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

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 1: The system is trained only one time, and then used in the field.** This means a large amount of training data is available.
- **Case 2: The system is trained for each batch of cheques.** This means only a small amount of training data is available.

Although Case 2 provides significantly less data than Case 1 (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, named `myrep.m`. This function is described 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 the user to keep the pre-processing consistent across the two different cases if desired.
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.
Both classifiers will be evaluated using various error estimations. This way, some indication of their performance can be made before sending them to the client.
Finally, to simulate the evaluation by the client for both cases, the benchmark test will be conducted by calling `e = nist_eval(myrep,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 as described in this section can be seen in figure 1.

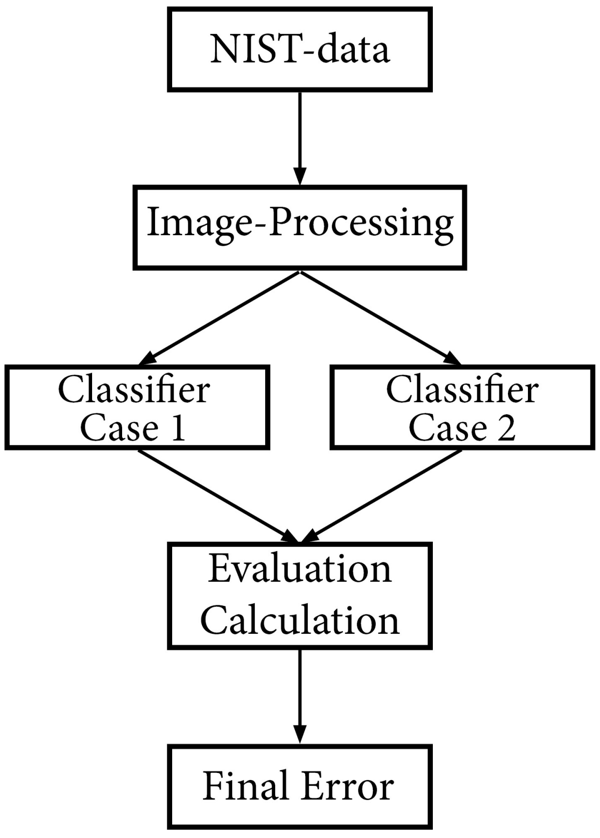

Figure 1: Process of classification of the NIST-data for the two cases.

2 Pre-Processing

Before the classifiers will be trained, some pre-processing is needed. In order to make all the images of the handwritten digits the same size, they were placed within the same bounding box. Next, all images were sampled down 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 random samples of the dataset, it was noticed that some of them contained fragmented digits. Also, 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` function 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.

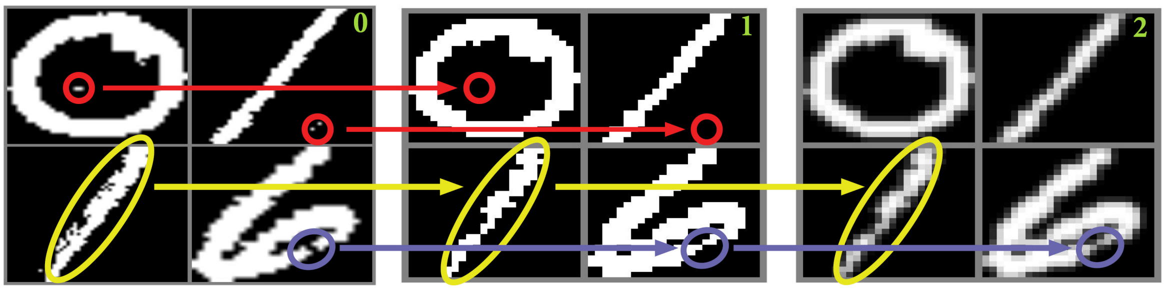

Figure 2: Effect the pre-processing has on certain defects in the images. Step 1 removes single pixels and small blobs using `bwmorph.m` and `bwareaopen.m`. Step 2 adds a gaussian filter, to reconnect broken digits and smooth out any rough edges.

3 Classifier Design Process

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

1. Investigate the 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.

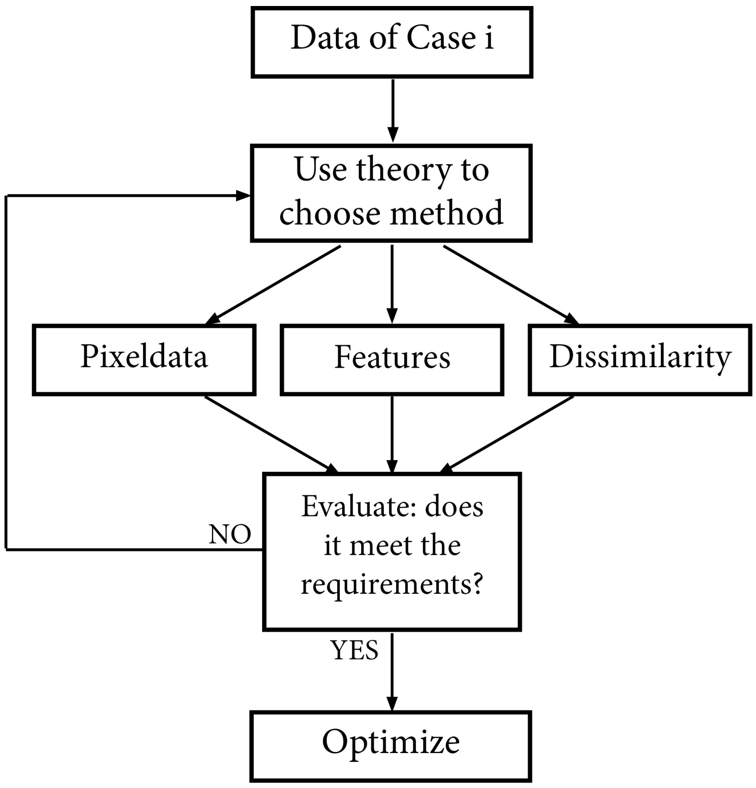
Data of Case i
Use theory to choose method
Pixeldata
Features
Dissimilarity
NO
Evaluate: does it meet the requirements?
YES
Optimize

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, with exception of the blob removal in Case 1, because it would take up too much time (see section 4 and the recommendations in section 7 for a more detailed explanation). Furthermore, it is expected that, for the final representation chosen for this case (representation by pixels), the presence of blobs and spots won't influence the performance that much. The rest of the pre-processing is a very subtle processing and will benefit all potential methods.

4 Case 1

Case Summary: Design a system for training on on the complete dataset, to be applied in the field afterwards.

In this scenario the classifying system is trained once, and then applied in the field. This means that all available data can be used as training input, which results in 1000 objects per class, and 10.000 objects in total. (1000 zeros, 1000 ones, etc.)
Per representation method, the best classification option will be investigated. For this case, all error rates of the different classifying algorithms were estimated using cross-validation using 10 folds.

4.1 Representation by Pixels

Density Estimating Classifiers

Large datasets facilitate classification using non-parametric density estimating classifiers. Classifiers of this type are known to perform very well when large datasets are available. After the image preprocessing was carried out as explained in chapter 2, several density estimating classifiers were tested: Parzen density estimating classifier, which optimizes its 'width' parameter iteratively, and k-nearest neighbours classifiers for k = 1,2,3. Although these classifiers are sensitive to uneven scaling of different features, rescaling of feature vectors is not necessary, because all pixel values are within the same range. The results of these classifiers can be seen in table 1. The images used have a dimension of 16x16 pixels, which results in a 256 dimensional feature-space. The error rates for these classifiers pass the threshold of 5% already. To investigate the best and worst areas of these classifiers, the error per class was plotted The error per digit class can be observed in figure 4. This shows that errors in '8' and '9' are high for all classifiers, as well as the error for '3', '4' and '5' to a lesser extent.
For reference, a quadratic discriminant classifier (QDC) was added to the comparison (see table 1 and figure 4), which is parametric in contrast to the non-parametric classifiers mentioned above. The QDC performs significantly worse than the other classifiers, because it suffers more from the 'curse of dimensionality'. That is to say that the performance of the classifier benefits from adding more features only up to a certain point, after which the error will rise with extra features added. The 'curse of dimensionality' implies that reducing the number of features might decrease the error attained by the classifiers.

Table 1: Overview of the Density-Estimating classifiers and their estimated error.

Classifier	Parzen	1-NN	2-NN	3-NN	QDC
Estimated error	0.0277	0.0280	0.0343	0.0321	0.2125

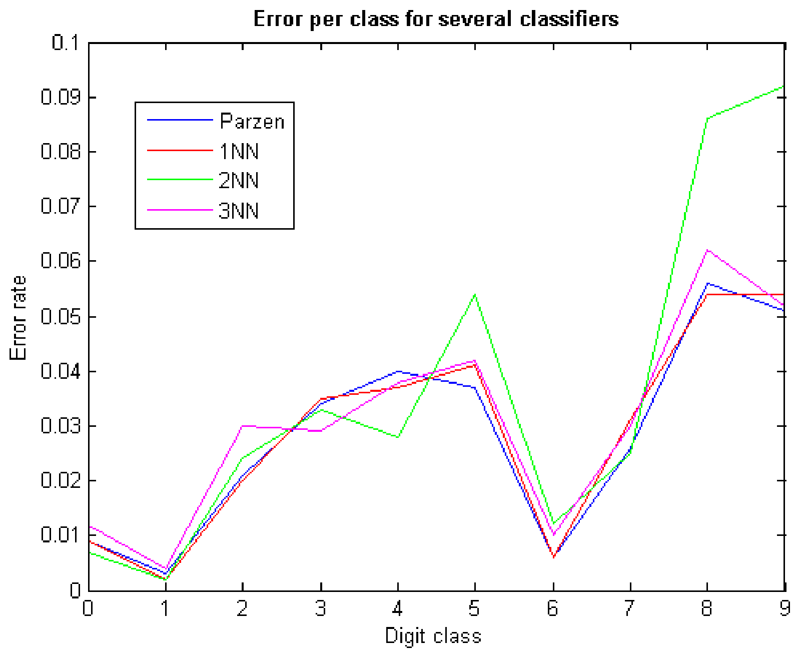

Figure 4: Error per class for different non-parametric density-estimating classifiers.

Feature Reduction

Because of the large number of features in the original data set (16x16=256 pixel values) and large number of objects, any feature selection method, especially backwards propagating, will take a significant amount of time. For this reason feature selection methods were discarded, and feature extraction methods were investigated further.
The most frequently used feature extraction technique is Principal Component Analysis (PCA). In the PCA-algorithm, it is possible to indicate how much of the total variance should be preserved in the newly created features. The cross-validated error was calculated for the same classifiers as above after applying PCA using values of 0.90 to 0.95 of variance preservation. This range was chosen as it corresponds to the intrinsic dimensionality. Results can be seen in table 2. It is remarkable that the performance of the Parzen classifier deteriorates dramatically, while the performance of the 1-NN classifiers decreases only slightly, and the QD-classifier improves a lot, benefiting from the reduced dimensionality. Note that PCA is an unsupervised method, or in other words: it does not take the class labels or resulting performance of the classifiers into account when choosing new features. This explains the decreased performance of some of the classifiers after using PCA.
PCA is sensitive to the scaling of the features. Therefore it is recommended to normalize the variance of the features before applying PCA. In the case of pixel values, however, the data is inherently normalized, because all feature values are in the same range. Applying variance normalization will actually worsen the performance in this case, because it enhances the noise coming from pixels with very low variance. For example: the pixel in the top-left corner will practically always be black, so it has a very low variance. Rescaling will enhance the variance in this pixel, which has unwanted effects.

In figure 5, the (significantly improved) performance per class for the QD-classifier after applying PCA is compared to the performance as seen in figure 4. It is noteworthy that the values of the QDC curves differ dramatically from the curves for the non-parametric classifiers without dimension reduction. For some classes QDC performs significantly better, while for others the ranking is inverted. This fact can be exploited using classifier combining.

Table 2: Overview of the performance of the classifiers after applying PCA, with different values for the preserved variance.

Preserved variance	0.90	0.91	0.92	0.93	0.94	0.95
Resulting dimensionality	41	44	48	53	58	65
Parzen	0.1122	0.1125	0.1134	0.1136	0.1159	0.1176
1-NN	0.0304	0.0305	0.0314	0.0292	0.0297	0.0297
QDC	0.0354	0.0354	0.0375	0.0379	0.0383	0.0361

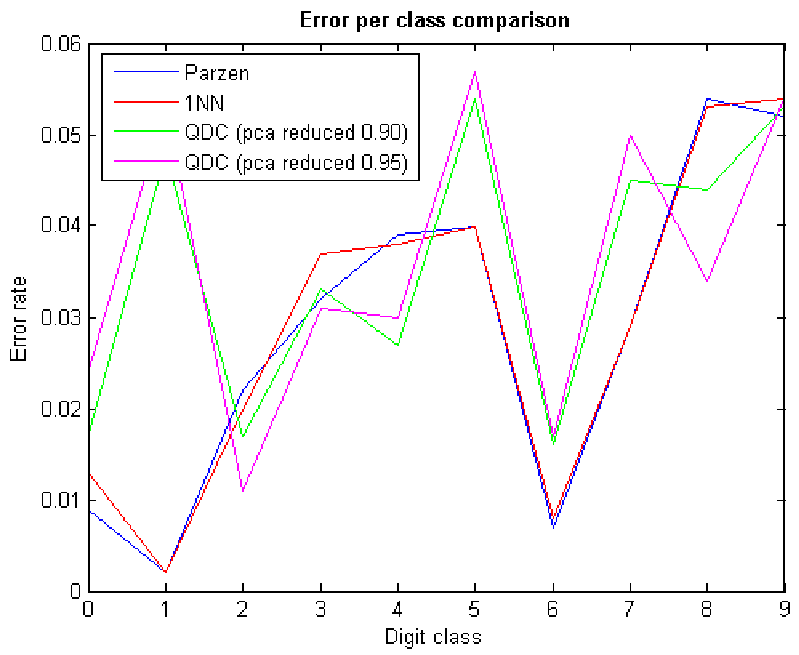

Figure 5: Comparison of the error per class for different classifiers.

Combining

Since the different classifiers in figure 5 perform well for different digit classes, the one alternately performing better than the other, it might be beneficial to combine them. Combining can be carried out using different combining rules. The following combining rules will be investigated: product rule, max rule, mean rule, median rule, min rule. Parzen is to be combined with PCA + QDC, for two different values of variance preservation, as is 1-NN. The result can be seen in table 3. The combination using the product rule of Parzen and QDC (after application of PCA preserving 90% of the variance) offers the best result. The error rate per class can be seen in figure 6. Except for the digits '0' and '1' (for which Parzen performs best), the combination performs better than the two original classifiers. The '9' is still the digit with the highest classification error.

Table 3: Estimated error for different combinations of classifiers, using different combination rules.

Combining rule	Product	Max	Mean	Median	Min
Parzen + QDC90	0.0222	0.0271	0.0263	0.0276	0.0333
Parzen + QDC95	0.0225	0.0274	0.0276	0.0267	0.0367
1-NN + QDC90	0.0361	0.0281	0.0282	0.0285	0.0354
1-NN + QDC95	0.0382	0.0298	0.0281	0.0281	0.0369

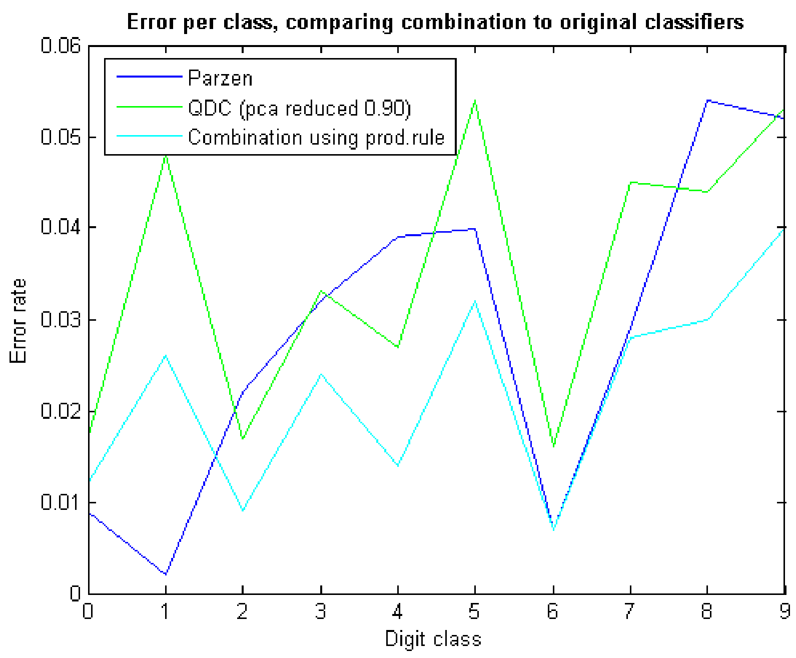

Figure 6: Overview of the error per class of the two original classifiers and their optimal combination.

Support Vector Machines

Support Vector Machines (SVM) are known to be especially useful for classifying in high dimensional feature spaces, which is a major advantage when analysing pixel data. However, the lengthy runtime of the SVM-algorithm is a major disadvantage. Because of the limited time available to analyse the different classifiers, the SVM-algorithm was discarded as a viable option. Once the SVM is created, the runtime of using it to classify and to test it is relatively short. But tweaking and testing it over and over would have taken a significant amount of time. Furthermore, enough training data is available to make adequate density estimations, as is supported by the previous tables and figures.

Preliminary Conclusion

for Case 1 with a representation by pixels, the combined classifier (product rule) of Parzen and QDC (using PCA to reduce the dimensionality to 41) gives the best performance, with an error of 0.0222.

4.2 Representation by Features

At first glance, the feature representation approach is not the most suitable for scenario 1. Given a large dataset, the performance of the classifiers will benefit from a relatively large number of features. The image analysis methods included in the PRToolbox give 10 features we deemed useful (see Case 2 for a more in-depth discussion of this choice). A number which is predicted to be insufficiently high for achieving an error rate comparable to the rates accomplished using the pixel data approach. Furthermore, because the blob-removal discussed in 2 was not implemented for the large dataset (due to a too large computation time), this method might perform sub-optimal.
In order to verify this prediction some indicative tests were carried out, using various classifiers on the dataset containing the extracted features. The tested classifiers are: nearest mean classifier (NMC), Parzen density estimating classifier, 1-nearest neighbour classifier, Linear Discriminant Classifier (LDC), logistic classifier (LOGLC), and quadratic discriminant classifier. In table 4 the results of these test are shown. For the relevant classifiers (NMC, Parzen, 1-NN) the features were normalized before attempting classification. The resulting error rates are significantly higher than the errors attained above, and have no prospect of improving by feature reduction, because the original dimensionality is low.

Table 4: Overview of the estimated error for different classifiers based on a feature-representation.

Classifier	NMC	Parzen	1-NN	LDC	LOGLC	QDC
Estimated error	0.4454	0.3472	0.2032	0.2178	0.1863	0.3207

Preliminary Conclusion

The hypothesis is accepted as true, and the feature representation approach is discarded for the large dataset case. The best performing classifier is a Logistic Classifier, with an error of 0.1863. This is not sufficient to surpass the pixel-representation. The feature-representation approach will be dealt with in some more depth in scenario 2, later in this report.

4.3 Representation by Dissimilarities

While the main focus for scenario 1 was on the pixel data representation, a brief amount of time was spent on looking at a dissimilarity-representation for the NIST-digits. The dissimilarity-representation offers a better outlook than the feature-representation, because it performs better in higher dimensionalities. In the dissimilarities approach, ‘distances’ to certain ‘prototype’ digits are used as a means of classification. Both the choice of distance measure and the choice of prototype influence the error rate of the resulting classifier. To be able to quickly test the effectiveness of the dissimilarity representation, only the euclidean distance measure was used on case 1. Results are averaged over 3 randomly sampled sets of 10 prototypes per class. The samples remaining in the dataset were used for training (990 samples per class). The results of this quick experiment can be seen in table 5. The tested classifiers match those tested for the feature representation.

Table 5: Overview of the estimated error for different classifiers based on a dissimilarity-representation.

Classifier	NMC	Parzen	1-NN	LDC	LOGLC	QDC
Estimated error	0.3487	0.0567	0.0638	0.0741	0.0510	0.0452

Preliminary Conclusion

The results have improved compared to the feature representation approach, but not exceeded the results of the pixel data approach. PCA was applied to reduce the dimensionality, but this had an adverse effect in this case. The best performing classifier for this representation was a QDC, with an estimated error of 0.0452.
To improve results, representative prototypes (a set of very averagely written digits) could be chosen, instead of picked randomly, and alternative dissimilarity measures could be investigated (see the Discussion in section 6).

4.4 Final Conclusion and Evaluation for Case 1

As can be seen in table 6, the combination of a Parzen classifier and a QD-classifier applied after PCA, combined using the product rule, yields the best result for scenario 1.

Table 6: Overview of the best performing classifiers per representation, and their errors.

Representation	Best Classifier	Estimated Error
Pixeldata	Combination of Parzen and QDC	0.0222
Features	Logistic	0.1863
Dissimilarities	QDC	0.0452

The error of the chosen combined classifier was determined more accurately using cross-validation employing 10 folds, and 10 repetitions. This way some measure of stability can be given, using the standard deviation between the repetitions. The cross-validation yields an error rate of 0.0219 with standard deviation 5.3125e-4. The standard deviation is very small, which means the classifier is stable for different test sets (picked at random from the original dataset).
The benchmark test (using `nist_eval.m`) yields an error rate of 0.0260 for the chosen classifier for scenario 1. This error is below the desired threshold of 5%, but is slightly higher than the error found using cross-validation. This might be a result of different class frequencies in the evaluation set compared to the training dataset. In the test set all classes were represented equally often, but in the evaluation set a more realistic class distribution could have been used, increasing the total error rate if this digit has a larger than average misclassification rate. It would not be strange if the frequency of '9' (which has the highest error rate for the used classifier) occurring on bank cheques was slightly higher in practice, due to prices of products often looking like -.99. Note that this last remark is speculation, as there is no way to verify the contents of the evaluation set.

5 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.

5.1 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 pre-processed 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. MATLABS'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 7

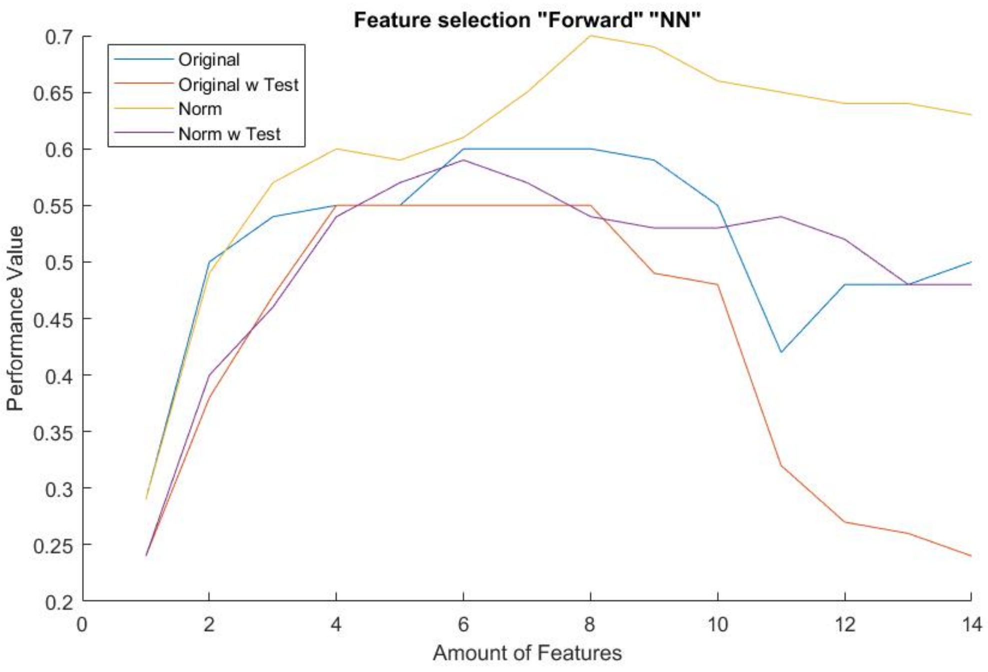

Figure 7: Performance of features selected by forward feature selection. Both scaling and non-scaling, and with or without test-set are shown.

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 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 8).

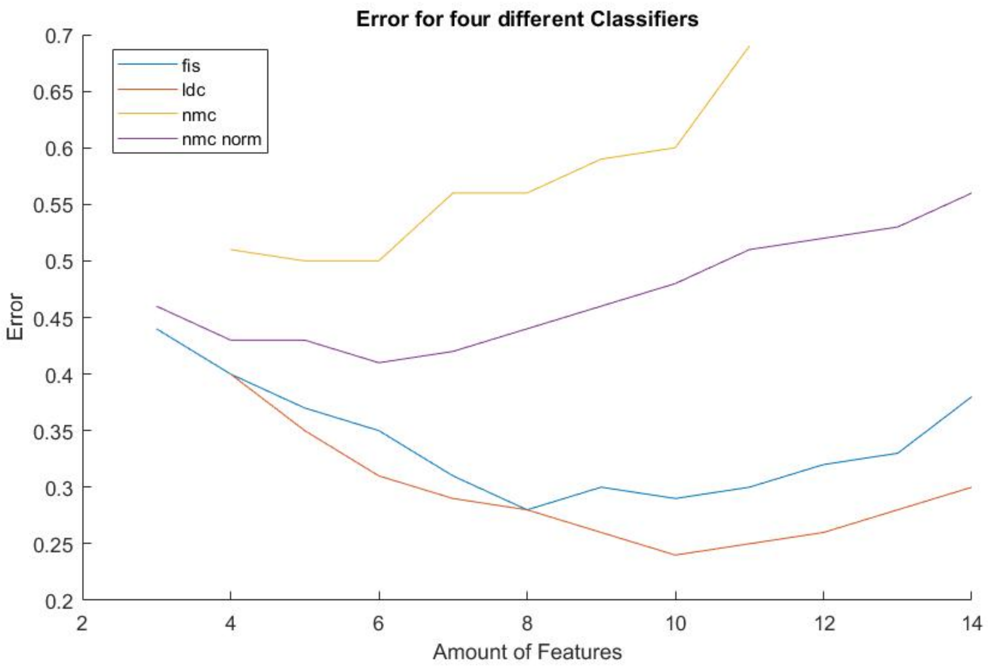

Figure 8: Error of different combinations of features for four different trained classifiers; Fischer's (`fischerc`), Linear Bayes-Normal (`ldc`), Nearest Mean (`nmc`) and a normalised version of Nearest Mean.

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, using these 10 features the LDC was trained on different training sets. 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 9 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', 'Conves 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.

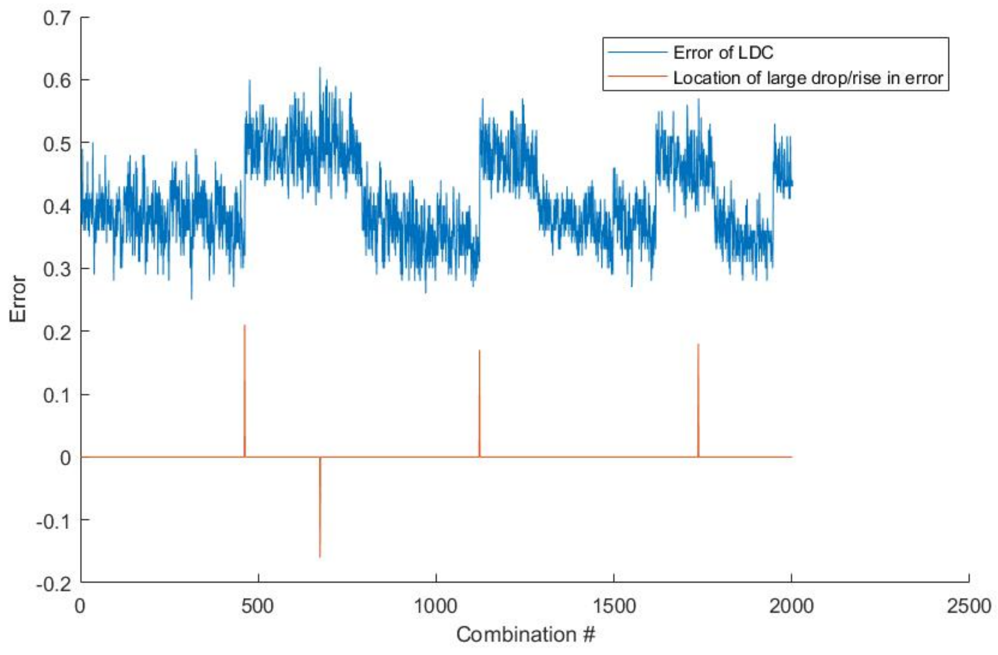

Figure 9: Error of the LDC-classifier for each possible combination of 10 out of 14 features. The orange line indicates places with a large increase or decrease in error-value.

Preliminary Conclusion

At this time, the complete system consisted of a Linear Classifier (LDC) trained on the dataset from which 10 out of 14 features were selected. Sadly though, test runs of the algorithm yielded an error of 30%, as can be seen in figure 9. Since this is too high to meet the requirements, a different approach was taken.

5.2 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 training-set Case 2 provides. Going back to the theory, dissimilarity-representation seemed to be able to do this. By picking a certain amount of 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 an initial idea of the performance of this method, it was first tested whether or not it performed well on the classification of a single class. 90 samples per class were taken, two representative 'ones' were chosen, and the distance between all objects and these two 'ones' was calculated using MATLAB's `proxm` function.

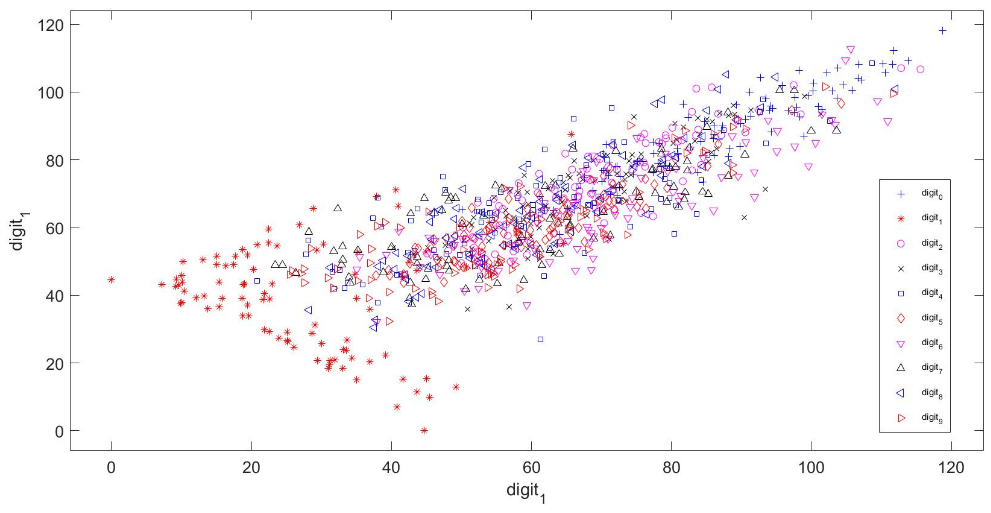

Figure 10: Dissimilarity of 90 samples per class to two '1' prototypes, using city-block distances.

In figure 10 a clear clustering of all the 'ones' can be observed. This cluster can be separated from the rest of the objects easily using a linear classifier (for instance LDC).
The next step is to get a sense of the optimal distance measure to use with `proxm`. The estimated error of the LD-classifier was determined for various distance measures. As can be seen in figure 11 a few distance measures perform comparable. City-block seemed to be the most consistent over different training sets, so this distance measure was chosen. [1]

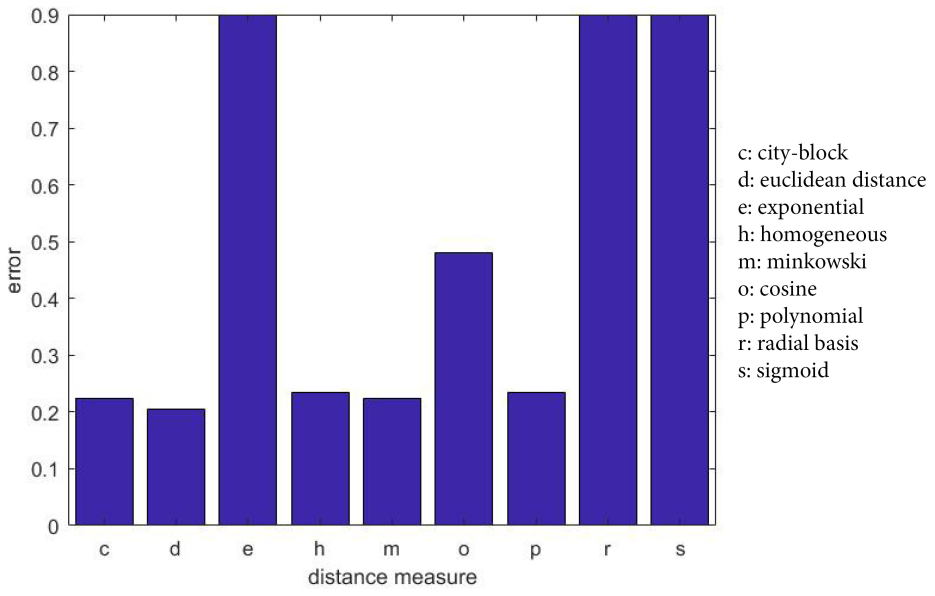

Figure 11: Estimated error of `LDC` using different distance measures.

This method now has to be scaled up for every class. At the same time, it was realised that manually picking prototypes in Case 2 might not be the best option. This manual selection has to be done for each batch of cheques, through some sort of Graphical User Interface. Because it was assumed that all prototypes should be selected again for each new batch, it was decided to give all objects as prototypes in order to prevent having to code some kind of user interface to manually select prototypes. The resulting high-dimensional feature space would be reduced by means of PCA. Ten samples per class were taken. This resulted in a dataset of 100 samples, each of them having 100 features. Initially, LDC was chosen as the classifier separating the 'ones' from the other digits, but for all classes there might be an other optimal one. Different classifiers were trained after using PCA to reduce the dataset to a wide range of dimmensionalities (dimensionality of 100 means keeping 100 features, so essentially not performing PCA). The results can be seen in table 7 and in figure 12.

Table 7: Estimated errors for different classifiers, with and without the use of PCA. In the case of PCA, the dimension in which the lowest error is attained is given in the last column.

Classifier	Without PCA	With PCA	Dimensionality PCA reduces to (lowest error)
LDC	0.243	0.2137	27
QDC	0.9	0.4228	16
SVC	0.2789	0.2754	80
ParzenC	0.2707	0.2691	32
NMC	0.3948	0.3947	69
kNNC	0.3242	0.2739	54

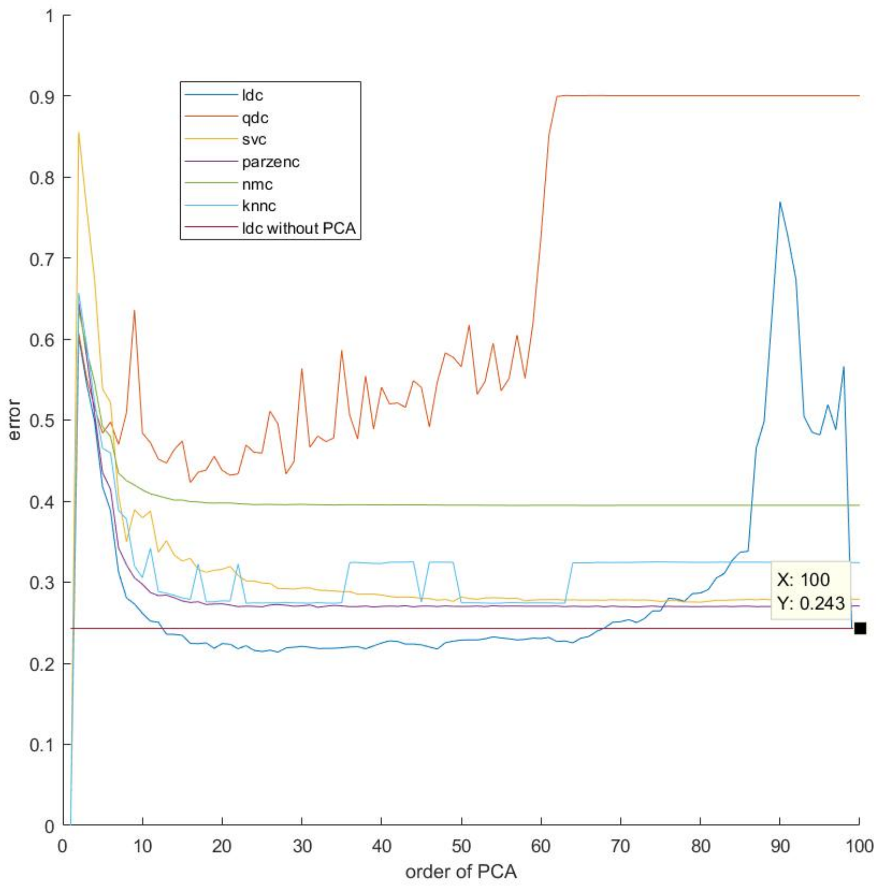

Figure 12: Estimated error of different classifiers, using PCA resulting in different amounts of dimensions.

Because of the high dimensionality of the feature space and the relative low amount of samples, the more complex classifiers tend to over-train. This supports the earlier choice of a linear classifier. LDC seems to give the lowest error, both with and without PCA. To find out which one of these performs best, both were trained 10 times on a randomly picked dataset of 10 objects per class, after which they were tested on the remaining 990 objects per class. The results for LDC including all feature reductions (using PCA) and the unaltered LDC are shown in figure 13.

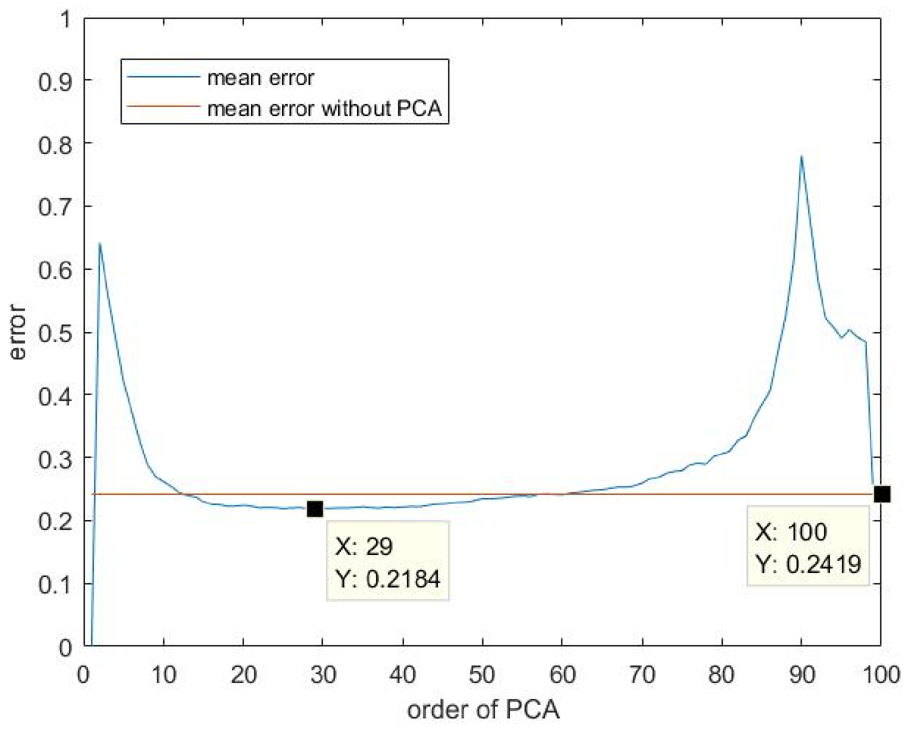

Figure 13: Average error of ten test stets of `ldc`.

Preliminary Conclusion

The best performing Classifier using a dissimilarity-representation is a Linear Discriminant Classifier, trained on a dissimilarity-set from which 29 features are extracted using PCA. This classifier has a mean error of 0.2184 with a standard deviation of 0.0153.

5.3 Representation by Pixels

Using the pixel representation results in a high dimensionality (16x16=256). The combination of high dimensionality and low training set size suggests that usage of density estimating classifiers can be ruled out in advance. More promising is the use of support vector classifiers, a nearest mean classifier, and perhaps a linear classifier. In table 8 the estimated error of 5 classifiers is shown. This error was averaged over 10 training sets of 10 objects per class, and tested using a test set containing the other 900 objects of each class. The standard deviation over different training sets is shown in the table as well. The SVC does not perform as well as expected, maybe another kernel would yield a better result.

Table 8: Estimated error of five classifiers for a pixel-representation. The shown error is the average of 10 runs.

Classifier	NMC	SVC (linear)	SVC (polynomial kernel 2)	SVC (polynomial kernel 3)	FisherC
Error	0.2676	0.3203	0.3245	0.3303	0.5313
Standard deviation	0.0182	0.0433	0.0499	0.0325	0.0398

The huge dimensionality compared to training set size calls for feature reduction. Using PCA preserving 90% of the variance, the dimensionality was reduced to approximately 20. The results can be seen in table 9. This decreases the error for the NMC slightly, increases the error for the SVC, and decreases the error for the FisherC dramatically.

Table 9: Estimated error of five classifiers for a pixel-representation, using a PCA to reduce the amount of features. The shown error is the average of 10 runs.

Classifier	NMC	SVC (linear)	SVC (polynomial kernel 2)	SVC (polynomial kernel 3)	FisherC
Error	0.2631	0.3247	0.3742	0.3661	0.2671
Standard deviation	0.0179	0.0243	0.0528	0.0477	0.0181

Preliminary Conclusion

The best classifier using the pixel representation seems to be the nearest mean classifier, after applying PCA to reduce the dimensionality. The resulting error is approximately 0.2631, with a standard deviation of 0.0179. The classifier is fairly stable over different training sets, but it does not perform well enough to pass the threshold.

5.4 Final Conclusion and Evaluation for Case 2

As can be seen in table 10, A Linear Discriminant Classifier with a PCA reducing the feature space to 29 features has the best performance out of all methods that were tried.

Table 10: Overview of the best performing classifiers per representation, and their errors.

Representation	Best Classifier	Estimated Error
Pixeldata	NMC with PCA	0.2631
Features	LDC with 10 out of 14 selected features	$\sim$0.30
Dissimilarities	LDC with a PCA reducing to 29 features	0.2184

The error of the chosen classifier was determined more accurately by training it ten times on 10 samples per class and testing it on the remaining objects. This way some measure of stability can be given, using the standard deviation between the repetitions. This achieved a mean error rate of 0.2184 with standard deviation 0.0153. The standard deviation is quite high, especially compared to Case 1. This is as expected, since the amount of data for Case 2 is limited. (picked at random from the original dataset).

The benchmark test (using `nist_eval.m`) yields an error rate of 0.194 for the chosen classifier for scenario 2. This error is below the desired threshold of 25%, and also a bit lower than the estimated error.

6 Conclusion and Discussion

6.1 Performance of Classifiers for both cases

The performance of the classifiers for both cases, and their shortcomings, will be summarized in this section.

Case 1

For Case 1, the recommended classifier is a Parzen density estimating classifier, combined using the product rule with a quadratic discriminant classifier which is applied to a dataset (representing the objects with pixels) whose dimensionality is reduced using principle component analysis preserving 90% of the original variance. All provided data was used to train this classifier (10x1000=10,000 objects). The error rate was determined to be 0.0219 with standard deviation 5.3125e-4 using cross validation. The benchmark test yields an error rate of 0.0260.
Although this result is more than sufficient, some improvements could be made. In the interest of time, support vector machines were not considered in depth in this report, due to the long time it takes to calculate the classifier. According to the literature, they offer a promising prospect for classifying in high dimensional spaces, which would be a major advantage for this case since the pixel representation was used. Furthermore, a brief look was given dissimilarity measures. Results were promising, but could not be pursued further in the interest of time. Testing the classifiers for different dissimilarity measures and using meticulously chosen prototypes might have improved the error rates, maybe beyond the results obtained using the pixel representation.

Case 2

For Case 2, the recommended classifier is a Linear Discriminant classifier trained on the dissimilarity-representation dataset (using all objects as prototypes) that has its features reduced to 29 by means of PCA. 10 objects per class were used to train this classifier (100 objects) and the remaining 990 per class were used to test on. The error rate was determined to be 0.2184 with a standard deviation resulting from variation of the training set of 0.0153. The benchmark test (using `nist_eval.m`) yields an error rate of 0.194
Although this result is sufficient, improvements could be made to make it more usable. One major improvement could be made by allowing the design to incorporate a set of prototypes, in stead of using the data in each batch. One could argue that this goes against the description for Case 2, but training will still involve each new batch. The major advantage of providing prototypes, is that they can be carefully selected to incorporate a wide variety of digits. This will yield a robust and versatile classifier. An alternative approach would be to incorporate a GUI, from which prototypes can be selected by hand for each batch.
Another point of interest in the design of Case 2, is the Support Vector classifier. For this classifier, only polynomial kernels were tried. Different kernels might have produced a different (better) result. Especially an advanced kernel like the ‘tangent distance kernel’, which incorporates rotation- and scaling insensitivity, could have improved the error rate.

6.2 Usefulness of the pre-processing

Finally, the image processing part of the presented algorithms needs to be evaluated. The classifiers for both cases were trained again, this time on a dataset without blob-removal and Gaussian smoothing (note that in case 1 there was also no blob-removing during the normal training, due to a large computation time.). The results are shown in table 11.

Table 11: Estimated errors of the used classifiers for both cases, with and without the image processing.

	Classifier	Error with pre-processing	Error without pre-processing
Case 1	Combination of ParzenC with QDC (with PCA applied)	0.0219	0.0309
Case 2	LDC (with PCA applied)	0.2184	0.2550

In both cases, implementation of the image processing makes the classifier perform better. In Case 1 this is only slightly better, and absence of the image processing still yields an error below the desired value, but in Case 2 the difference is larger. There, the absence of image processing results in a classifier performing worse than desired.
Some optimization could be done however. To start, the size of the images was chosen to be 16x16 pixels. This choice was motivated by the wish to reduce the run-times of the algorithms. The effect of using differently resized images for training could be investigated in the future. It seems logical that using larger, more detailed images, could improve classification up to a certain point. Similarly with the value for the Gauss filter: it was chosen to be 0.8 using visual inspection of the resulting images, but this value was not optimized considering the error rates of the classifiers.

7 Recommendations

When continuing designing the digit classification system for the purpose of classifying bank cheques, the following points are worth taking a closer look at.

- **Implement a reject option:** Especially when dealing with important interactions like bank cheques, any mistake made by the system may be a costly one. It is possible to lower the error rate further by implementing a reject option. This means that the system will abstain from making a definitive decision when the digit cannot be classified with a certainty above a certain threshold. After a digit is rejected several continuations are possible: the cheque containing a rejected digit can then be handled by human employees, or the submitter of the cheque can be contacted automatically, for example.

- **Discontinue case 2 classification:** In case 2, a new classifier is trained for every batch of cheques to be processed. It seems difficult to come up with a real world scenario in which this would be an efficient procedure. A large training set containing handwritten digits is available, which makes it possible to create a classifier with an error rate below 3%. This classifier can easily be used for all batches of cheques, without adaptation problems. To keep the classifier up to date with handwriting trends it can be retrained after a certain amount of time. Training a new classifier for every batch delivers an error rate close to 20%. Practically all submitted cheques would suffer misclassification in several digits. Discontinuation of this procedure should therefore be considered.

- **Investigate adding extra features:** The image analysis toolbox used in this report produces at most 14 usable features. This turned out to be insufficient to reach the thresholds set for the error rate. If the dimensionality of the feature space could be enlarged, this might benefit the performance of the classifiers (for case 1 in particular). Investigating ways to extract more features from the given images might turn out to be beneficial. Classifying using feature representation might be able to compete with the pixel data approach.

- **Investigate combining different representations:** The possibility to combine classifiers was considered in this report, but not the combination of different representations. For instance for Case 1, combining a classifier which is able to classify ‘8’ and ‘9’ using features or dissimilarities, with the classifiers described above using the pixel data approach, might further decrease the error rate.

- **More training data?:** For scenario 2, using more training data would obviously help, but this is not in the spirit of this scenario. For scenario 1 a learning curve was made for the recommended combination of classifiers, which can be seen in figure 14. The apparent error sinks to zero, because the Parzen classifier gives an error of zero on its own training set. The estimated true error seems to converge to an asymptotic value. This suggests that the attained error will not decrease significantly after adding more training objects.

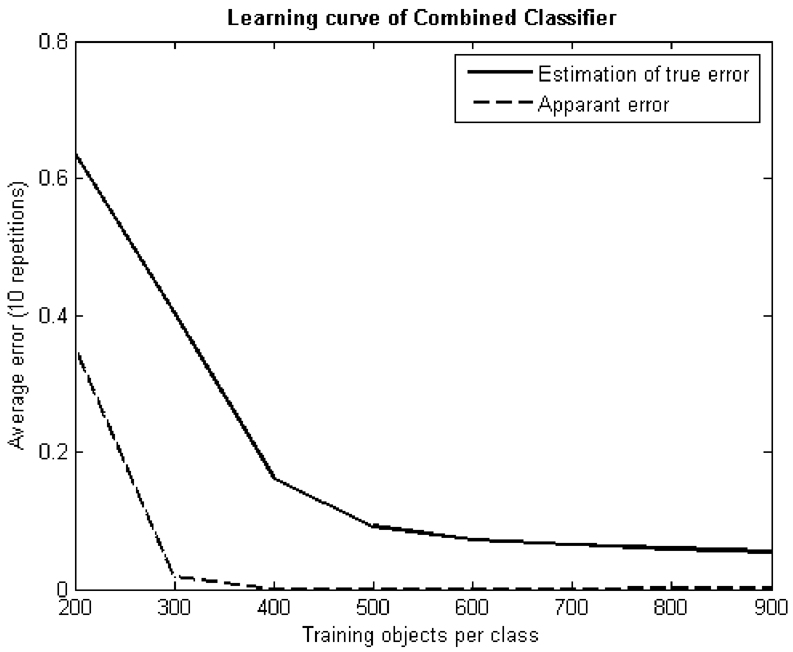

Figure 14: Learning curve for the combined classifier of Case 1.

- **Remarks about computation time**

  1. *Case 1:* For Case 1, once the classifier is trained on the entire dataset, time is not so much an issue. The training of the classifier itself takes a long time though. This is due to the large amount of data, in combination with the relatively cumbersome image processing algorithms in the pre-processing phase. If, for whatever reason, a new classifier has to be trained on a new datafile of numbers, this should be taken into account.
     If the classification itself needs to happen faster, for whatever reason, a (proper) implementation of an SVM might be preferred over ParzenC. While the training of an SVM might take long, its classification-performance after training is fast.
  2. *Case 2:* In the instance of Case 2, more could be done to optimize the training of the classifier. Since the classifier is trained for each batch of cheques (and there will presumably be a lot of batches) this will benefit the user. Again, the most time-consuming factor here is the pre-processing. One way to optimize this (and by the way, this will also work for Case 1!) is to incorporate parallel computing in the algorithm, using `parfor` in stead of `for` loops. This will spread the calculations over the different cores of the computer, resulting in a much faster runtime.

References

[1] Dengsheng Zhang and Guojun Lu, *Evaluation of Similarity Measurement for Image Retrieval*, Gippsland School of Computing and Info Tech, Monash University, Churchill, 2003.