Classification project report

Martin Falang
Eirik Flemsæter Falck

April 23, 2020

Abstract

This report discusses the classification of three species of Iris flowers and the classification of handwritten numbers.

In the classification of the Iris flowers, a linear classifier was used and and the linear separability of the flowers' four features was examined. The linear classifier achieved a low error rate of 3.3% when classifying the flowers using all the features. Due to the linear separability of its features, the error rate remained low even as features were removed from the training of the classifier. When removing the sepal width feature, the error rate increased to 5%. Further removing the sepal length feature, the error rate increased yet again to 8.3%. Using only the petal width feature, the error rate remained the same at 8.3%.

When classifying handwritten numbers the principle of Nearest Neighbor (NN) classification was used, both with and without clustering of the training data. First an 1-NN classifer without clustering was implemented based on Euclidian distance to all of the training data samples. This classifier gave the lowest error rate of all the classifiers at 3.1%, but had a very long runtime of close to half an hour. To increase the classification speed, clustering was introduced in order to preprocess the training data. This proved to greatly decrease the runtime to just shy of 20 seconds of the 1-NN classification and only increased the error rate by a small margin to 4.6%. Lastly a 7-NN classifier was implemented, but this increased the error rate to 6.5% and did not affect runtime compared to the 1-NN classifier. The 1-NN classifier was therefore found to be the overall best classifier.

Contents

1 Introduction

Machine learning is becoming ever more popular, fueled by improvements in large computing clusters as well as increased computing power in smaller, embedded devices. It is found in a multitude of products and services, and is in many ways shaping the future of technology. Hence, knowing when and where to use different machine learning techniques can be vital to solve relevant problems efficiently.

The classification techniques presented in this report serve as a good introduction to the world of machine learning and classification. They form a basis of knowledge to further explore and learn other machine learning concepts such as neural networks.

The goal of the project is to explore two different methods of classification; linear classifier on the Iris species dataset and nearest neighbor algorithm on the MNist dataset. They will be implemented from scratch in Matlab to understand every step of the methods. Firstly, the necessary theory is presented in section 2, and the two tasks are introduced in section 3. In section 4 and section 5, the implementation are described and the results are presented for the Iris task and the MNist task, respectively. These sections will also discuss the results. Finally, section 6 will conclude the report.

2 Theory

This section will present the necessary theory to be able to understand the following sections. Firstly, a brief introduction to classification is given, then specific descriptions of the linear classifier and nearest neighbor classification with relevant topics are given.

2.1 Classification

In classification, the goal is to be able to classify data, e.g. a measurement, as one of several predefined classes. A class means a particular variant of something, e.g. a car may be a class when classifying vehicles or a horse may be a class when classifying animals. The means in which a measurement is classified as a class is through a classifier. A classifier is the process that classifies the input data based on some of its characteristics. For the example concerning the classification of vehicles, a possible characteristic to classify from would be the weight of the unknown vehicle.

The error rate is the fraction of misclassified to total cases, and is an important part of the performance of the classifier. Choosing the right classifier for the problem at hand is therefore important, and the choice depends on the structure of the classes. This means knowing the characteristics of each class and how this relates to the characteristics of the other classes, in order to classify the input data with the highest possible performance. There could be a linear or non-linear relationship between the characteristics, in which case finding a curve to best separate the classes is a viable classifier. In other problems, it may be that the classes seem to bundle up in groups and thus when presented with an input, a viable classifier may be to check which class group the input data is closest to and classify it as that class. These two approaches respectively represent two different types of classification, namely parametric and non-parametric classification.

The practical problem of designing the classifier can be done using a concept known as supervised learning [1]. Supervised learning involves a data set with input data and a label, i.e. the correct class, and this is referred to as the training data set. An example could be image recognition of animals where the the training set are lots of images of animals and a label on each image telling the correct animal depicted. As the name suggests, the training data is used to train the classifier and it is important to have a sufficiently large training set to achieve good performance. A common technique for training is gradient descent where some cost function is minimized in order to achieve the best possible classifier.

After the classifier has been trained using the training data set, it should be tested on a separate set of labeled data, a test set, in order to characterize its performance. By running the test set through the classifier and comparing the classifier's output with the labeled data, the error rate can

be found. Should this error rate not be sufficiently low, measures must be taken to improve the classifier. This could for example be done by increasing the training set or redesigning the classifier based on other class characteristics.

2.2 Linear classifier

As mentioned above, if the classes have a linear relationship with respect to each other, then a linear classifier is sufficient for the classification problem. The linear classifier is a type of parametric classifier which divides the classes to fit a linear model. The basis for the linear classifier is the discriminant function g which for more than two classes is defined in eq. (2.1) as

$$g(x) = Wx + w_0 \tag{2.1}$$

where W the tuning variable (a matrix with dimension CxD, where C is the number of classes and D the number of characteristics) and w_0 is the offset for the classes [6]. The goal of the discriminant function is to provide the values for how sure it is that a certain measurement is part of each class, as g will be a vector with the same dimension as the number of classes. The decision rule when classifying is thus to choose the maximum entry of g and classify the input data as that class.

In order for g to be able to provide sufficient results the W matrix must be optimized. A popular method is to try to minimize the mean square error (MSE) between g and the known data t [6], given in eq. (2.2)

$$MSE = \frac{1}{2}\sum_{k=1}^{N}(g_k - t_k)^T(g_k - t_k) \tag{2.2}$$

where g_k is the discriminant function for a given input and t_k is the actual known class of the given input. As the labeled inputs will often be binary, the discriminant function should be scaled similarly to a value between 0 and 1, and this can be done using the sigmoid function [6].

Using gradient descent, the MSE can be minimized and thus the overall discriminant function will be as accurate as possible. As W is the tuning variable, the gradient of the MSE must be found with respect to W. Then, as the gradient indicates which direction the MSE grows the most, W must be moved in the opposite direction to achieve a declining MSE. It is important that the step length is sufficiently small to ensure a constant descent.

2.3 Linear separability

For a linear classifier to be efficient, the classes must be close to linearly separable. This means that the classes can be separated from each other using a hyperplane, e.g. a line or a plane in 2D and 3D respectively. This is illustrated in fig. 2.1.

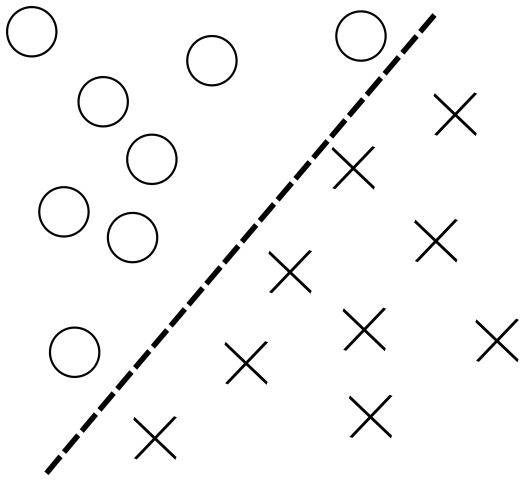


Figure 2.1: Figure showing two linearly separable classes.

If the classes are not linearly separable, then a linear classifier will not be able to classify inputs correctly and a different classifier should be used.

2.4 Nearest Neighbor classification

Nearest neighbor is a non-parametric classification method used in e.g. pattern recognition. In the nearest neighbor algorithm, the similarity metric is the distance between a test vector and training vectors. The test vector is then classified as the closest training vector. A common choice for the distance metric is the Euclidean distance eq. (2.3). Euclidean distance measures the straight-line distance between two points in space. It is easy to implement and use, and will often give satisfactory results.

$$d(x, y) = \sqrt{(x - y)^\intercal (x - y)}, \quad x, y \in \mathbb{R}^n \qquad (2.3)$$

The decision rule in nearest neighbor (NN) is to simply select the class of the data point closest to the test data. This mean to choose the "most similar" class. A more sophisticated decision rule is the k-nearest neighbors (kNN). This selects the class based on the majority vote of the k nearest data points.

The choice of k depends on the data, and can lead to different results. It is common to choose an odd value for k to avoid ties [2]. Figure 2.2 shows how different choices of k can affect the given class to the green point. The dotted line shows k=1, the solid line shows k=3 and the dashed line shows k=5. Respectively, the green point is classified as blue square, red triangle and blue square again. This illustrates how the choice of k can lead to different classification.

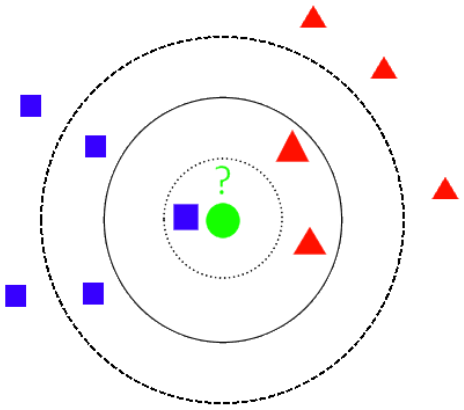


Figure 2.2: The figure shows how different choices of k can affect the classification result. Inspired by: [8].

The kNN algorithm requires labeled training data, making it a supervised method. The labels can be obtained by manually labeling them, or using other methods such as clustering to find appropriate labels.

2.5 Clustering

Clustering is the process of grouping data points such that points in the same group are more similar than points in different groups. There exist many different algorithms for clustering data. An example is the k-means algorithm. This algorithm clusters n data points into k clusters where each cluster in represented by the centroid, i.e. the central position or mean, of the cluster.

Clustering is useful to reduce the data set and create a smaller number of representative templates. Each template is blend of multiple data points. The clusters can then be used as the training data in kNN to achieve greater speed of classification.

3 Task

This section will outline the two tasks that were done. For each task, a short description will be given.

3.1 Iris flower task

The Iris flower comes in three species, Iris Setosa, Iris Virginica and Iris Versicolour, and was first introduced into the field of statistics by Ronald Fisher in 1936 as an example of linear discriminant ananlysis [7]. The flowers consist of petals and sepals, where each of the three species has certain characteristics for its petals and sepals, see fig. 3.1.

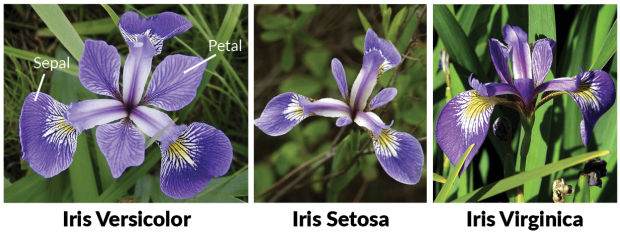


Figure 3.1: Figure showing the different species of the Iris flower, as well as the petals and sepals. Image taken from [5].

The typical features used for classification of Iris flowers are the petal length and width and the sepal length and width. The classes in the Iris classification problem, i.e. the different species, are close to linearly separable which means that a linear classifier can be used to classify the species.

The data provided are 50 samples of each species, containing the four features mentioned above, as

well as the correct class for each set of feature samples.

The first part of the task will be to design a linear classifier in order to classify the species, and to analyze the performance of the classifier given different training and testing criteria. The second part will investigate the linear separability of the classes and how removing specific features from the classifier will affect its performance.

3.2 MNist handwritten numbers task

The MNist database is a large collection of preprocessed images of handwritten numbers from 0 through 9. They are all 8-bit grey scale, cropped to 28x28 pixels and centered for ease of use. The database consist of 60 000 training examples and 10 000 testing examples. Figure 3.2 shows some examples of handwritten images from the MNist database.

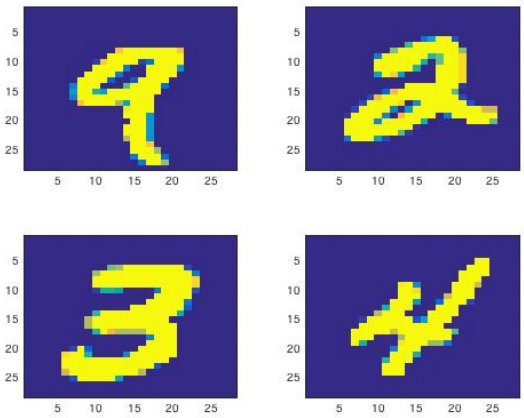


Figure 3.2: Examples of numbers from the MNist database. The numbers are 9, 2, 3, and 4. Source: [4]

The task is split into two parts. In the first part, a NN classifier based on Euclidean distance is designed and tested. Each testing example is compared to every training example and the closest training example is chosen. In the second part, the training data is clustered into 64 clusters. A 7-NN classifier is designed and tested using the clusters as training examples.

This task was chosen because the task of classifying handwritten numbers is widely used as an introductory task to classification and machine learning. This is an interest of the authors and this task gave a good introduction to the field.

4 Implementation and results for Iris flower task

This section includes the implementation, results and discussion of the Irir flower task, described in section 3.1

4.1 Implementation

The implementation of the Iris task was done using Matlab. In taks 1, the objective was to use 30 samples of each class for training and 20 samples of each class for testing. The entire data set was loaded into the Matlab workspace, and from there 30 samples of each class were stored in a training vector and the remaining 20 samples of each class were stored in a test vector.

In this task, an MSE was used to optimize W in the linear classifier. The gradient of the MSE with respect to W as given in [6] was implemented as a function in Matlab. At the local minimum of the MSE, the gradient will be 0. In order for the program to terminate in reasonable time, the program must be allowed to exit once the gradient is acceptably close to 0. For this a tolerance variable was introduced and a while loop was created to run until the norm of the gradient of the MSE was below this tolerance. A value of 0.4 was used for the tolerance.

While the norm of the gradient is above the tolerance the classifier will keep on training itself on the training set. A for loop will loop through all the training samples and calculate the discriminant function and then the gradient of the MSE given that sample. All the gradients were added together to get an upper bound for the gradient for all samples. This gradient was then used to move W a little closer in the opposite direction of the gradient, thus ensuring that the MSE decreases.

The W obtained through the training was then used on the test set in order to find the discriminant

function for each test sample. After scaling the discriminant function using the sigmoid function, the largest element was chosen to be the predicted class for that input and the result was saved in a vector containing the predicted values. This process was repeated with all the test samples and then this was compared with the known values of the tests in order to produce a confusion matrix showing the performance of the classifier. The same process was done using the training set in order to find the error rate when testing with the training set also. A sample of the code including the main while loop is shown in appendix A.

In order to use the last 30 samples for training and the first 20 for testing, the training and test vectors were changed to include the new samples and the same process as described above was applied to find the new error rates.

A flowchart highlighting the key points of the implementation and showing the control flow can be seen in fig. 4.1.

In task two, histograms containing the same feature from each class were plotted using Matlab's `histogram` function. To remove features from the classification, a vector of the desired features was made and only these features were included in the sample vectors. Then the same classification as described above was implemented, using the first 30 samples for training and the last 20 for testing. In order to better observe the linear separability of the features, a scatter plot of the classes was generated using Matlab's `gscatter` function.

4.2 Results

In task 1, first the test set and training set performance were tested using the first 30 flower samples as training samples and the last 20 flower samples as test samples. The results can be seen in fig. 4.2. As seen the overall error rate was 3.3% for the test set and 2.2% for the training set.

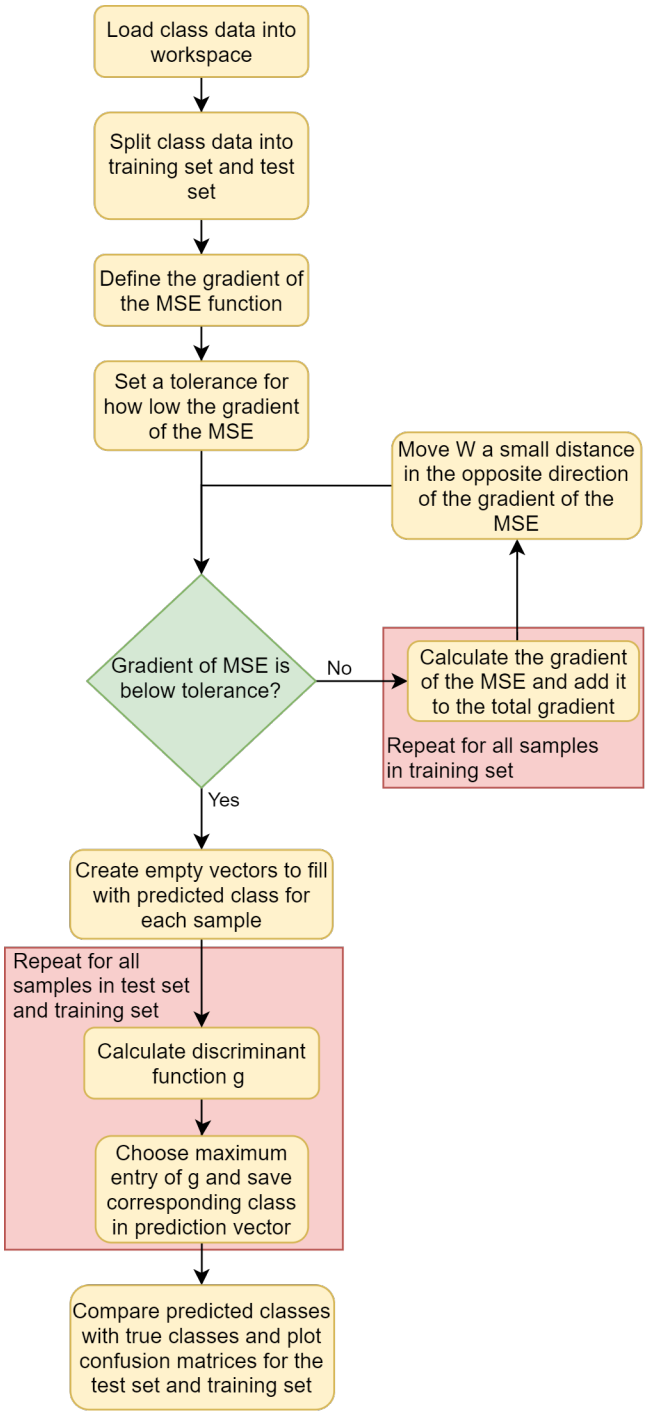
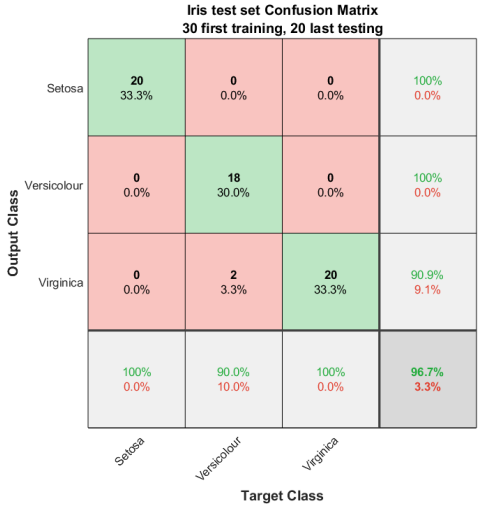


Figure 4.1: Flowchart showing the main steps used to classify the flowers in task 1.

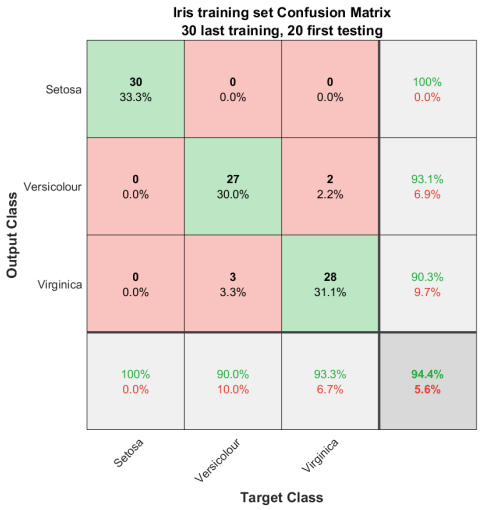


(a) Confusion matrix for the test set.

(b) Confusion matrix for the training set.

Figure 4.2: Confusion matrices when using the first 30 samples for training and the last 20 for testing.

(a) Confusion matrix for the test set.



(b) Confusion matrix for the training set.

Figure 4.3: Confusion matrices when using the last 30 samples for training and the first 20 for testing.

Similarly the results for the case where the last 30 flower samples were used for training and the first 20 used for testing can be seen in fig. 4.3. The error rate of the test samples decreased to 1.7%, while the error rate of the training samples increased to 5.6%.

For task two the histograms comparing the features across the classes can be seen in fig. 4.4. Note that the both the petal width and petal length features are overlapping very little across the classes as opposed to the sepal length and sepal width. In order to investigate this further scatter plots showing the relationship between two and two features of all the classes was generated and is shown in fig. 4.5. It can be seen in this figure that most of the features are close to linearly separable as they have few overlapping points.

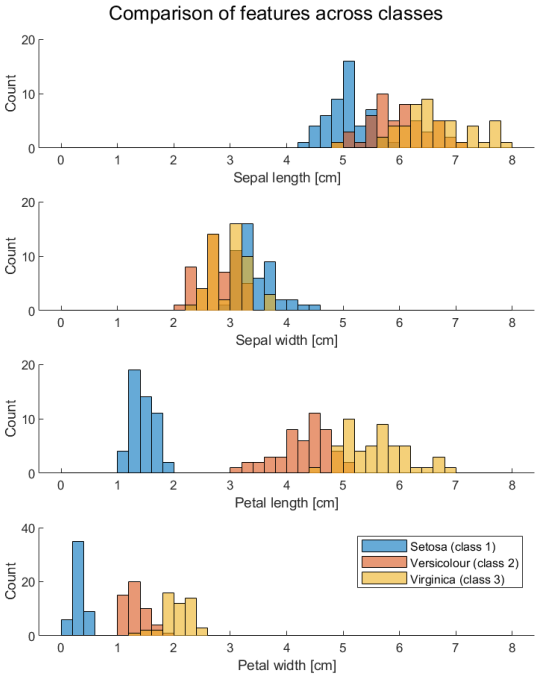


Figure 4.4: Histograms showing the flower features compared across the classes.

When training the classifier using fewer features, first feature 2 (sepal width) was removed and the results for the test set and training set can be seen in fig. 4.6a and fig. 4.6b. Here the error rates was found to be 5%. When removing feature 1 (sepal length) from the classifier as well, the results were as shown in fig. 4.7a and fig. 4.7b. Here the error rates can be seen to have increased to 8.3% for the test set and 13.3% for the training set. Finally when only feature 4 (petal width) was used to train the classifier, the error rate remained the same for the test set at 8.3%, but declined slightly to 12.2% for the test set, this can be seen in fig. 4.8a and fig. 4.8b.

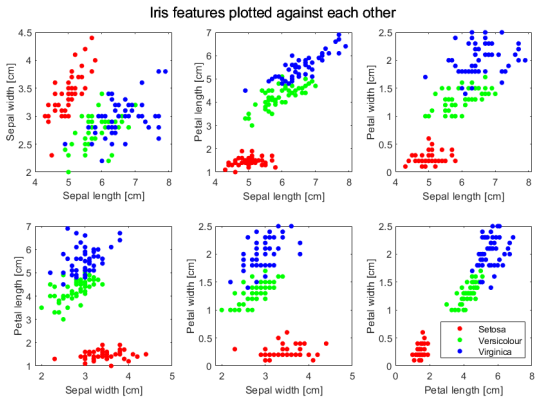
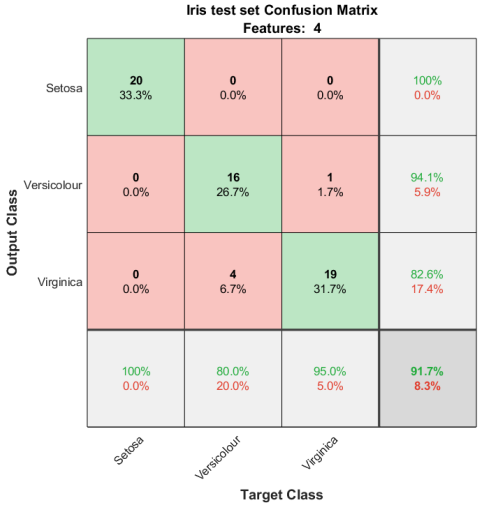
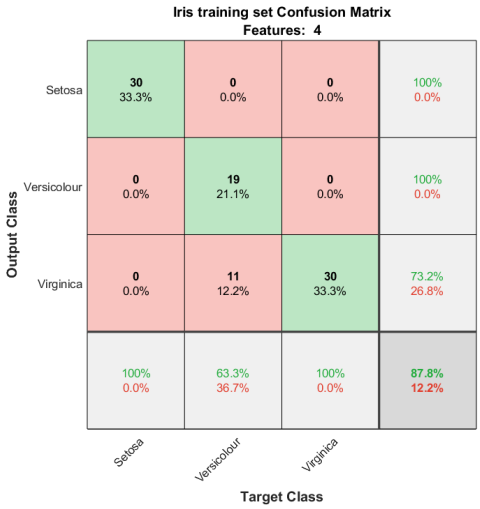


Figure 4.5: Scatter plots of features across the different classes.

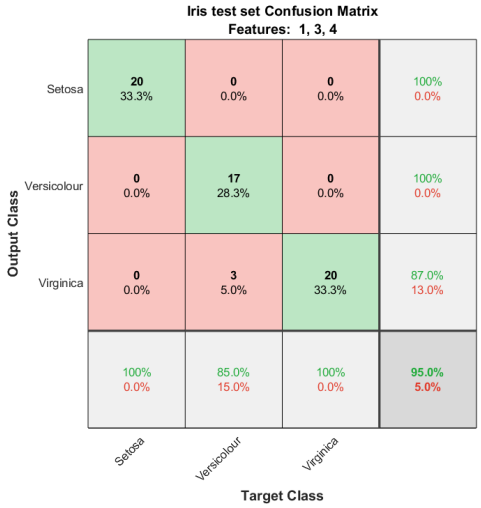


(a) Test set.

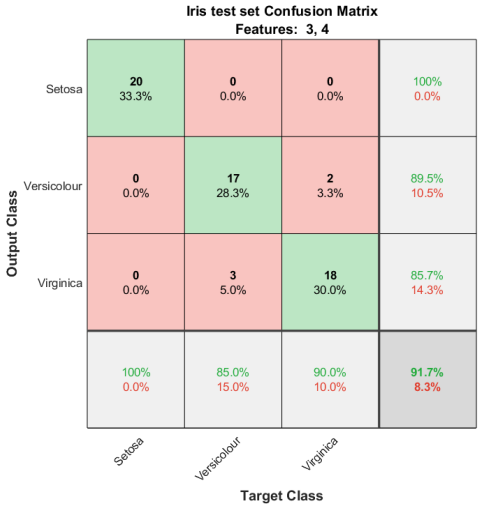


(b) Training set.

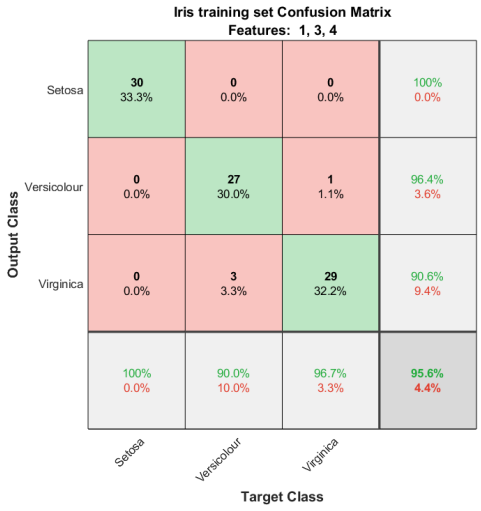
Figure 4.8: Confusion matrices when using only petal width for training.



(a) Sepal length, petal length, petal width.

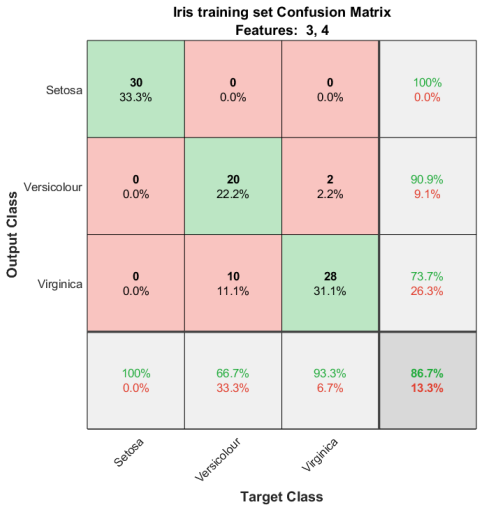


(a) Test set.



(b) Sepal length, petal length, petal width.

Figure 4.6: Confusion matrices when using sepal length, petal length and petal width for training.



(b) Training set.

Figure 4.7: Confusion matrices when using petal width and petal length for training.

4.3 Discussion

Firstly, the results from the first task will be discussed. When using the first 30 samples for training, both error rates are relatively low and it can be seen that the only flower which the classifier misclassifies is the Versicolour which it misclassifies as Virginica twice. This suggests that there is more overlap between the Versicolour and Virginica classes, and this is observed to be true in the scatter plots in fig. 4.5 and the histograms in fig. 4.4. Here it can be seen that the Versicolour and Virginica classes are not as clearly separated as the Setosa class, and thus a linear classifier will have a harder time differentiating the two. As the classifier is trained and optimized using the training set, it is logical that the error rate is lower when testing on the training set than the test set. As seen, this is indeed the case, however it does not actually classify fewer incorrect flowers.

When using the last 30 samples for training the error rate for the test set was seen to decrease while the error rate for the training set increased. Also here the misclassifications only happen between the Virginica and Versicolour classes, while the Setosa class is always classified correctly. The reason why the training set performs worse than the test set may be because the last 30 samples are less linearly separable than the first 30, causing the classifiers performance to decrease. It could also be a coincidence due to this relatively small data set, causing small differences between the two training sets to affect the testing outcome more.

For task two, the histogram produced in fig. 4.4 shows that the sepal width and length for the tree classes are more overlapping than the petal length and width. In order for the linear classifier to be efficient, the classes upon which it uses to train must have features distinct from each other. E.g. the sepal widths are very similar across the classes, and thus a classifier will not be able to differentiate the classes based on sepal width alone. Similarly, it can be seen that the petal width has very little overlap across the classes, and thus a classifier should be able to classify relatively well based on this feature alone. This means that the classes are close to linearly separable for a feature like petal width, but not at all linearly separable for the sepal length feature. This can also be seen in fig. 4.5 where all plots including either petal width or petal length are close to linearly separable, while the plot only containing sepal length and width is not. It was based on this observation that the order of removing features was to first remove sepal width, then sepal length, and then petal length, in order to achieve the lowest error rate possible.

As seen in fig. 4.6, the error rate increases as features are removed, but is never more than 13.3%. The Setosa class is still never misclassified, which is reasonable as there is no overlap between this class and the two others in the histogram fig. 4.4 for petal length and width. It can be seen that the increase in misclassifications happens between the Versicolour and Virginica classes, likely caused by the slight overlap between these two classes in the histograms. The reason why the error rate in the testing set slightly decreases when removing the petal length could be due to the fact that there is more overlap between the Versicolour and Virginica class in petal length than there is in petal width. This could cause the petal length feature to increase the error rate when it is used for training only alongside the petal width feature.

5 Implementation and results for MNist task

This section will look at the MNist task described in section 3.2. A brief presentation of how the task was implemented in Matlab and solved will be given, and then the results will be presented and discussed.

5.1 Implementation

The dataset was given as a Matlab data file and was simply loaded into Matlab. This gives four workspace variables for training and testing images and labels. The images are 1x784 row vectors stored in a large matrix. For instance, the training images is a 60 000x784 matrix. See appendix C for code to draw these images using Matlab.

Task 1 asked for the implementation of an NN (1-NN) classifier based on the Euclidean distance. A for loop loops over the entire test matrix, one

row vector at the time. The function `dist` from the Deep learning Toolbox is used to calculate the Euclidean distance from the single test vector `i` to all the training vectors. This creates a 60 000x1 column vector of distances. The index of the smallest distance is selected using `min` and the corresponding training label is the guess for test vector `i`. Lastly, the confusion matrix is plotted as described in appendix B.

[4] raises a concern about memory usage when calculating the distances using `dist`. It suggest splitting the training data into chunks to reduce the memory usage. It was, however, found that this was not necessary when comparing a single test vector to the entire training set. The distance is of type double, which in Matlab is sorted as 8 bytes [3], meaning the entire 60 000x1 vector of distances only uses 0.48 MB (mega bytes). Since the distance vector is overwritten each iteration, this was seen as insignificant and chunking was not implemented.

Task 2 asked for the implementation of an NN classifier on clustered data. Firstly, the data had to be clustered. The training images were split into cells for each number 0 through 9 and clustered into 64 clusters using the k-means algorithm. Matlabs Statistics and Machine Learning Toolbox has a useful function `kmeans` which was used to perform the clustering on each number. This function returns the centroid of each cluster which serves as the templates for a specific number. The centroids were then concatenated into a matrix which is the clustered training set.

For the 1-NN classifier, the same procedure as in task 1 was used. For the 7-NN classifier, the code was modified. Instead of simply selecting the smallest value, the distance vector from `dist` was sorted using `sort` and then the 7 smallest values were extracted and counted. The number with the highest count was chosen as the prediction.

The execution time of each classification was measured using Matlab's `tic` and `toc` stopwatch timer.

5.2 Results

The result from the 1-NN classifier in task 1 can be seen in fig. 5.1. The error rate can be read from the bottom right corner and is 3.1%.

The execution time for the 1-NN classifier on the unclustered data was 1778.9 seconds, or close to half an hour.

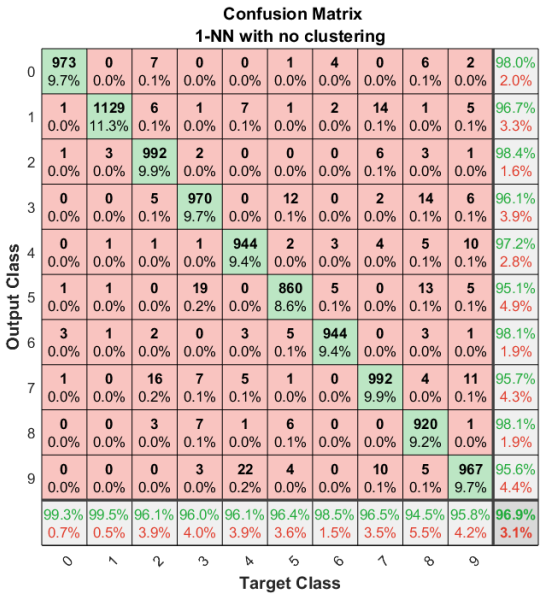


Figure 5.1: Confusion matrix for the 1-NN classifier using unclustered data.

The results from the 1-NN classifier on the clustered data from task 2 can be seen in fig. 5.2, and the error rate is 4.6%. The 7-NN classifier performed worse and had an error rate of 6.5%, which can be seen in fig. 5.3.

The execution time of the clustering was 44.9 seconds. The execution time for the 1-NN classification on clustered data was 19.7 seconds and the 7-NN classification on clustered data was 20.2 seconds.

5.3 Discussion

The 1-NN classifier on the unclustered data in task 1 performed the best out of the three classifications when looking solely on the error rate. Out of the 10 000 test examples, only 310 were misclassified. This is satisfying results.

Figure 5.4 shows some of the misclassified images. As a human reader, some of these mistakes are understandable. For instance, the image guessed as 9 but in reality is a 4 could be misinterpreted even by a human. The same goes for the image

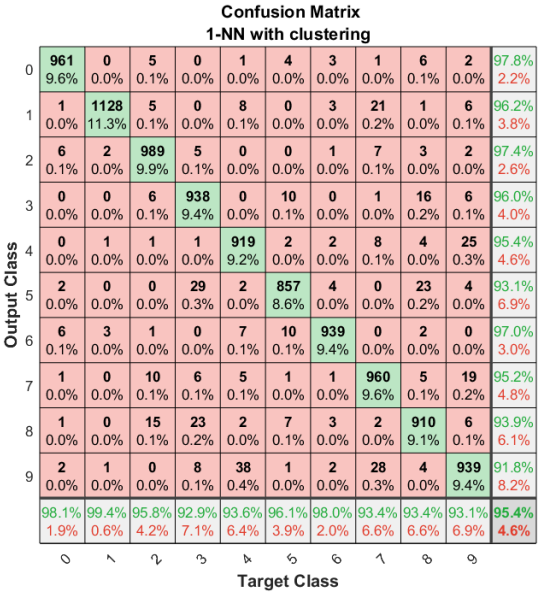


Figure 5.2: Confusion matrix for for the 1-NN classifier using the clustered data.

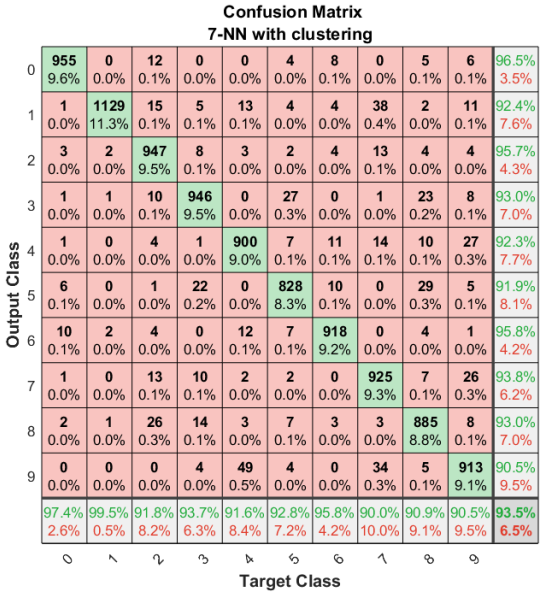


Figure 5.3: Confusion matrix for for the 7-NN classifier using the clustered data.

guessed as 0 but in reality is a 6. With poor/sloppy handwriting, these numbers tend to look similar, and the misclassification of those numbers are understandable.

Looking at fig. 5.1, it can be seen that the misclassificaition of 4 as 9 occurs with the highest frequency, with 22 occurrences. Second highest occurrence is misclassifying 3 as 5 with 19 occurrences and 2 as 7 on third with 16 occurrences. Indeed, all misclassified with an occurrence of over 10 are numbers likely to be misinterpreted by humans.

One interesting note is that the number 4 is more often misclassified as 9 than the number 9 is misclassified as 4 (22 occurrences vs. 10 occurrences). A number 9 is often written as a circle with a line down from the right side, while a number 4 is often written as two parallel lines where one is a little longer, and the lines are connected. Figure 5.5 shows these two ways. A possible reason for the bias towards misclassifying the 4 as 9 can be that humans can be inexact when writing the two parallel lines and they can look like they form a closed shape, almost like a circle, shown in fig. 5.4. However, when writing a 9, humans may not be as likely to make straight lines instead of a circle. Figure 5.4 also show an alternative way of writing a number 4, written as a triangle with a vertical line from one corner. This can be seen in the number classified as a 1 but in reality is a 4. Under certain circumstances, this can be misinterpreted as a 9.

Figure 5.5 shows some correctly classified images. Most of these are clearly the predicted number. The number 5 is the most debatable, it could be misinterpreted as a poorly written 6.

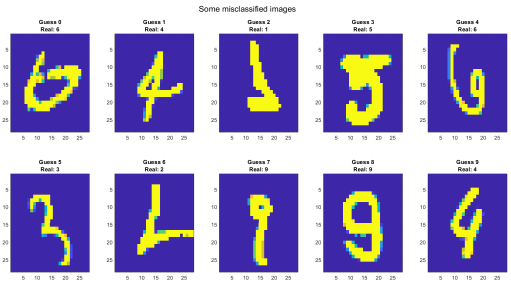


Figure 5.4: Some misclassified images from the 1-NN classifier using unclustered data. The guessed numbers are 0 through 9 from left to right, top to bottom. The correct number is annotated above each image as "real".

Moving on to the clustered results, the error rate is higher than the unclustered results. This

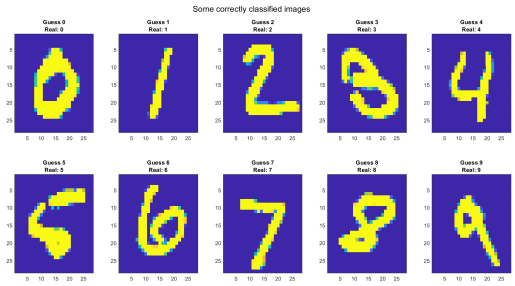


Figure 5.5: Some correctly classified images from the 1-NN classifier using unclustered data. The guessed numbers are 0 through 9 from left to right, top to bottom. The correct number is annotated above each image as "real".

is to be expected as the clustering process reduces some information from the training data. Comparing the occurrences of misclassifications in the 1-NN clustered case to the 1-NN unclustered case, the same mistakes are observed. The most common misclassification is still the number 4 mistaken for a 9.

However, using the error rate as the only performance metric does not give the complete picture. As presented in section 5.2, the 1-NN on unclustered data used almost half an hour to complete, while on the clustered data it used just shy of 20 seconds. The time used to cluster the data is a one time expense, and is not relevant when comparing the execution speed. In some circumstances, this increase in speed is more important than the slight decrease in accuracy. For example, in real-time digit recognition this speed is preferable. In other circumstances, the accuracy is more important. For example, in optical character recognition (OCR) a high accuracy can be more important since there is no deadline for the classification to finish.

Moreover, the 7-NN classifier on the clustered data was found to perform worse than the 1-NN classifier. A possible reason is the issue illustrated by fig. 2.2. When selecting the 7 closest points, similar numbers, such as the 4 and 9 as seen earlier, can cause the count to favor one number over the other and give an incorrect classification. Besides, there were no noticeable speed increase over the 1-NN on clustered data. All in all, the 1-NN classifier is in this case preferable to the 7-NN classifier on the count of both speed and accuracy.

6 Conclusion

Task 1 included the implementation of a linear classifier for classifying three species, classes, of Iris flowers, nameley Iris Versicolour, Iris Setosa and Iris Virginica. The classifier was implemented in Matlab and trained using 30 samples from each class and tested using the remaining 20 samples. When training with the first 30 samples and testing with the last 20, the error rates were found to be 3.3% and 2.2% for the test set and the training set, respectively. When training with the last 30 samples and testing with the first 20, the error rate for the test set decreased to 1.7%, while the error rate of the training set increased to 5.6%. This was believed to be because of variations in the relatively small data set.

As features were removed one by one from the training of the classifier, the linear separability of the data set was investigated. It was found that the petal width and length features were the most linearly separable, while the sepal length and width were less linearly separable. Thus the sepal length and width were removed first and then the petal length was removed until finally petal width was the only feature included in the training. When removing features, the error rate was still never higher than 13.3% in either the training set or the test set as the remaining classes were close to linearly separable. The error rate of the training set decreased when removing the petal length and this was believed to be because there was more overlap between the classes when combining petal length and petal width, than in the petal width feature alone.

Task 2 asked for the implementation of a kNN classifier for handwritten digits from the MNist database. An algorithm for the kNN classifier was implemented in Matlab and used to classify the 10 000 test examples given in the MNist dataset. The training data was later clustered into 64 clusters to reduce the training data set and increase the processing speed. The error rate from running a 1-NN classifier on the unclustered dataset was 3.1%, 1-NN on the clustered dataset was 4.6% and the 7-NN

on the clustered dataset was 6.5%. The execution time was also drastically reduced when running the classifier on the clustered dataset.

The choice of which classifier and which dataset to use depends on the problem at hand. In real-time settings, speed is preferred and the 1-NN on clustered data is preferred. When time is not an issue, the 1-NN on unclustered data can be preferred. In all cases, however, the 7-NN is not preferred due to it having the worst accuracy and no gain in speed.

All code used in this project is available on GitHub at:
`https://github.com/eirik-ff/classification-ttt4275`

A Training a linear classifier in Matlab

The code in listing A.1 shows the main while loop when training a linear classifier. The variable `tk` is a one hot vector containing the correct class for the sample data `xk`.

```
grad_W_MSE_k = @(gk, tk, xk) ( (gk - tk) .* gk .* (1 - gk) ) * xk';
condition = 1;
tol = 0.4;
alpha = 0.001;
while condition
    grad_W_MSE = 0;
    for k = 1:C*N_train
        xk = [xd(k,:)'; 1];
        zk = W*xk;
        gk = sigmoid(zk);
        tk = t(:,k);

        grad_W_MSE = grad_W_MSE + grad_W_MSE_k(gk, tk, xk);
    end
    condition = norm(grad_W_MSE) >= tol;
    iters = iters + 1;

    W = W - alpha*grad_W_MSE;
end
```

Listing A.1: Matlab code for training a linear classifier.

B How to plot confusion matrix in Matlab

The Matlab function `plotconfusion` from the Deep learning Toolbox was used in section 5 to create the confusion matrices. This function expects a matrix of targets (i.e. the known labels) and a matrix of predictions (i.e. the guessed labels) in a one-hot format. Listing B.1 shows the code to create the target matrix `known` and listing B.2 shows the code to create the prediction matrix `guess`. Here, the imagined function `make_prediction` can be any classification algorithm returning an index for the predicted label. Lastly, these two matrices are passed to `plotconfusion(known,guess);`.

```
known = zeros(10,N_test);
for i = 1:N_test
    l = testlab(i);
    known(l+1,i) = 1;
end
```

Listing B.1: Matlab code for generating the one-hot matrix of targets.

```
guess = zeros(10,N_test);
for i = 1:N_test
    test = testv(i,:);
    j = make_prediction(test)
    pred = trainlab(j);
    guess(pred+1,i) = 1;
end
```

Listing B.2: Matlab code for generating the one-hot matrix of predictions.

Official Matlab documentation page can be found at:
`https://se.mathworks.com/help/deeplearning/ref/plotconfusion.html`

C Drawing MNist image with Matlab

Listing C.1 shows the Matlab function used to draw an MNist image vector. The image vector can be given as a 1x784 row vector or a 28x28 matrix. The image has to be flipped and rotated to be in the correct orientation.

```
function draw_image(imv, label)
    % Draws the image imv and add title with label
    %  imv:   image vector, either as 1x784 vector or 28x28 matrix
    %  label: label of picture, will be put as title of plot
    if isequal(size(imv), [1,784])
        imv = reshape(imv, [28,28]);
    end

    imv = fliplr(imv);
    imv = rot90(imv);
    image(imv);

    if nargin == 2
        title(sprintf('%d', label));
    end
end
```

Listing C.1: Matlab function for drawing an MNist image vector.

References

[1] Jason Brownlee. *Supervised and Unsupervised Machine Learning Algorithms*. [Online; accessed 21-April-2020]. 2019. URL: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

[2] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Second edition. Wiley-Interscience, 2001.

[3] *Floating-Point Numbers*. [Online; accessed 15-April-2020]. URL: https://se.mathworks.com/help/matlab/matlab_prog/floating-point-numbers.html#f2-101310.

[4] Magne H. Johnsen. *TTT4275 Project descriptions and tasks in classification*.

[5] Srishti Sawla. *Iris Flower Classification*. [Online; accessed 21-April-2020]. 2018. URL: https://medium.com/@srishtisawla/iris-flower-classification-fb6189de3fff.

[6] Magne H. Johnsen Tor A. Myrvoll Stefan Werner. *Estimation, detection and classification theory — compendium*.

[7] Wikipedia contributors. *Iris flower data set — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2020]. 2020. URL: https://en.wikipedia.org/wiki/Iris_flower_data_set.

[8] Wikipedia contributors. *K-nearest neighbors algorithm — Wikipedia, The Free Encyclopedia*. [Online; accessed 14-April-2020]. 2020. URL: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.