PATTERN RECOGNITION
DELFT UNIVERSITY OF TECHNOLOGY

Final Assignment

Classification of Hand-Written Digits

Jurriaan Govers BSc, 4163753
Sebastiaan Joosten BSc, 4103513
Stijn van der Smagt BSc, 4306686

February 2018

Executive Summary

Samenvatting over report (weet niet of het er in moet?)

Contents

1 Introduction

This report describes the design and performance of a Pattern Recognition algorithm that is able to classify images of hand-written digits on bank cheques. It considers two cases:

- **Case I: The system is trained only one time, and then used in the field.** This means a large amount of training data is available.
- **Case II: The system is trained for each batch of cheques.** This means only a small amount of training data is available.

Although Case II provides significantly less data than Case I (and this definitely has to be taken into account during the design of the classification system) it is worthy to note that both cases call for a supervised learning approach.
The available data will be extracted from the NIST-list, a large dataset containing labelled hand-written digits from 0 to 9. These images will be processed in order to make it easier to handle them. The loading of the list and processing of the images it contains is done in a separate function. This function is descibed in detail in section 2. Implementing the pre-processing in this way enables the inclusion of the image-processing part of the algorithm in a later benchmark test. It also enables us to keep the pre-processing consistent across the two different cases.

even de titel van de m-file noemen (dus hoe de functie heet)

After the processing, two classifiers will be trained, one for each of the aforementioned cases. These classifiers will be stored as w, so that they too can be implemented in the benchmark test. The design of these classifiers is described in detail in section 3.

Finally, for both cases, the benchmark test will be conducted by calling `e = nist eval(filename,w,n)`. The error e this function returns should be less than 5% for case 1 and less than 25% for case 2. An overview of the handling of the data can be seen in figure 1.

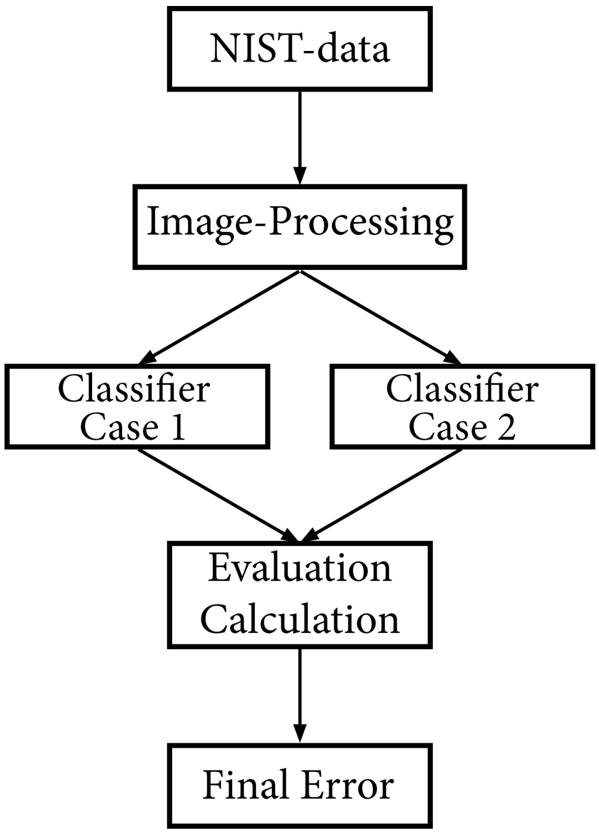

Figure 1: Process of classification of the NIST-data for the two cases.

2 Pre-Processing

Before the classifiers will be trained, some pre-processing is needed. In order to make all images the same size, they were placed within the same bounding box. Next, all images were downsampled to a size of 16 by 16 pixels, to reduce the amount of pixels and save computational time and space.
Then some image processing was implemented. When looking at some random samples of the dataset, it was noticed that some of them contained fragmented digits. Also, some spots and blobs could be observed in some images. When using certain Pattern Recognition tools, this could create various problems. For instance, when using a feature-representation to classify the objects, the feature 'perimeter' will look for the first object in every image, and calculate its perimeter. If a small blob is present in the top left corner of an image, it will calculate the perimeter of this blob, and ignore the actual digit.
In order to overcome this problem relatively easy, two image processing steps are implemented in the pre-processing phase. First, all images were extracted from the dataset and converted to an image using the `data2im.m` in MatLab. This means that all kinds of image-processing techniques become available. Since this course focusses on Pattern Recognition, and not so much on Image Processing, it was decided to keep it relatively simple, and just implement two functions to remove some of the spots and blobs. `bwmorph.m` was used to remove single pixels, and `bwareaopen.m` was used to remove some larger blobs. After this, the images were converted back to data using `im2obj.m`, and stored in a prdataset.
Finally, a small Gaussian filter was used to overcome some 'broken' digits and smooth out some rough edges that originated from the downsampling. While more advanced techniques like image propagation would be more suitable to close for instance sixes and eights, it was decided not to implement this in the interest of time and with the goal of the course in mind. A few examples of the effect the pre-processing has on some digits can be seen in figure 2.

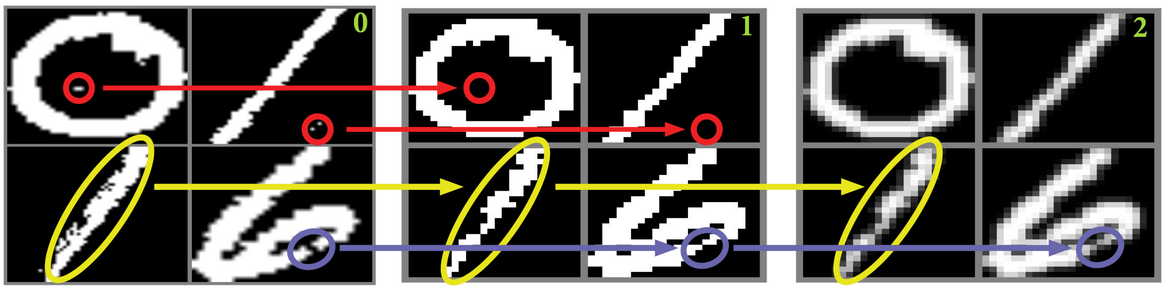

Figure 2: Effect the pre-processing has on certain defects in the images. Step 1 removes single pixels and small blobs using `bwmorph.m` and `bwareaopen.m`. Step 2 adds a gaussian filter, to reconnect broken digits and smooth out any rough edges.

3 Classifier Design

As stated in the introduction, a classifier had to be build for two different cases. For both cases, a similar design approach was used.

1. Investigate amount of data and use the theory to choose an initial representation method
2. Train simple classifiers based on the chosen representation method
3. Perform evaluation
4. If the performance does not meet the requirements, go back to the theory
5. If the performance meets the requirements, optimise the system

This approach is visualised in figure 3.

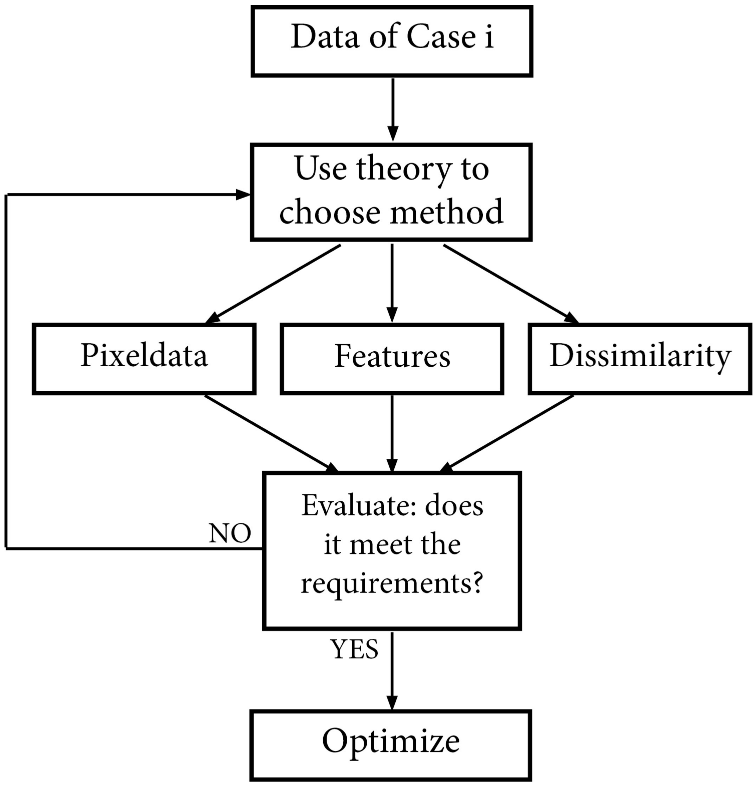
Data of Case i
Use theory to choose method
Pixeldata
Features
Dissimilarity
Evaluate: does it meet the requirements?
NO
YES
Optimize

Figure 3: Design-process to determine the best representation method for each of the two cases described in the Introduction.

The image processing described in chapter 2 was kept the same for both cases, since it is a very subtle processing and will benefit all potential methods.

3.1 Case 1

Case Summary: Design a system for training on all cheques at once. The system must be able to correctly classify the handwritten digits after being trained on at least 200 objects per class.

This case has a lot of data available. A wise choice to start would be to use a pixeldata representation, potentially in combination with some form of feature reduction.
Smag

3.2 Case 2

Case Summary: Design a system for training on each batch of cheques to be processed. The system must be able to correctly classify the handwritten digits after being trained on a maximum of 10 objects per class.

The amount of data available in this case poses some restrictions. While a pixel-data representation worked very well in Case 1, it is likely to have too little data to work with in this case. Some more prior knowledge is needed to handle these batches of cheques. One way of doing this is by letting the system recognize certain distinct properties of the digits. A straight-forward way of doing this is by using a feature representation of the objects. A more abstract way is using a dissimilarity representation.

Representation by Features

To start out, features were chosen as the best representation for this case. The function `im_features` from the prtools-toolbox was used to compute 14 different features of all the images in the preprocessed database. Because the amount of features is relatively high compared to the amount of data that is used, some form of feature reduction needs to by applied. MatLab's forward feature selection (`featself`) was used initially to get a sense for the features that should be used and the features that could be omitted. This was done for four cases; with and without scaling of the data, and with and without implementing a separate test-set in the `featself` function. The resulting performance can be seen in figure 4

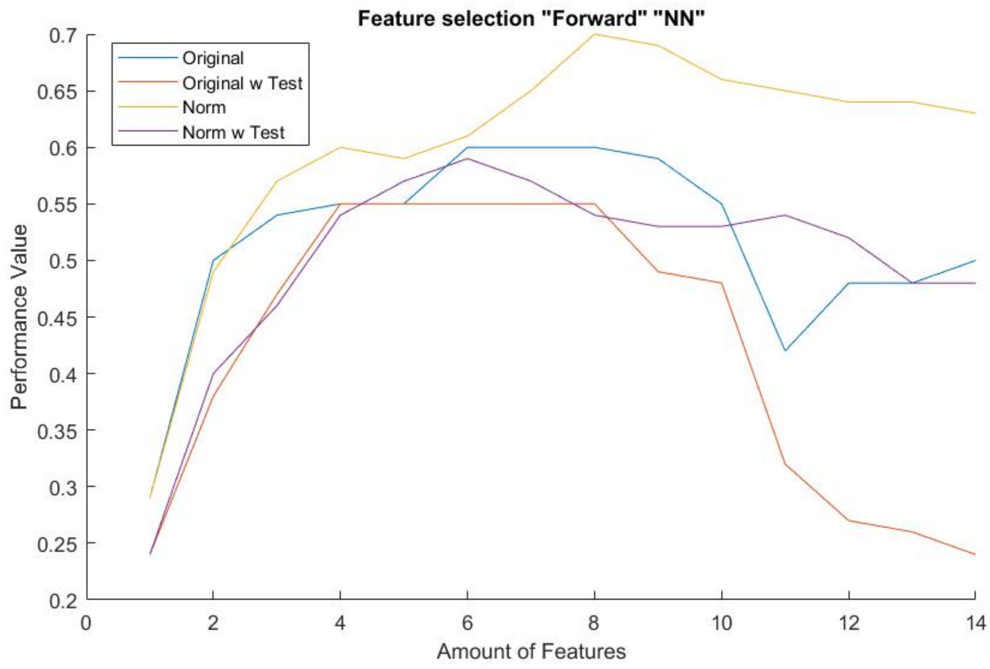

Figure 4: Performance of features selected by forward feature selection. Both scaling and non-scaling, and with or without test-set are shown.

As can be seen in the figure, there are a lot of differences between the different parameters. There appears to be an optimal amount of features however, somewhere between 4 and 12. In order to get a better grip on which features to use, different feature selection strategies were compared. After the forward selection, the backward and the plus-2-takeaway-1 were tested. This is was done for both the '1-Nearest Neighbor' and the 'Summed Mahalanobis distances' criterion. All these methods indicated that there was an optimum somewhere between 4 and 12 features. However there seemed to be no correlation between which of the features belonged to this optimal set. Therefore a brute

Ik heb hier nog bij gezet dat ik ook andere dingen dan forward heb force approach was taken. Each possible combination between 3 and 14 features was used for an error-calculation (using the `testc` function of PRTools) of four different classifiers (Fischer's Classifier, Linear Classifier, Nearest Mean Classifier and a Nearest Mean Classifier trained on normalized data). After running this setup overnight, the following results were obtained: (see figure 5).

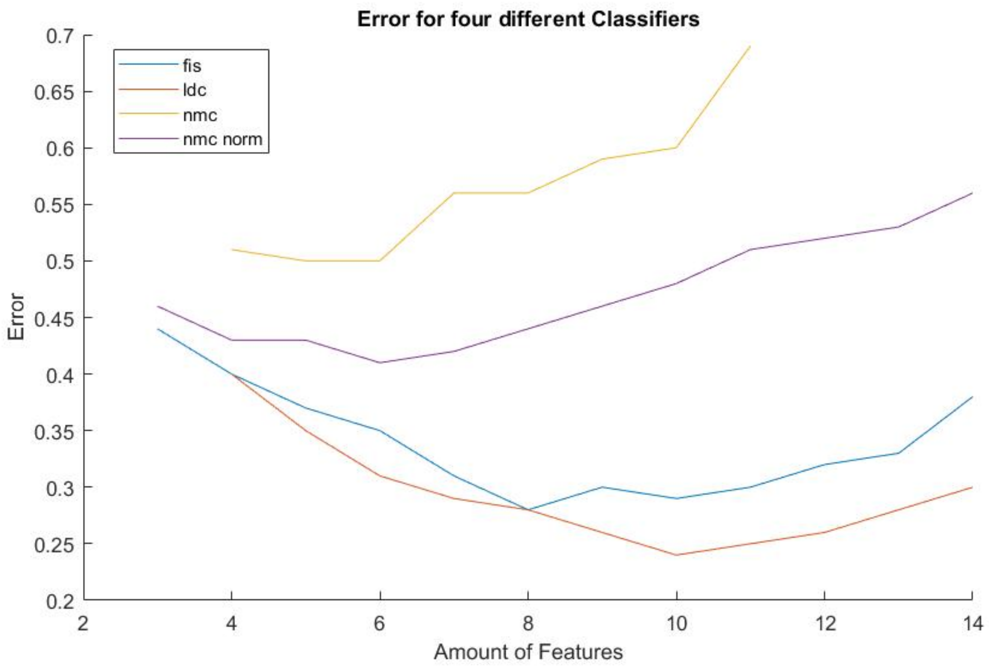

Figure 5: Error of different combinations of features for four different trained classifiers; Fischer's (`fischerc`), Linear Bayes-Normal (`ldc`), Nearest Mean (`nmc`) and a normalised version of Nearest Mean.

This figure shows the lowest error obtained with a certain amount of features. It suggests the optimal amount is 10. It also shows that the Linear Classifier will probably perform best, although even its best error is not under the 25 percent yet.
Next, for the LDC classifier only, these 10 features were tested on different datasets. The result of this was a big fluctuation in errors between datasets, which implies that the set of 10 features were only optimal for one particular dataset. Therefore another brute force approach was used to test all possible combinations of 9 to 12 features on different datasets. In figure 6 the error of every combination of 9 features can be seen. Very noticeable are the large drops and jumps in the graph, visualised with the orange line below it. After close inspection of the used features at these points, it was discovered that every increase in error corresponded to the same four features being added, namely 'Convex Area', 'Convex Hull', 'Convex Image' and 'Eccentricity'. A similar pattern was recognized in the other results. It was decided to omit these features, leaving the final dataset with the desired amount of 10 features.

Vanaf hier heb ik de tekst een beetje veranderd

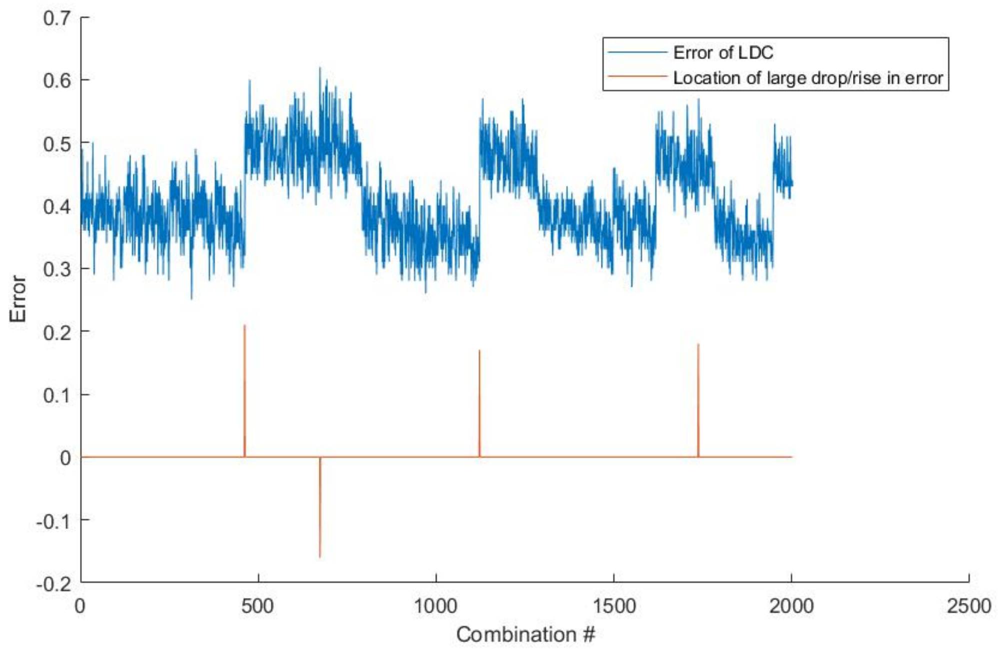

Figure 6: Error of the LDC-classifier for each possible combination of 10 out of 14 features. The orange line indicates places with a large increase or decrease in error-value.

At this time, the complete system consisted of a Linear Classifier (LDC) trained on the dataset from which 10 features were selected. Sadly though, testruns of the algorithm yielded an error of 30%. Since this was too low to meet the requirements, a different approach was taken.

Representation by Dissimilarity

It was decided that more information needed to be intrinsic in the system in order for it to be able to correctly classify the digits with the small trainingset Case 2 provides. Going back to the theory, dissimilarity-representation seemed to be able to do this. By picking a few representative examples of each digit by hand (the prototypes) and calculating the distance of the remaining objects to these prototypes, correct classification should be possible.
To get a first test of this principle, it was tried to first filter out all some digits. 90 samples per class were taken, two representative ones were chosen, and the distance between all objects and these three zeros was calculated using MatLab's `proxm` function.

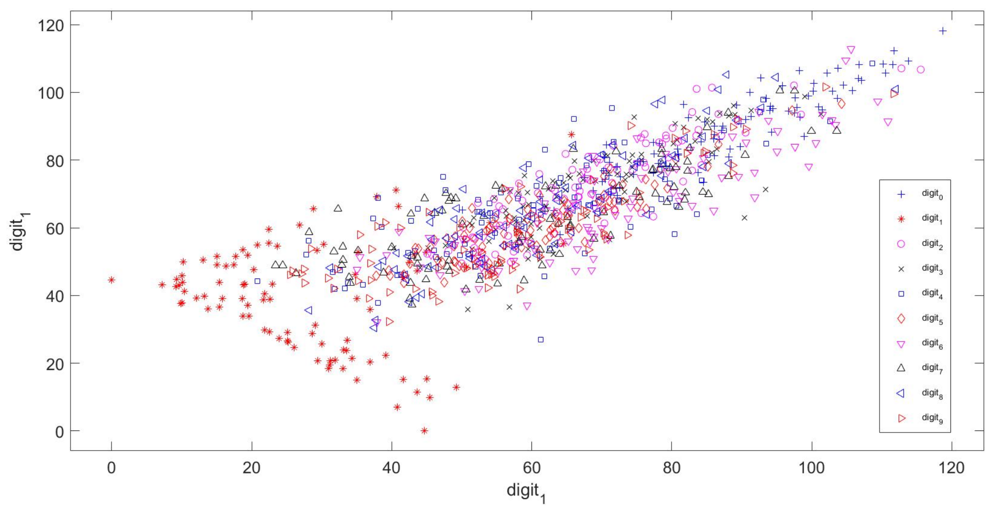

Figure 7: Dissimilarity of 90 samples per class to two digit-one prototypes, using city-block distances.

In figure 7 a clear grouping of all the ones can be observed, and this group can be separated from the rest of the objects easily. This was done for various distance measures. As can be seen in figure 8 a few distance measures perform comparable, but city-block seemed to be the most consistent over different training sets.

VERWIJ NAAR PAPER, en zeggen dat het in image goed werkt

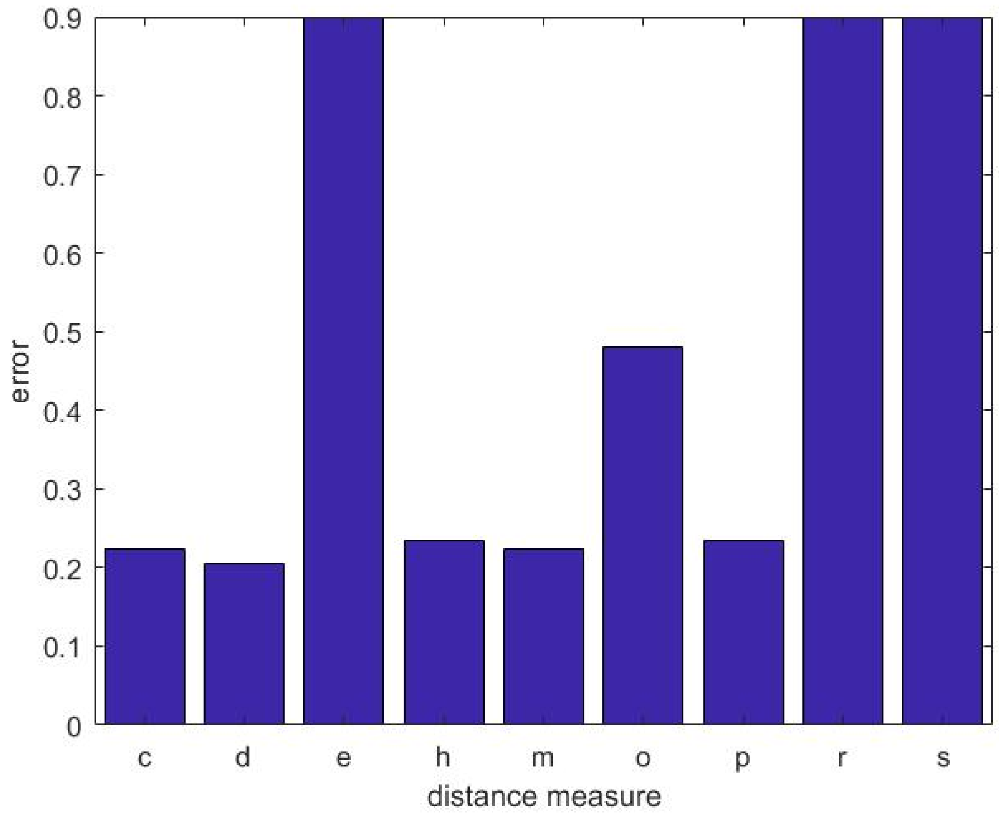

Figure 8: Error of `ldc` classifier, using different distance measures.

The next step is to try this with every digit. Ten samples per class were taken, and in stead of selecting the prototypes manually, all objects were given as prototypes. This meant a dataset of 100 samples, each of them having 100 features. Now a decision must be made on what classifier to use. Because of the high dimensionality of the feature space and the relative low amount of samples, the more complex classifiers tend to

fuck up.

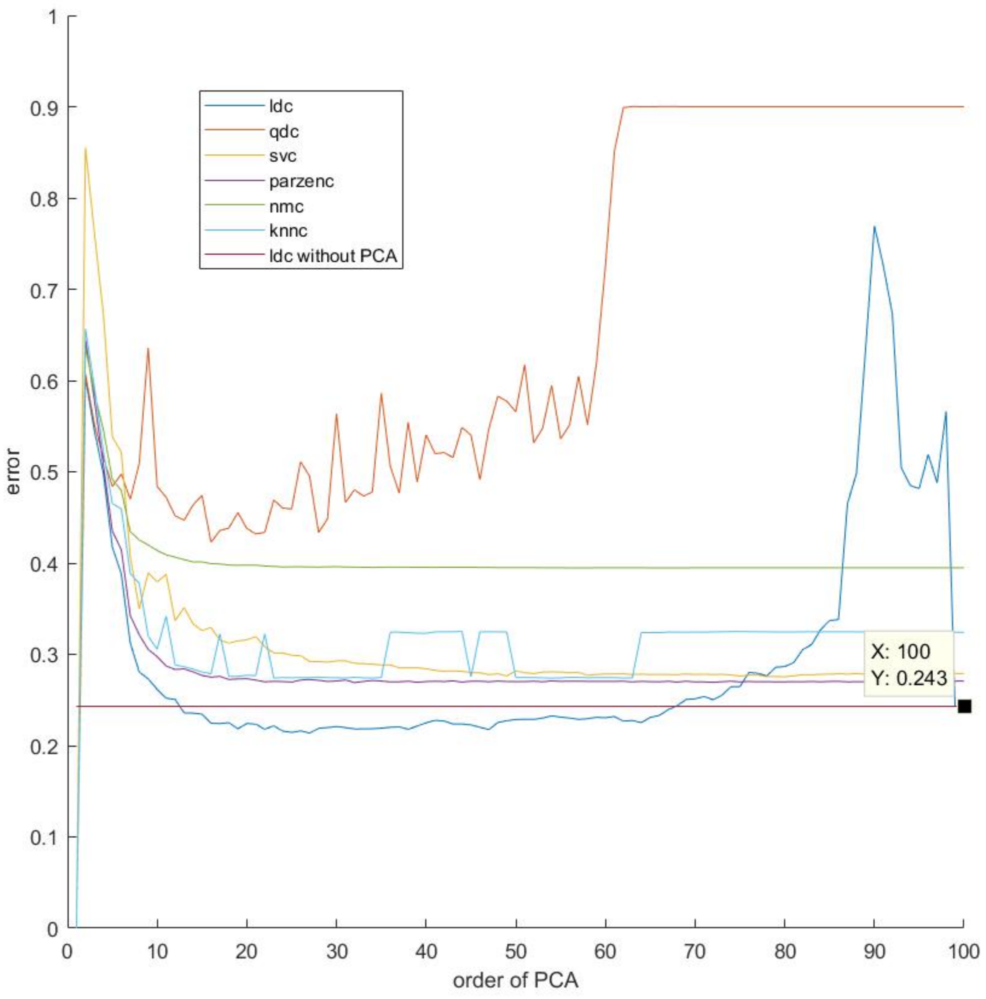

Figure 9: Error of different classifiers, using different orders of PCA.

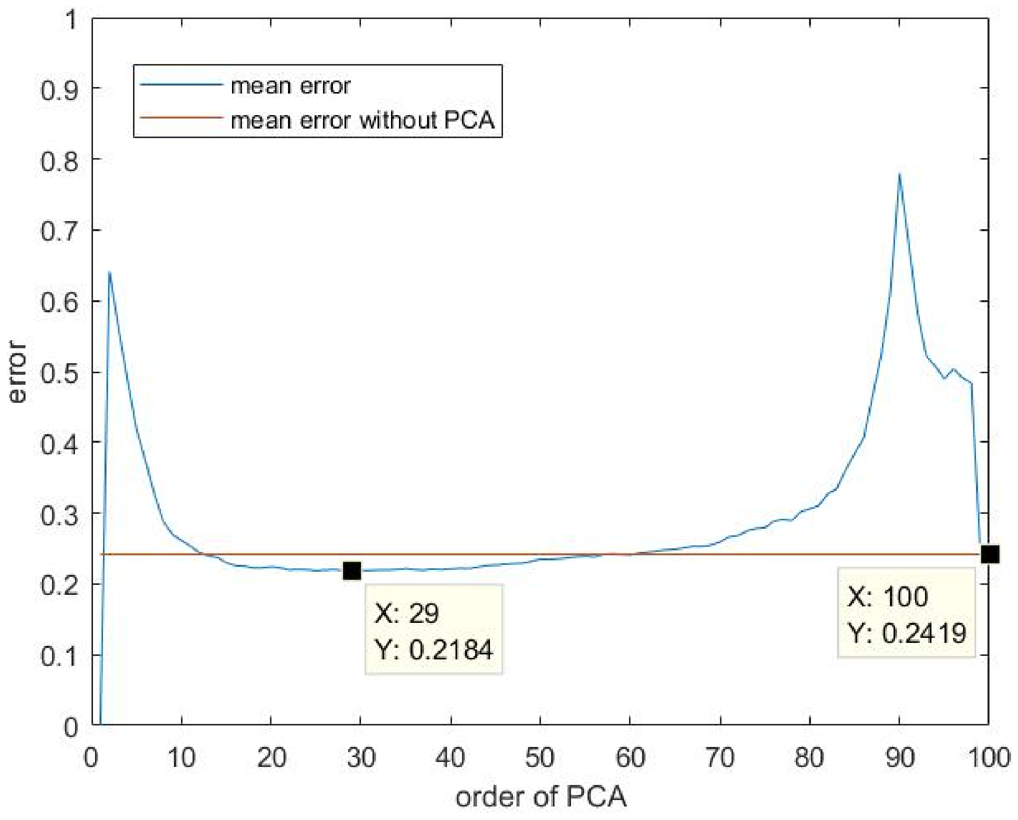

Figure 10: Average error of ten test stets of `ldc`.

4 Discussion and Conclusion

Sebas en Allemaal

Include a section Recommendations to your report, in which you advise your client in detail on possible steps to improve performance in light of the overall system they are constructing.
For example:
will having more training data help?
would other features help?
does the application require reject options?
does a single classifier suffice for all digits on a bank cheque?
what is the role of time constraints?