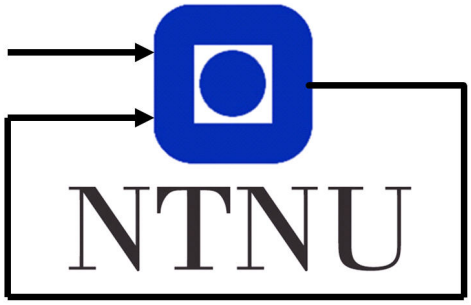
TTT4275 - Iris and Digits project

Sigurd von Brandis: 544973
Aleksander Klund: 544983

April 29, 2023



Department of Engineering Cybernetics

1 Summary

This report presents the findings of a classification project that involved two parts. In the first part, a data set containing three types of Iris flowers was provided, with data on length and width of petal and sepal. A linear classifier was used on the data set, which revealed that two of the iris flowers were not linearly separable, but one of them was. We experimented with removing features with a high degree of overlap, but found that while removing features resulted in faster runtimes, it also led to increased error rates, although still preforming quite well.

In the second part of the project, variants of the nearest neighbour (NN) classifier was explored. We initially used a NN-based classifier with the entire training set as templates, which achived the best precision with an error rate of approximately 3.1%. However, this approach resulted in a runtime of 44 minutes. To address the long runtime, we divided each class into 64 clusters, which reduced the runtime to 32 seconds but increased the error rate to 4.5%. Finally, we tried a K-nearest neighbour (KNN) classifier with the same clustering and taking into account the seven nearest neighbors, but it resulted in a higher error rate of 6.3% and a slightly worsened runtime compared to the NN classifier with clustering.

Contents

1 Summary **1**

2 Introduction **3**

3 Theory **3**
3.1 Iris Flowers 3
3.2 Digits 5

4 The Tasks **7**
4.1 Iris 7
4.2 Digits 7

5 Implementation and results **8**
5.1 Iris 8
5.2 Digits 12

6 Conclution **17**
6.1 Iris 17
6.2 Digits 18

7 Appendix **20**
7.1 MATLAB code; Iris Flowers 20
7.2 MATLAB; Digits, NN classification 23
7.3 MATLAB; Digits, NN classification with clustering 24
7.4 MATLAB; Digits, KNN classification with clustering 26

2 Introduction

Machine learning has brought about a paradigm shift in our modern world, as it empowers computers to learn from data and make predictions and informed decisions without explicit programming. This transformative technology continues to shape our world today, and it's impact is poised to persist into the foreseeable future. In this report, we unveil the results of our classification project, a small step into the vast realm of machine learning. Classification, a cornerstone task in this field, boasts a wide array of applications, spanning from image and speech recognition to medical diagnosis and autonomous vehicles. Given our background as cybernetics and robotics students, delving into this topic was a natural choice for us.

The goal of the project is to get to know some different types of simple classifiers, to be used in the tasks that will be described shortly. In chapter 3, the fundamental theory needed to understand the tasks is presented. After the fundamentals are in place, we will delve into the classification tasks implementation and their results. Finally, we will conclude with some analysis and thoughts regarding the results.

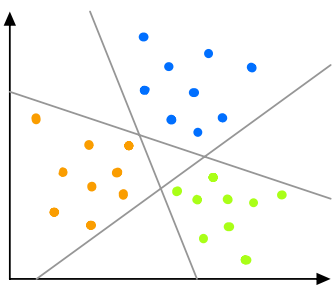
3 Theory

This section explains the theoretical foundation necessary for comprehendig the two tasks at hand. First, the theory relevant to the iris flower task is going to be explained. Subsequently, we will do the the same for the Digits task. The theory presented in this chapter is based on the course compendium[1].

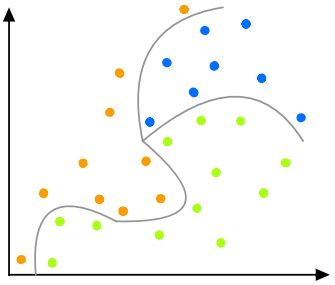
3.1 Iris Flowers

Linear separability, linear classifiers and how to train linear classifiers is the theory needed to understand the Iris Flowers classification task.

Linear separability is a concept in machine learning and pattern recognition that refers to the ability of a set of data points to be separated into distinct classes using a straight line or a hyperplane in the feature space. In other words, if a data set can be divided into different categories using a linear decision boundary, it is said to be linearly separable. Mathematically, a data set is linearly separable if it is possible to find a hyperplane, which is a higher-dimensional generalization of a line, that can perfectly separate the data points of different classes without any misclassifications. This means that all the data points of one class lie on one side of the hyperplane, while all the data points of the other class lie on the other side. Linear and non-linear separability is illustratied with 3 classes in figure 1a and figure 1b respectively.



(a) Linear Separability



(b) Nonlinear Separability

A linear classifier has the output given by the discriminant vector

$$g = Wx + w_0 \tag{1}$$

where W is a adjustable variable, x is the input data for an instance, and w_0 is the offset of a spesific class. This can be rewritten to combine W and w_0

$$g = \begin{bmatrix} W & w_0 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} \rightarrow g = Wx \tag{2}$$

To optimize the performance of a linear classifier, it is trained by minimizing a cost function on a known data set. For this project the "Minimum Square Error" (MSE) cost function was choosen. It is given by

$$MSE = \frac{1}{2} \sum_{k=1}^{N} (g_k - t_k)^T (g_k - t_k) \tag{3}$$

where g_k represents the predicted output for an instance, and t_k is the known real value of the same instance. The Gradient of the MSE, with respect to W, is given by

$$\nabla_W MSE = \sum_{k=1}^{N} \nabla_{g_k} MSE \nabla_{z_k} g_k \nabla_W z_k \tag{4}$$

It can be shown that this is equal to

$$\nabla_W MSE = \sum_{k=1}^{N} [(g_k - t_k) \circ g_k \circ (1 - g_k)] x_k^T \tag{5}$$

Where $\circ$ means element-wise multiplication. In our classifier implementation we also use the sigmoid function

$$g_k = \frac{1}{1 + \exp -z_k} \tag{6}$$

to compress the output of the cost function to a value between 0 and 1 for easier handling of data.

The value of the cost function is dependant on the performance of W, with a perfect W resulting in an MSE of 0. If W is not optimal, an adjustment in the opposite diretion of the gradient of the cost function is made. The gradient of the cost function indicates the direction which increases the cost function the most. Therfore, we want to move in the opposite direction, such that we move in the direction of steepest decent, thus reducing the cost function. This process of optimizing the classifier by adjusting W is known as training, and will be further discussed in the implementation section.

3.2 Digits

The nearest neighbour (NN) classifier is a type of machine learning algorithm, more specifically a template-based algorithm, that works by finding the nearest data point(s) in a training set to a given input data point. The NN classifier then assigns the input point the same label as its nearest neighbour. The distance metric used to measure the similarity between data points is the Euclidian distance, meaning measuring the absolute distance between two points in a Euclidian space. This absolute distance can be generalized for higher dimensional space with the formula

$$d = \sqrt{\sum_{k=1}^{N}(x_{ki} - x_{kj})^2} \tag{7}$$

where $x_{k1} - x_{kj}$ is the distance between two points i and j in the k-th dimention[2].

The K nearest neighbour (KNN) classifier is an extension of the NN classifier that considers the K nearest neighbours to a given input data point. The KNN classifier assigns the input point to the class that appears most frequently among its K nearest neighbours. If the same amount of two different classes end up among the K nearest, the class that is closest is picked.

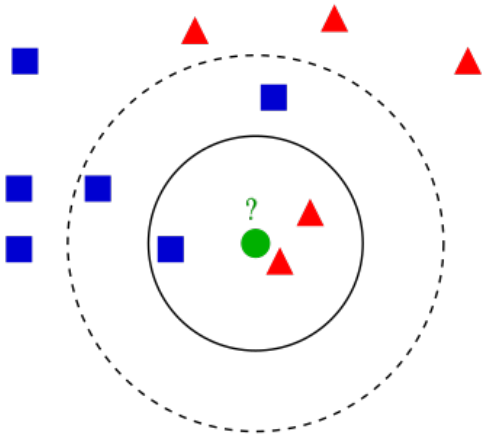


Figure 2: Illustration of a KNN-classifier with K=3 and K=5

A way to improve the computational efficiency of the classifier is by utilizing clustering. The principle of clustering data involves grouping similar data points together into clusters. This can be achieved using various clustering algorithms, one such algorithm being k-means clustering. An iterative, data-partitioning algorithm that assigns the data to exactly one of k clusters defined by centroids, where k is chosen before the algorithm starts. The iterations consists of performing calculations to optimize the position of the centroid[3].

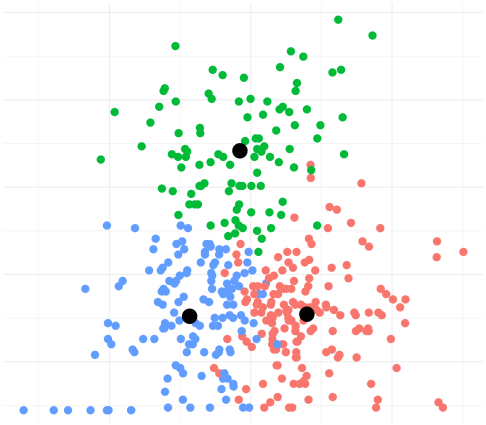


Figure 3: Data set divided into three clusters, each with a centroid

4 The Tasks

This section contains the description of the Iris Flowers and Digit tasks.

4.1 Iris

The Iris flower has three variants called Setosa, Versicolor, and Virginica. They have varying lengths and widths of Sepal and Petal leaves. These measurements are used as input features to create a data set with 50 examples for each variant. The task is to design and evaluate a linear classifier for the Iris task, and to analyze the importance of each of the four features with respect to linear separability.

The first part of the task involves designing, training, and generalizing a linear classifier using the first 30 samples for training and the last 20 for testing. The confusion matrix and error rate for both sets are calculated, and the last 30 samples are used for training and the first 20 for testing to compare the results.

The second part of the task focuses on features and linear separability. Histograms are produced for each feature and class, and the feature with the most overlap is removed. A classifier with the remaining three features is trained and tested, and the experiment is repeated with two and one feature(s). The confusion matrixes and error rates for the four experiments are compared, and the property of the features with respect to linear separability is analyzed.

4.2 Digits

The MNIST database contains 28x28 8-bit greyscale images of handwritten digits 0-9, preprocessed for classification. The database has 60000 training examples and 10000 test examples written by different persons. The task has two parts both using a nearest neighbourhood classifier.

The first part uses the whole training set as templates to design a NN-based classifier using the Euclidian distance. Misclassified and correctly classified images are plotted and analyzed.

In the second part, clustering is used to produce a smaller set of templates for each class. The training vectors for each class are clustered into 64 templates using k-means. The NN classifier using these templates is evaluated for its confusion matrix and error rate. A KNN classifier with K=7 is also designed and compared with the other two systems.

5 Implementation and results

This section provides a detailed description of the implementation of each task, while also drawing connections to the theory discussed. In the end, the results will be discussed and explained.

5.1 Iris

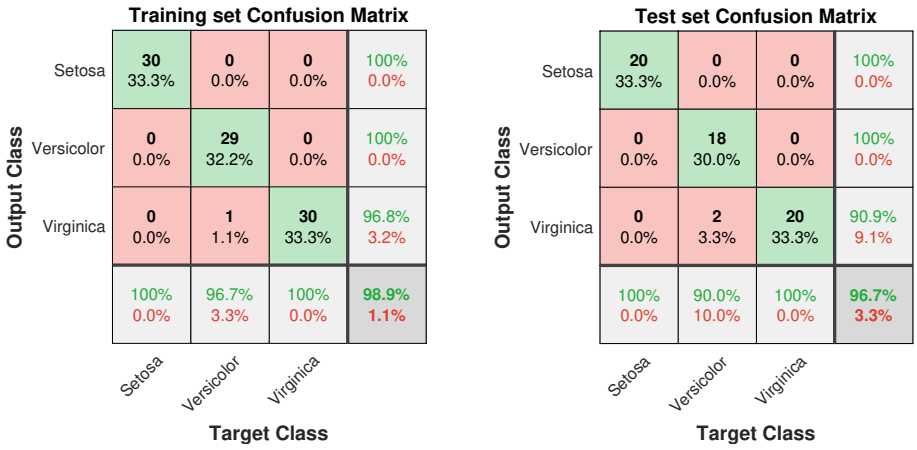
For this task, the data set consising of 50 samples from each of the three classes was loaded into our Matlab file, before dividing them into training and test data, using the first 30 as our training set with the last 20 as our test set, and then the last 30 as our training set with the first 20 as our test set.

The weighing matrix W for our classifier, was found iteratively using $\nabla_W MSE$, with a step factor $\alpha = 0.005$ and fixed at 30 000 iterations. For each iteration the samples of the training set were also iterated through, first passing the samples trough the discriminant function (2), before using the sigmoid function (6) on its output. The output from the sigmoid function is then used, together with the correct classification of the samples, to calculate $\nabla_W MSE$ (5). A new value of W is then calculated using

$$W(k) = W(k-1) - \alpha \nabla_W MSE \tag{8}$$

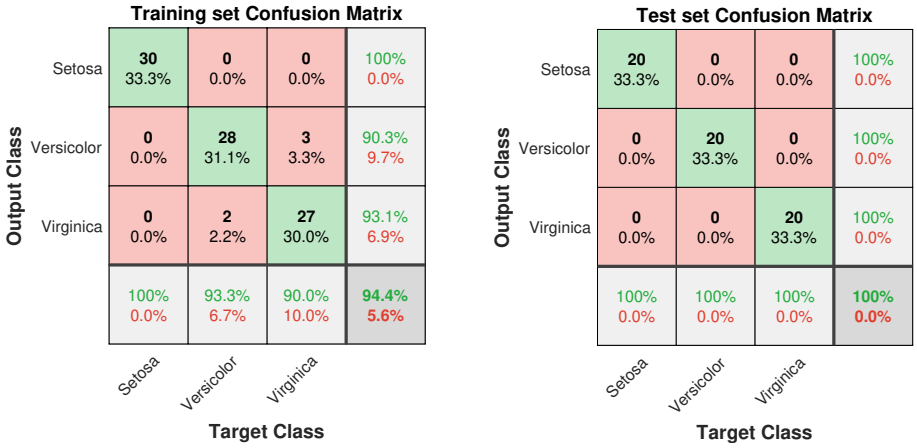
Where k is the iteration number. The discriminant function, with our newly found weighing matrix, is then used on the training set and the test set separately. The largest element was then picked out from the output g, using the matlab function $\mathbf{x = max(g)}$, giving us the class that the sample most likely belongs to according to the linear clasifier. The result are then given in a confusion matrix, plotted with the help of **plotconfusion(targets,outputs)**, another matlab function.

We see from the results that Iris Setosa is linearly separable from the other two, as it correctly classified in all cases. Iris Versicolor and Iris Virginica however are not linearly separable from each other, but considering the small training set, the performance was still very good. In figure 4 and 5, you can see the result of two ways of dividing the data set, using the 30 first samples for training and using the 30 last samples for training. The latter is able to perfectly classify the test set, but the preformance does however get worse when classifying the training set. The former gives better classification of the training set, but has worse performance when classifying the test set. This seems to imply that the second halves of the data sets for Versicolor and Virginica are harder to linearly separate from each other, while the first 20 samples are entirely linearly separable.



(a) Training Set (b) Test Set

Figure 4: Training with first 30, testing with last 20



(a) Training Set (b) Test Set

Figure 5: Training with last 30, testing with first 20

When plotting the histograms as seen in figure 6, sepal width seems to be the feature with the most overlap, with only Setosa being somewhat separate from the rest. Sepal length also suffers from significant overlap, with Versicolor and Virginica having very similar distributions, although somewhat shifted. Setosa being mostly separate with a strong peak. Petal length and Petal width seem to follow this trend, with the latter having almost no overlap, exept some between Versicolor and Virginica.

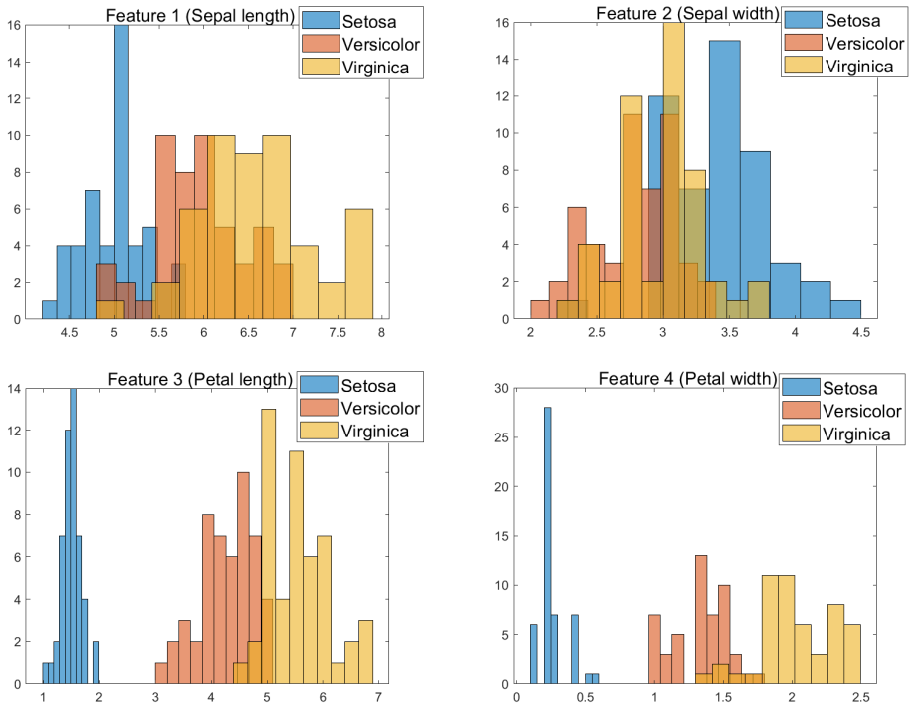
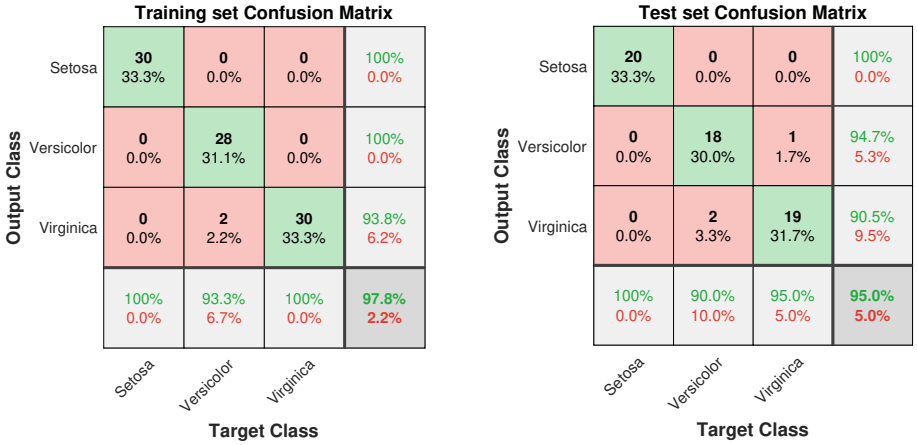


Figure 6: Histograms of all 4 features

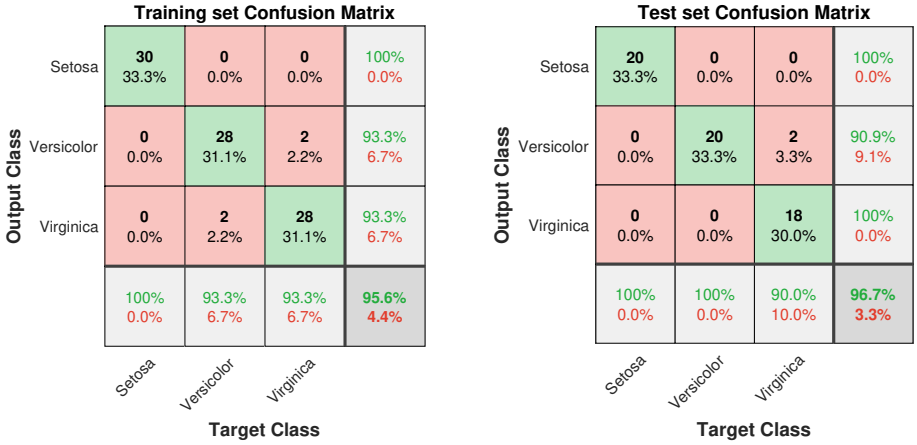
Sepal width was therefore the first to be removed, which led to an increased error rate in the test set from 3.3% to 5.0%, and from 1.1% to 2.2% for the training set. Even with the very large overlap, using more features gives us more dimensions to separate the classes in, and should improve performance as long as they are not completly overlapping.



(a) Training Set (b) Test Set

Figure 7: Without Sepal width

When sepal length is removed, as seen in figure 8, there is a drop in performance for the training set as expected, with the error rate increasing to 4.4%. For the test set however there is a drop in the error rate to 3.3%, likely a statistical anomaly due to a small data set, as the linear classifier trained with these features just happens to be able to better classify the training set, atleast when looking at the total error rate.

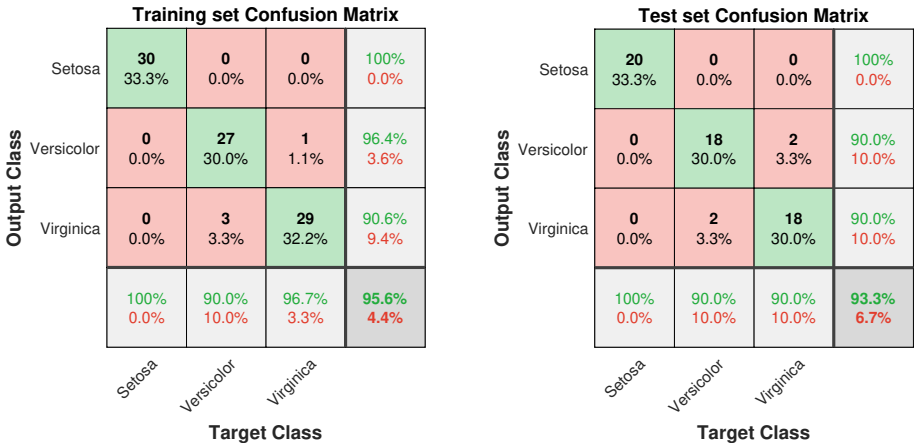


(a) Training Set (b) Test Set

Figure 8: Without Sepal length and Sepal width

Romoving petal length, the classifier is left with only one feature, petal width. The error rate for the training set remains at 4.4%, but with a shift to more Versicolor being misclassified. The error rate of the test set increasing to 6.7%, which although not as good as when using all four features,

accomplishes this with only a quarter of the data. Overall, a slight decrease in performance can be observed when removing features, but the classifier still maintains good performance even when only using one feature, aswell as decreasing the computing time when training.



(a) Training Set (b) Test Set

Figure 9: Without Sepal length, Sepal width and Petal length

5.2 Digits

We started by loading the data from the MNIST database, provided by Matlab data files. The training data was formed as a 60000×28^2 matrix, where the row vectors was the images. All of the classification implementations used a for-loop to iterate through the test data. Matlab's built-in tic toc functionallity was used to get the total computation time.

Initially, the NN classifier was implemented by utilizing the entire training set as templates. This approach involved comparing each item in the test samples with all 60,000 training samples. As a result, a large amount of computation was required, which led to a runtime of 44 minutes, and a error rate of 3.1%. For each iteration, the current sample of the test data was compared to the 60 000 training samples using the matlab function **dist()**, to calculate the Euclidean distances. The built in **min()** function was then used to find the index in the training set, that was closest to the current sample, by using the calculated distances. The class corresponding to that index was then found by checking with the answer for the training data. Following that, this value would be assigned as the estimated class for that specific instance in the output matrix, before starting the next iteration. The implementation can be found in the Appendix, and the result can be seen in figure 10.

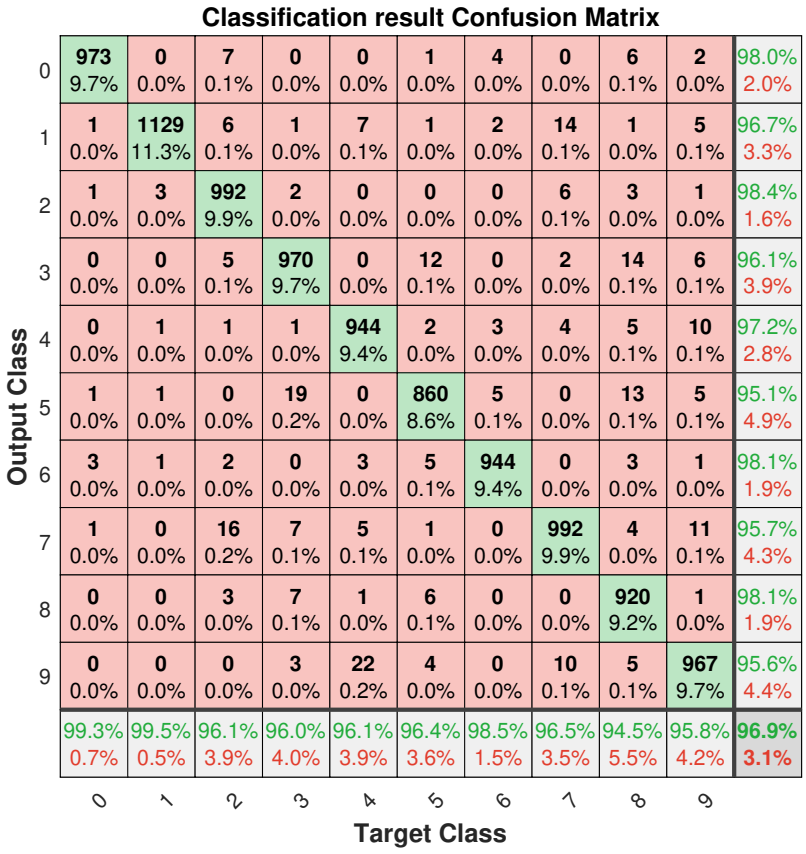


Figure 10: NN classifier without clustering

K-means clustering was implemented by exctracting the unique class lables from the training set using the function **unique()** and splitting the training set into clusters of size M using the function **k-means()** on each class separately. The resulting centroides was stored in **trainv_cluster**. For each cluster, the corresponding class label was stored in **trainlab_cluster**. For each sample in the test set, the index of the closest centroid was then stored in **closest_distance_index**. The predicted class for the sample, was then estimated to be the class of the cluster with the closest centroid. This was then stored in **outputs**. The implementation can be found in the Appendix, and the result can be seen in figure 11.

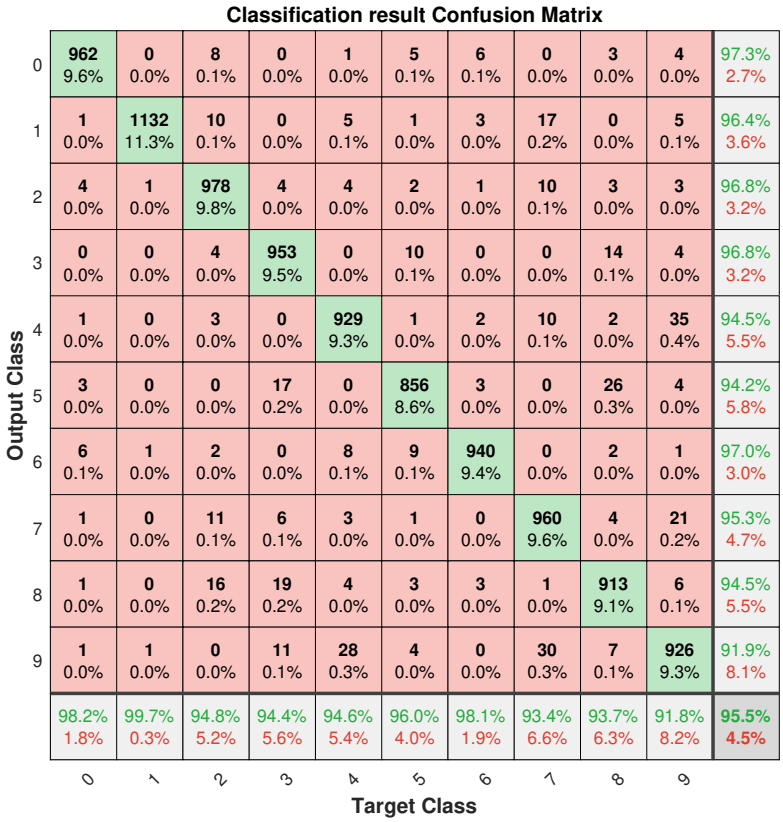


Figure 11: NN classifier with clustering

Although there is a slight variation in the results after reclustering, the error rate remained around 4.5%. While using the entire training set as templates yields lower error rates, the performance achieved with clustering is still good. Clustering takes an additional 30 seconds, but significantly reduce the runtime after clustering to just 35 seconds.

The KNN classifier with clustering was implemented by sorting the distances from closest to futhest away. We used **trainlab_cluster**, which contains the classes for the different clusters, to find the K closest classes. We then used the Matlab function **mode()** to find the most frequent class among the K closest. The class of the current sample was then estimated to be the most frequent class. The implementation can be found in the Appendix, and the result can be seen in figure 12.

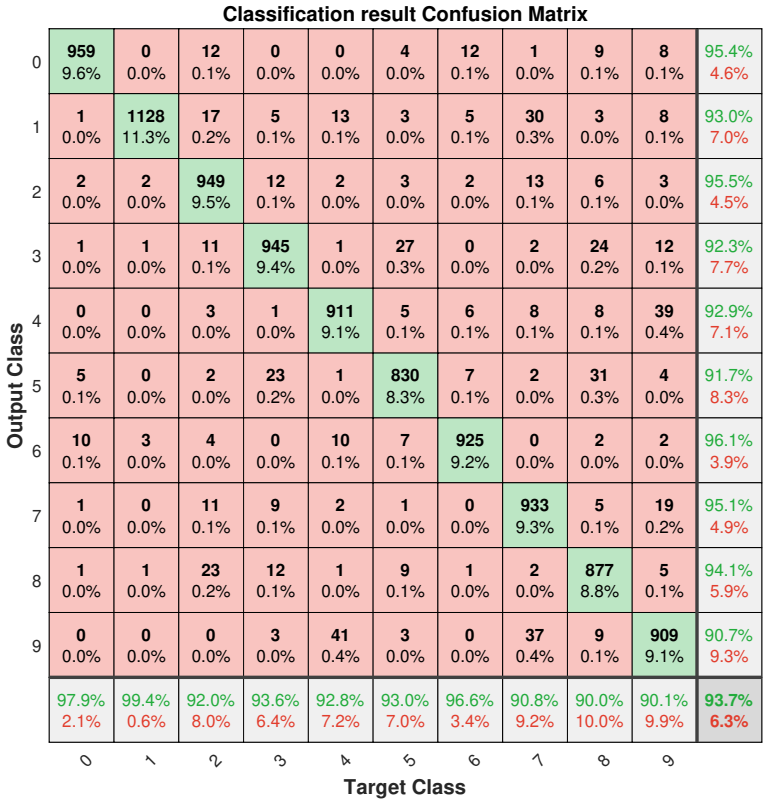


Figure 12: KNN classifier with clustering, K=7

The KNN classifier with K=7 and clustering resulted in worse performance compared to the NN classifier. The error rates were around 6.3%, and the runtime increased by a few seconds. During testing, we experimented with different values for K between two and seven, but making the KNN classifier a NN classifier by assigning K the value of one, delivered the best performance. Perhaps due to Euclidian distances not being the best way to differentiate the classes in this case,

As we can se from figure 10, 11 and 12, the lowest error rates was achived with the NN classifier with no clustering. The error rate was at 3.1%, but because of the way this classifier work, which was discussed in the theory chapter, the runtime was 44 minutes on our computers.

In figure 13 and 14 we can see some of the correctly and incorrectly classified digits from using this classifier. Overall, the digits present in the correctly classified images exhibit a neater appearance as opposed to those which are incorrectly classified. However, it should be noted that the digit in the third row and first column of the correctly classified images could have been a 4 instead of a 9. Despite being somewhat poorly written, the

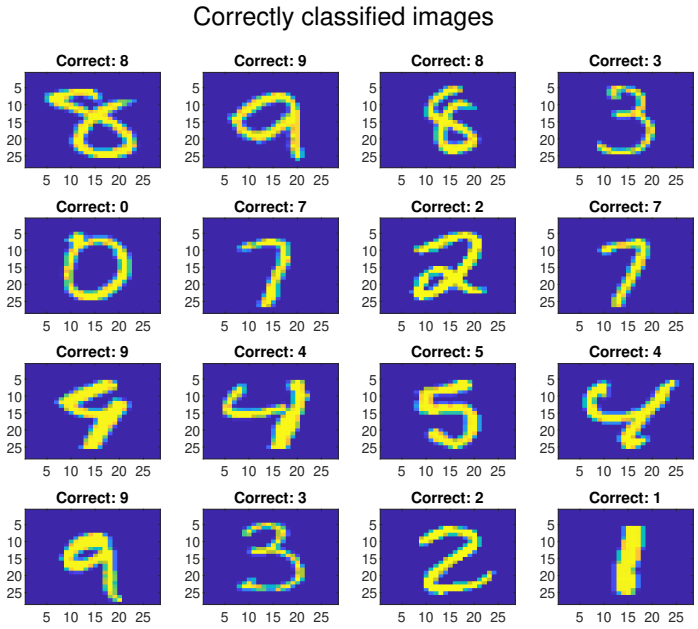


Figure 13: Images that have been Correctly classified

incorrectly classified digits did not pose a significant challenge for us to classify. On the other hand, we had difficulty making sense of the last two digits in the first row and the last two in the third column. Generally the classifier seems to struggle more with classifying the digit 8, which in terms of pixel values could be somewhat close to a few other digits. It most rarely misclassifies 1 and 0, perhaps due to being quite different from the other digits, aswell as less complexity involved in drawing them, thus a higher level of uniformity throughout the data set. When looking at the error rates we also see that the classifier seems to do perform rather well when classifying 6, but it does misclassify the second row and second column as seen in figure 14, even if it is rather obvious to our human eyes..

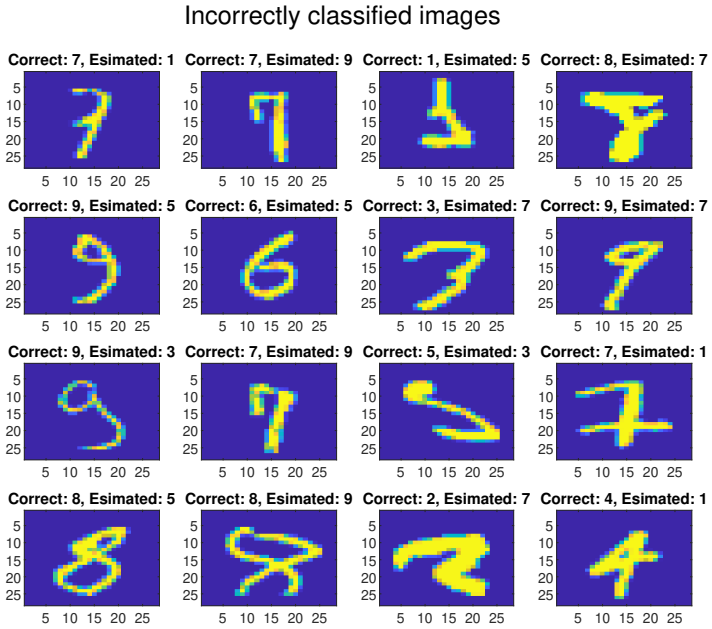


Figure 14: Images that have been incorrectly classified

6 Conclution

Presented here are the conclusions drawn from the two tasks, along with a summary of the results obtained. Furthermore, we share our reflections on the project from our point of view.

6.1 Iris

The objective of this assignment was to employ a linear classifier for the purpose of categorizing three distinct flower varieties.

There is uncertainty surrounding the results of the classifications due to the small size of the training set, which consists of 30 samples from each class. As a result of the small size of the training set, the results are more vulnerable to variations in the length and width of the samples within each class. Simply picking the 30 first samples, is as we have seen, not a good way to get a training set that properly represents the entire class. Removing features does not give better performance, with the exception of what was likely an anomaly, that became more significant due to the small data set. It does however require less data, and is less computationally demanding.

The assignment was quite intriguing, as it provided an opportunity to acquire knowledge on the simple, yet in this case, effective medthod of linear classification. Furthermore, the results of removing features, as mentioned

earlier, made for an interesting experience.

6.2 Digits

The objective of this assignment was to use different versions of the nearest neighbor classifier for accurately categorizing the handwritten digits from the MINST database.

We may perhaps have achived better error rates with KNN without clustering. This is because clustering removes some of the training data points and thus leads to a loss of information. However, the reason for using clustering is to reduce the computational complexity of the KNN classifier by reducing the number of templates. Therefore, it is a trade-off between performance and computational efficiency. Doing KNN classification without clustering was however not asked for by the project tasks.

Since the NN classifier only considers the pixel values and does not recognize shapes, it may struggle to recognize digits that are easily recognizable by humans. This is because humans rely on a combination of features such as shape, context, and texture to recognize objects, including digits. Using non-Euclidian distance would likely improve the performance, but at the cost of added complexity. However, the Euclidian distance, pixel-based approach still worked well and it's ease of use can outweigh it's limitations.

It was an engaging and informative exercise to tackle. The realm of machine learning is a interesting and impactfull field, and this activity served as an excellent primer for further study.

References

[1] Tor A.Myrvoll, StefanWerner, Magne H.Johnsen *Book: Estimation, Detection and Classification Theory.* 2017. Chapter 3.

[2] Brian Halabisky *Euclidean Distance In 'n'-Dimensional Space* `https://hlab.stanford.edu/brian/euclidean_distance_in.html` 28.04.2023

[3] Education Ecosystem (LEDU) *Understanding K-means Clustering in Machine Learning.* `https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1`, 29.04.2023.

7 Appendix

7.1 MATLAB code; Iris Flowers

```
%% Load the data
x1all = load('Delivered/class_1');
x2all = load('Delivered/class_2');
x3all = load('Delivered/class_3');
%

%% Histograms;
figure(1)
histogram(x1all(:,1), 10);
hold on;
histogram(x2all(:,1), 10);
hold on;
histogram(x3all(:,1), 10);
lgd = legend('Setosa', 'Versicolor', 'Virginica');
fontsize(lgd,14,'points')
sgtitle('Feature 1 (Sepal length)')

figure(2);
histogram(x1all(:,2), 10);
hold on;
histogram(x2all(:,2), 10);
hold on;
histogram(x3all(:,2), 10);
lgd = legend('Setosa', 'Versicolor', 'Virginica');
fontsize(lgd,14,'points')
sgtitle('Feature 2 (Sepal width)')

figure(3);
histogram(x1all(:,3), 10);
hold on;
histogram(x2all(:,3), 10);
hold on;
histogram(x3all(:,3), 10);
lgd = legend('Setosa', 'Versicolor', 'Virginica');
fontsize(lgd,14,'points')
sgtitle('Feature 3 (Petal length)')

figure(4);
histogram(x1all(:,4), 10);
hold on;
histogram(x2all(:,4), 10);
hold on;
histogram(x3all(:,4), 10);
lgd = legend('Setosa', 'Versicolor', 'Virginica');
fontsize(lgd,14,'points')
```

```
sgtitle('Feature 4 (Petal width)')


%Parameters that classes are based upon.
class_Setosa= x1all;
class_Versicolor= x2all;
class_Virginica= x3all;

% class_Setosa= [x1all(:,1), x1all(:,3), x1all(:,4)];
% class_Versicolor= [x2all(:,1), x2all(:,3), x2all(:,4)];
% class_Virginica= [x3all(:,1), x3all(:,3), x3all(:,4)];

% class_Setosa= [x1all(:,3), x1all(:,4)];
% class_Versicolor= [x2all(:,3), x2all(:,4)];
% class_Virginica= [x3all(:,3), x3all(:,4)];

% class_Setosa= [x1all(:,4)];
% class_Versicolor= [x2all(:,4)];
% class_Virginica= [x3all(:,4)];

[Ntot,dimx] = size(class_Setosa);

%% Make Training and test sets
% TrainingSetLength = 21:50;
% TestSetLength = 1:20;
TrainingSetLength = 1:30;
TestSetLength = 31:50;
N_Training = length(TrainingSetLength);
N_Testing = length(TestSetLength);

%% Make Training Data
Tot_Training_Data = [class_Setosa(TrainingSetLength,:);
                     class_Versicolor(TrainingSetLength,:);
                     class_Virginica(TrainingSetLength,:)];

%% Make Test Data
Tot_Testing_Data = [class_Setosa(TestSetLength,:);
                    class_Versicolor(TestSetLength,:);
                    class_Virginica(TestSetLength,:)];

%% Make matrices used in confusion matrix
Correct_Answer_Training = [kron(ones(1,N_Training),[1; 0; 0]),
    kron(ones(1,N_Training),[0; 1; 0]), kron(ones(1,N_Training),[0;
    0; 1])];
Correct_Answer_Testing = [kron(ones(1,N_Testing),[1; 0; 0]),
    kron(ones(1,N_Testing),[0; 1; 0]), kron(ones(1,N_Testing),[0;
    0; 1])];

Measured_Answer_Training = zeros(size(Correct_Answer_Training));
```

```
Measured_Answer_Testing = zeros(size(Correct_Answer_Testing));

%% Train linear classifier
W = eye(3, dimx+1);
Alpha = 0.005;
iterations = 0;

tic;
while iterations < 30000
    gradientMSE = 0;
    for k = 1:3*N_Training
         xk = [Tot_Training_Data(k,:)'; 1];
         z = W * xk;
         gk = sigmoidFunction(z);
         tk = Correct_Answer_Training(:,k);
         gradientMSE = gradientMSE + ((gk-tk).*gk.*(1-gk))*xk';
    end
    W = W - Alpha*gradientMSE;
    iterations = iterations + 1;
end
toc

%% Testing linear Classifier
for i = 1:length(Tot_Training_Data)
    x = [Tot_Training_Data(i,:)';1];
    z = W * x;
    g = sigmoidFunction(z);
    [val, class] = max(g);
    Measured_Answer_Training(class, i) = 1;
end

for i = 1:length(Tot_Testing_Data)
    x = [Tot_Testing_Data(i,:)';1];
    z = W * x;
    g = sigmoidFunction(z);
    [val, class] = max(g);
    Measured_Answer_Testing(class, i) = 1;
end

%% Prints and Comparisons
figure(5);
plotconfusion(Correct_Answer_Testing, Measured_Answer_Testing,
    'Test set');
xticklabels({'Setosa', 'Versicolor', 'Virginica'})
yticklabels({'Setosa', 'Versicolor', 'Virginica'})

figure(6);
plotconfusion(Correct_Answer_Training, Measured_Answer_Training,
    'Training set');
```

```
xticklabels({'Setosa', 'Versicolor', 'Virginica'})
yticklabels({'Setosa', 'Versicolor', 'Virginica'})

%% Functions

%Sigmoid function;
function y = sigmoidFunction(z)
    y = 1./(1 + exp(-z));
end
```

7.2 MATLAB; Digits, NN classification

```
outputs = zeros(10, num_test);
targets = zeros(10, num_test);

%% Classifying
tic;
for k = 1:num_test
    targets(testlab(k)+1, k) = 1;
    test_sample = testv(k,:);
    distances = dist(trainv, test_sample');
    [~, closest_distance_index] = min(distances,[],1);
    outputs(trainlab(closest_distance_index)+1, k) = 1;

    if mod(k, 500) == 0
        disp(k*100/num_test + "% done")
    end
end
toc

% save('Data/saveOutputsTask1.mat', "outputs")
% save('Data/saveTargets.mat', "targets")

%% Plot

num_pictures = 16;

incorrect_plot = zeros(num_pictures, 2);
correct_plot = zeros(num_pictures, 2);

a = 0;
b = 0;
for k = 1500:num_test
    output_value = find(outputs(:,k)) - 1;
    if output_value ~= testlab(k) && (a < num_pictures)
        a = a + 1;
        incorrect_plot(a,:) = [k; output_value];
```

```
    elseif output_value == testlab(k) && (b < num_pictures)
        b = b + 1;
        correct_plot(b,:) = [k; output_value];
    end
end

% Incorrectly Classified
figure(1);
sgtitle("Incorrectly classified images")
for n = 1:num_pictures
    x = zeros(28,28);
    x(:) = testv(incorrect_plot(n,1),:);
    x = fliplr(x);
    x = rot90(x);
    subplot(4,4,n);
    image(x);
    Correct = testlab(incorrect_plot(n,1));
    Estimated = incorrect_plot(n,2);
    title("Correct: " + Correct +", " + "Esimated: " + Estimated)
end

% Correctly Classified
figure(2);
sgtitle("Correctly classified images")
for n = 1:num_pictures
    x = zeros(28,28);
    x(:) = testv(correct_plot(n,1),:);
    x = fliplr(x);
    x = rot90(x);
    subplot(4,4,n);
    image(x);
    Correct = testlab(correct_plot(n,1));
    Estimated = correct_plot(n,2);
    title("Correct: " + Correct)
end


figure(3)
plotconfusion(targets, outputs, 'Classification result');
xticklabels({'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'})
yticklabels({'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'})
```

7.3 MATLAB; Digits, NN classification with clustering

```
num_classes = 10;
M = 64;
```

```
outputs = zeros(10, num_test);
targets = zeros(10, num_test);

%% Clustering

[classes, ~, idx_train] = unique(trainlab);
trainv_sorted = splitapply(@(x){x}, trainv, idx_train);

trainlab_cluster = zeros(M*num_classes, 1);
trainv_cluster = zeros(M*num_classes,784);

for i = 1:num_classes
    % Save class of all clusters
    trainlab_cluster(M*(i-1)+1:M*i, 1) = (i-1)*ones(M,1);

    [~, C_i] = kmeans(trainv_sorted{i,1},M);
    trainv_cluster(M*(i-1)+1:M*i,:) = C_i;
end


% save('Data/saveTrainvCluster.mat', 'trainv_cluster')
% save('Data/savetrainlab_cluster.mat', "trainlab_cluster")

%% Classifying
tic;
for k = 1:num_test
    targets(testlab(k)+1, k) = 1;
    test_sample = testv(k,:);

    %calculate distance, find index of closest centroid and check
        its class
    distances = dist(trainv_cluster, test_sample');
    [~, closest_distance_index] = min(distances,[],1);
    outputs(trainlab_cluster(closest_distance_index)+1, k) = 1;

    if mod(k, 500) == 0
        disp(k*100/num_test + "% done")
    end
end
toc

% save('Data/saveOutputsTask2.mat', "outputs")
% save('Data/saveTargets.mat', "targets")

%Confusion matrix and error rate

figure(1)
plotconfusion(targets, outputs, 'Classification result');
xticklabels({'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'})
```

```
yticklabels({'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'})
```

7.4 MATLAB; Digits, KNN classification with clustering

```
num_classes = 10;
M = 64;
K =7;

outputs = zeros(10, num_test);
targets = zeros(10, num_test);

% Clustering is done in DigitClassifierWithClustering.m. Remember
    to run this first.

%% Classifying
tic;
for i = 1:num_test
    targets(testlab(i)+1, i) = 1;
    test_sample = testv(i,:);
    distances = dist(trainv_cluster, test_sample');

    % Finds the K closest classes, uses mode() to get the most
        frequent
    [~, indexes_sorted] = sort(distances);
    K_closest_indexes = indexes_sorted(1:K);
    K_closest_classes = trainlab_cluster(K_closest_indexes);
    Most_frequent_class = mode(K_closest_classes);
    outputs((Most_frequent_class+1), i) = 1;

    if mod(i, 500) == 0
        disp(i*100/num_test + "% done")
    end
end
toc

% save('Data/saveOutputsTaskKNN.mat', "outputs")
% save('Data/saveTargets.mat', "targets")

%Confusion matrix and error rate
figure(1)
plotconfusion(targets, outputs, 'Classification result');
xticklabels({'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'})
yticklabels({'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'})
```