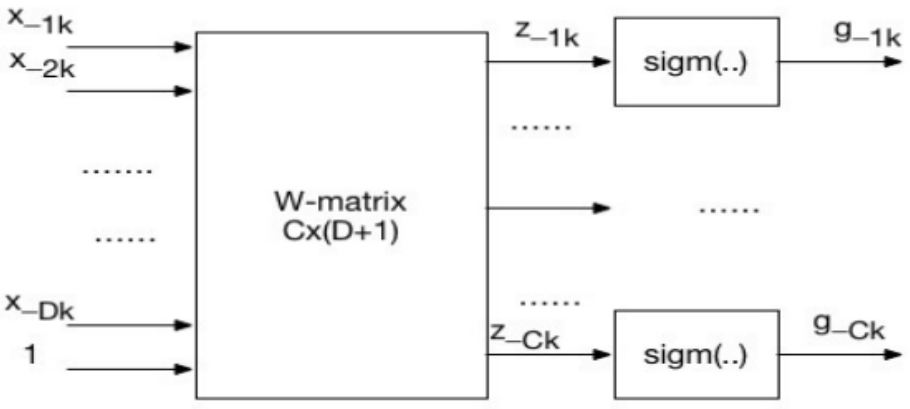
$$MSE = \frac{1}{2}\sum_{k=1}^{N}(g_k - t_k)^T(g_k - t_k), \tag{1}$$

where t_k is the target values in vectorial form. The dimension of the target vector is the same as the number of classes C and consists of binary values. In the case of three classes where the second class is the target, the resulting vector would be $t_k = [0, 1, 0]^T$.

In order to match the output vector g_k to the target vector t_k we need a heavyside function such that g_k gives an output between 0 and 1. An approximation of a heavyside function is the squashing function called a *sigmoid*

$$sigmoid(x_ik) = g(ik) = \frac{1}{(1 - e^{-z_ik})}, \quad i = 1, ..., C. \tag{2}$$

The variable z_ik is the same as g_k used in equation 1. This structure of this type of linear classifier is shown in figure 1.



Figur 1: Sigmoid [3, p.17].

If we rewrite the cost function, i.e. equation 1, with respect to the weights we get

$$\nabla_W MSE = \sum_{k=1}^{N} \nabla_{gk} MSE \nabla_{zk} g_k \nabla_W z_k. \tag{3}$$

By replacing the gradients in equation 3 with the resulting vectors we get

$$\nabla_W MSE = \sum_{k=1}^{N} (g_k - t_k) \circ g_k \circ (1 - g_k) x_k^T \tag{4}$$

Since the goal is to reduce MSE we need to move the weigths in the opposite direction of the the gradient

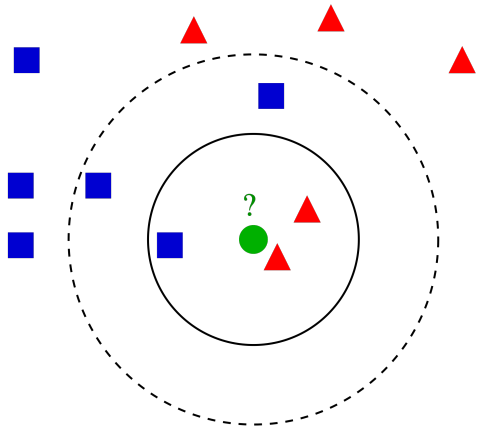
$$W(m) = W(m-1) - \alpha \nabla_W MSE, \tag{5}$$

where m gives the iteration number and α is the step factor. This way we update W throughout the iterations.

3.2 Template based classifier

The Nearest Neighbour - NN algorithm is a template based classifier matching an input with a reference set of the same form. The input is then simply sorted by which reference it is closest to. The K-Nearest neighbour- KNN method holds for a more advanced algorithm. The KNN aims to predict the correct class for the test data by calculating the distance between the test data and all the training points. You find the $K > 1$ nearest references and choose the class whom are reoccurring the most.

An exemplification of a KNN decision ruling is made in figure 2. The green dot represents the test sample, and is to be classified either as a red triangle or as a blue square. Choosing $K = 3$, its nearest references to x are two red and one blue. This would conclude in x being red. However for $K = 5$ x would be classified as blue.



Figur 2: KNN [2].

3.3 Fitting

When it comes to LDCs we have the term *fit*. The fit determines how the classifier performs on training and testing data and is often given as a percentage. If the model has too much specified/detailed training data it can become overly sensitive to testing data. The model will perform good on the training data, but can easily classify unseen data as noise/unknown. This is called *overfitting* the model and will negatively impact the models ability to generalize. The threat of overfitting is not common in linear models, but is worth mentioning. A solution would be to remove some of the detailed training data we can avoid it. As for the case of *underfitting* the model has too little data to train on, resulting in bad performance on both

the training data as well as the testing data, classifying data at random. Best fit depends on how good performance is demanded by the user.

3.4 Euclidean Distance

The Euclidean distance between two points in Euclidean space is given by the length of a line segment between two points. This measurement can be calculated by the use of Pythagorean theorem and the Cartesian coordinates of respective points [1]. The calculation is shown in equation 6.

$$d(x, ref_{ik}) = \sqrt{\sum_{k=1}^{N}(x - \mu_{ik})^2}. \tag{6}$$

An advantage presented by the Euclidean distance is that the parameters $ref_{ik} = \mu_{ik}$ can be chosen directly from a training set. Concerning the NN-algorithms, this is one of the most used ways of locating the references.

3.5 Clustering

Clustering collects and divides objects on the basis of similarities. The task is to separate data points within groups of similar traits into assigned clusters. There are different methods of clustering and one significant is the partitioning method. The data points are separated into K-clusters which are without overlap. The K-means method is a iterative clustering algorithm aiming to compute the best centroid in each iteration. The algorithm requires specified number of clusters K, and then randomly assign each data point to one of the clusters. The cluster centroids need to be computed and a re-assignment of each point to the closest cluster centroid. The cluster centroid then again has to be re-computed. This calibration should be repeated until the positions of the centroid remains unchanged. This process establish a K cluster based on similarities.

3.6 Confusion Matrix

The confusion matrix is a table that shows us the performance of the classifier. This is done by comparing known true values to predicted values. The confusion matrix are often used for the test data set, but can also be applied to the training data. The representation is implemented such that the rows shows the true classes and the columns shows the predicted classes. Inside the table we have numbers which tells us if the data has been classified correctly or wrongly. From this we can calculate the total error rate,

$$ERR_T = 100 \cdot \frac{sum\ of\ wrongly\ classifed\ data}{sum\ of\ all\ data}. \tag{7}$$

4 The task

When deciding on what tasks to focus on it was possible to choose from multiple projects in the categories estimation, detection and classification. The estimation projects deals with the implementation of estimators such as maximum likelihood estimator and best linear unbiased estimator. The detection project addresses spectrum sensing in OFDM based wireless communication systems. However the tasks for this rapport is based in the classification category. In this category there were projects concerning the design of classifiers dealing with the classification of vowels, music genre, flowers or digits.

This rapport addresses the flowers and digits classification tasks. These tasks seemed interesting as they give an introduction to the basics behind machine learning by using classification. Both of the tasks uses classifiers to assign a class label to a data input.

The flower task deals with data containing features of three different types of iris flowers, called Setosa, Versicolor and Virginica. Each type of iris flower have the following four features: sepal length(cm), sepal width(cm), petal length(cm) and petal width(cm). The data we were given contained were given separated into three data set, each containing 50 examples of each class. We want to use these features to create a linear discriminant classifier REF.

The second task, concerning the digit classifier, uses a database with pictures of handwritten numbers from 0 to 9. The database consists of 60000 training examples written by 250 people, and 10000 test examples written by another 250 people. To predict what number is written we will design a NN-classifier using the Euclidian distance REF. In addition, designs concerning clustering and a KNN-classifier will be implemented and used to compare.

For both tasks we will need to find parameters such as confusion matrixes and error rate. For the digits task the confusion matrix is found by clust

5 Implementation Results

This section will contain a description of the implementation and design of the classifiers. We will first present the choices made to implement the Iris flower classification task and discuss the results it gives. Further we will precede to do the same for the Handwritten number task. Both tasks are written in Python due to its simplicity and the documentation supporting it. When referring to the implementation of the flower and the number tasks, it will respectively be refered to the codes *Iris* and *Digits* which may be found in section [**?**].

5.1 Iris flowers - implementation

The main purpose of the flower task is to study the performance of the LDC when using different sizes of testing and training data. In addition we want to look at the classifier when modeling with fewer features and produce histograms for each feature. To compare the results we look at error rates and confusion matrices.

The first thing we need to do is to split the given data into training and test sets. As mentioned earlier we have three data sets, each containing 50 samples. To begin with the data of each class is split such that the first 30 samples are labeled as training data and the remaining 20 samples as test data. All together the training set contains 90 samples and the test set 60 samples, giving us a split ratio equal to 60/40. The LDC will also reverse the samples by having the 30 last sampled as training set and the 20 first as test set.

Training the linear classifier is done according to the theory mentioned in subsection3.1. To initialize the weights W an array full of zeros with the dimensions CxD (number of classes and features) was implemented. To make the training converge we discovered that 2000 iterations was sufficient and that having $\alpha = 0.01$ produced a small MSE.

To produce the histograms we removed one feature at a time from the training and test set, having the 30 first samples as the training set.

5.2 Iris flowers - results

The results of the LDC implementation are presented by confusion matrices, error rates and histograms.

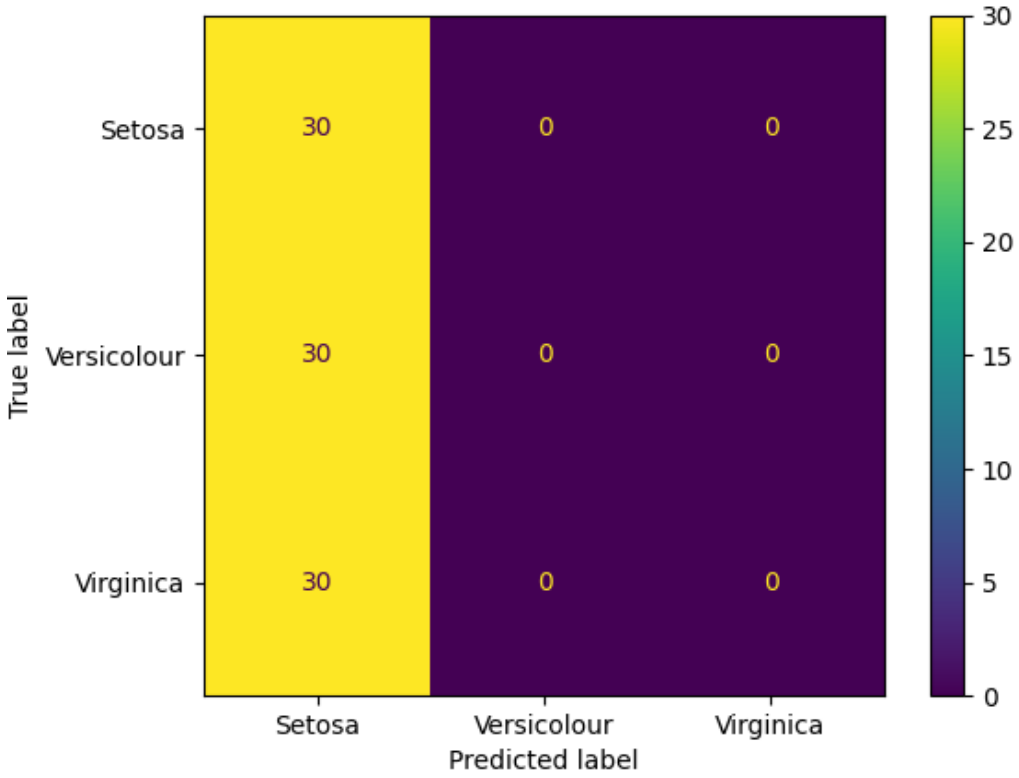
After trying to plot the confusion matrices we obtained unwanted results. As shown in figure 3 and 4 all the samples in the training and test data is classifies as Setosa, giving us an error rate $ERR_T = 66.7\%$. The equations mentioned in subsection 3.1 were all implemented section wise making it hard to test whole the system. We believe that the classification is caused by wrongful implementation of the discriminant vector and the sigmoid function as it gave us strange output numbers.

The histograms presented in figure 5 gives us a visual view of the different features and how they interact. We can observe that features such as petal width and petal length proves to be more useful when trying to classify a iris flower to its respective class. The features sepal width and sepal length has some overlap between the features making it difficult for the classifier to distinguish between the flower classes. Overall it is made clear that the Setosa flower has the most unique features making it the easier one to classify compared to Versicolour and Virginia.

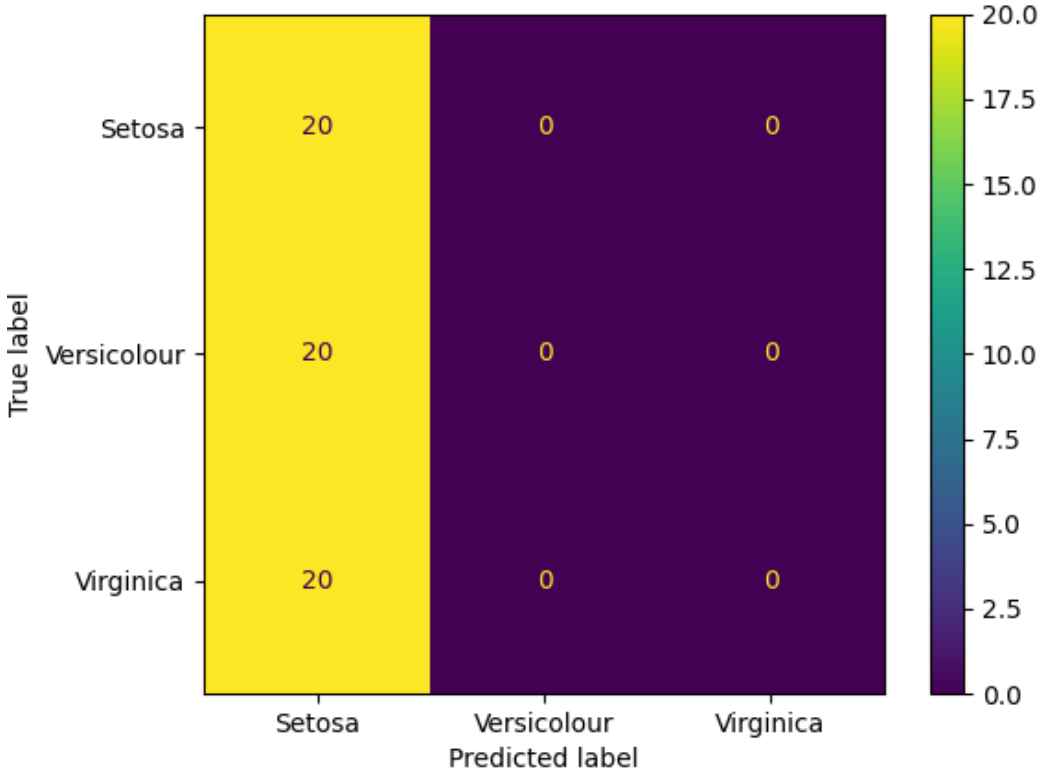
Eventhough we were not able to find the confusion matrices when removing one feature at a time we have some prediction based on the histogram presented above and knowledge obtained when creating the linear classifier. The confusion matrix for petal length will give the smallest error rate and the confusion matrix for sepal length will have the highest error rate.

5.3 Handwritten numbers - implementation

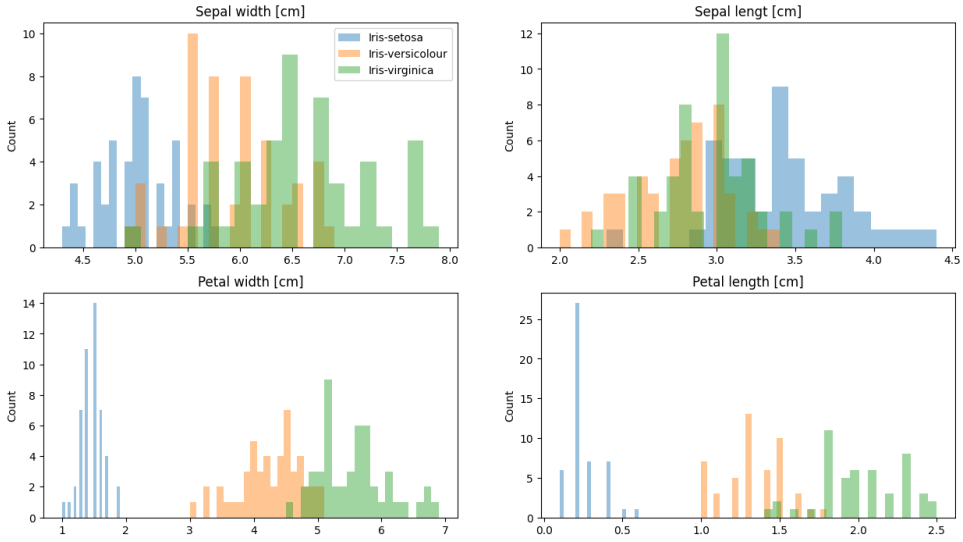
To undertake the task at hand, we divide this section into two parts. First we attempt to classify the data by the use of a NN algorithm. The same is attempted with a KNN algorithm in the second part.



Figur 3: Confusion matrix for trained data.

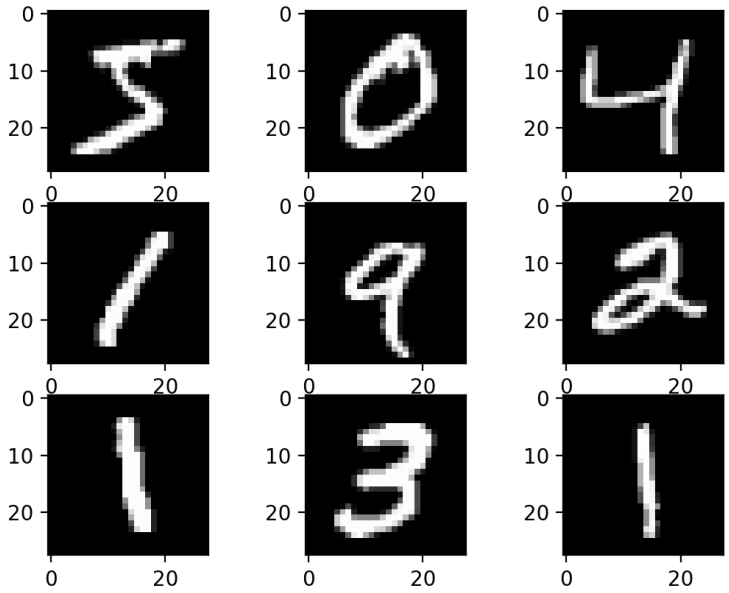


Figur 4: Confusion matrix for test set.



Figur 5: Histogram showing the different features of the iris flower.

The data set contains of images with resolutions of 28x28 pixels in 8-bit greyscale, and as the task-description states, is the database divided to sets of 60,000 training and 10,000 test examples. By preprocessing the pictures have been centred and scaled, thus making them ready for classification. An illustration of the data set is shown in figure



Figur 6: Example of the data set.

The NN classifier is based on the measuring of the distance in the test and training images. The Euclidean distance as described in section 3.4 is implemented for with this purpose. This is allowing for a comparison between the test image and training samples, and with that locating the image with the closest reference points. This is implemented by the definitions *euclidean_dist* and *prediction* and can respectively be found in line 44-46 and 77-83.

An implementation of clustering is attempted by the definition *clustering*, line 113-130. The aim is to divide the training set by creating 64 clusters for each class. To prepare the data for clustering, it is sorted and processed. Further, the $KMeans$ function from the *sklearn* library is obtained for the clustering. By doing this the clusters of the images are made to be the new training sets. On the basis of this definition, we attempt to apply clustering to the KNN classifier in line 37-45, $KNNClustering$.

In order to study the results the error rate is implemented in line 90-97, and a plotting of the confusion matrix discussed in section 3.6 is attempted applied in line 80-86. There is also assumed to be miss classifications, and a plot of those are made in line 134-141.

5.4 Handwritten numbers - results

When loading the MNIST data set using *keras* imported from the library $tensorflow$ we received a backend problem that we unfortunately could not resolve in time. Sadly, the classifiers could not be tested as a whole and there is no results to be shown. We now see that installing the modules with *pipinstall* might have caused the problems, and with more time at hand a development in an *anaconda* environment would have been preferable. Another way of data-fetching would also be sufficient. We also want to note the faults of our algorithms.

6 Conclusion

Unfortunately we have to conclude that neither of the tasks were completed to their fullest extent. In the iris flower task the discriminant vector were wrongfully implemented. This led to the confusion matrices showing us that all the training and test data had been classified as one flower. As a consequence the error rate was 66.7% for both of the data sets. With that being said, the histograms shown in subsection 5.2 as well as the theory in 5.2 gave us useful information and insights when creating LDCs.

The implementation of the hand written task numbers were done according to the theory. Unfortunately the results was not obtained as it became impossible to execute parts of the code since the needed libraries created backend problems in our code editors.

All in all the coding part of the tasks demanded more time and a better understanding than we expected, but gave us valuable knowledge about classifiers.

Referanser

[1] *Euclidean distance*, Wikipedia, `https://en.wikipedia.org/wiki/Euclidean_distance`

[2] *k-nearest neighbors algorithm*, Wikipedia, `https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm15March2022`.

[3] Magne H. Johnsen, *Classification*, 2017.

A Appendix

The project can be found in the Github respiratory: `https://github.com/salyken/TTT_4275_classification.git`