TU Delft

IN4085: Pattern Recognition

Group 47

Final Assignment

Alexandru Bălan
4722574

Andra Ionescu
4722485

Frank Gallagher
4295900

Abstract

In this paper pattern recognition techniques are analysed and deployed on a set of handwritten digits provided by the US National Institute of Standards & Technology. A classification pipeline was designed which subjects the raw data to: 1. pre-processing, 2. data representation, 3. dimensionality reduction and 4. classification. Different strategies for all four steps were analysed and discussed. Two scenarios where considered where the classifier was trained with a large (200 objects per class) and small (10 objects per class) data set.

Initial experiments showed (classification of the un-processed data) best performance of *fisherc* on the dissimilarity data set (4.3% error for euclidean dissimilarity) for scenario 1. For scenario 2 *ldc* performed best on the feature data set with 22% error.

Pre-processing of the data included noise removal, slant correction and normalization. Best results where found for *fisherc* on the euclidean dissimilarity data set with 3.6% and *parzenc* on the pixel data set with 12.8% error (scenario 2).

A PCA dimensionality reduction showed improvement for most classifiers in both scenarios.

Finally combinations of different classifiers where tested. Improvements were found for scenario 1, where a combination of *parzenc(PCA)* trained on the pixel data set and *loglc* trained on the feature data set resulted in an error of 3.3%. The performance for scenario 2 was lower than the previous, as was expected. The combination of *parzenc(PCA)* and *ldc(PCA)* obtained the best performance of 10%.

The final pipeline was evaluated with the provided evaluation function. In both scenarios the design requirements were met, with a performance of roughly 2.5% and 19% error, respectively, across multiple iterations. Additionally, the algorithm was deployed on a self created data-set of 60 (6×10) digits. The algorithm obtained a 16.7% error.

Contents

1 Introduction

The recognition of handwriting is a widely encountered pattern recognition problem. For this assignment we take the role of a pattern recognition consultancy company. We are faced with the task to research the possibility of using pattern recognition techniques to classify individual digits in bank account numbers and the monetary amount. Two scenarios as described by the client:

1. *"the pattern recognition system is trained once, and then applied in the field."*
2. *"the pattern recognition system is trained for each batch of cheques to be processed."*

For scenario one there is a large set of training data available and may be put to use. In this case, at least 200 and at most 1000 objects per class. For scenario two there is a much smaller amount of training data available, at most 10 objects per class.

The data supplied by the fictional client was put together by the US National Institute of Standards & Technology, NIST and consists of 2800 images of handwritten digits. It is required for the first and second pattern recognition system to deliver an error rate below 5% and 25% respectively.

2 Strategical Outline

This paper shows step by step how the task was approached. Before the steps are discussed in detail a summary is given of the strategical outline.

1. **Pre-process** - The raw data may contain unwanted irregularities which can affect the performance of the classification. Before the data is classified, it will be filtered for noise, smoothed, corrected for slant and finally normalized.
2. **Representation Transformation** - How the data is presented to the classifiers will affect the complexity and performance of the task. Three different representation are discussed, namely *Pixel-, Profile-, Feature-Representation* and *Dissimilarity.*
3. **Dimensionality Reduction** - The last step before the classification is done a dimensionality reduction will be performed on the data-set. A *feature selection* and *feature extraction* will be analysed.
4. **Classifier Evaluation** - Finally the processed data is given to multiple classifiers. Their performance is analysed using a benchmark provided by the client.

2.0.1 Experimental set-up

All coding will be done in MATLAB (1) and extensive use is made of the PRTools toolbox (2). The computations were performed on a Apple Macbook Pro 15" with a quad-core 3.1[GHz] i7 processor, 16GB RAM.

2.1 Preprocessing

An essential step in any recognition system is pre-processing of the large data-set to make the results more suitable for classification (3). In the NIST data-set the automatically segmented digits may contain noise such as random pixels, deformations and parts of other digits. The digits also differ from size and style and most are not positioned in the center. Therefor the data will be *pre-procssed* by correctly positioning the digit inside a box, possible noise is remove, the digit is straightened and finally normalized. It should be noted that the process does not increase the inherent information content in the data, but it does increase the dynamic range of the features. Furthermore it will also reduce the complexity in the raw data which decreases the rate of recognition.

2.1.1 Noise removal

When the NIST data-set is imported, it is first placed inside a box using the PRtool function `im_box` sizing them to square images. To filter noise present in the images, three PRtool functions are used: `imclose`, `imerode` and `imdilate`. Here `imclose` closes holes, after which a layer of pixel is scraped of

and re-added by `imerode` and `imdilate` respectively. The latter helps to remove exaggerated contours if present. The result can be seen in fig. 1.

(a) Raw image (b) Noise filtered image

Figure 1: The result of the first pre-process step. On the left side the raw image of a zero digit. The raw image is transformed into image object, after which the image is placed inside a box. Noise is finally filtered using the PRTool functions `imclose`, `imerode` and `imdilate`. The result is shown on the right side.

2.1.2 Slant Correction

According to (4), the uniqueness of one's handwriting is determined by the slope or slant of their writing, among other things. When analysing the raw NIST data it can be seen most digits indeed have a certain angle, some more apparent then others. As we only wish to discriminate between the different digits and not two different images of the same digit, it would be help full for the classification if this would be corrected. In (5) a method using central moments is proposed to correct this angle. The central moments are given by:

$$\mu_{j,k} = \sum_x \sum_y (x - \bar{x})^j (y - \bar{y})^k f(x, y) \tag{1}$$

where $\bar{x} = m_{10}/m_{00}$ and $\bar{x} = m_{01}/m_{00}$ are the coordinates of the center of gravity of the digit in the image. The angle orientation of each digit is then given by:

$$\theta = \frac{1}{2} \arctan \frac{2\bar{\mu}_{11}}{\bar{\mu}_{20} - \bar{\mu}_{02}} \tag{2}$$

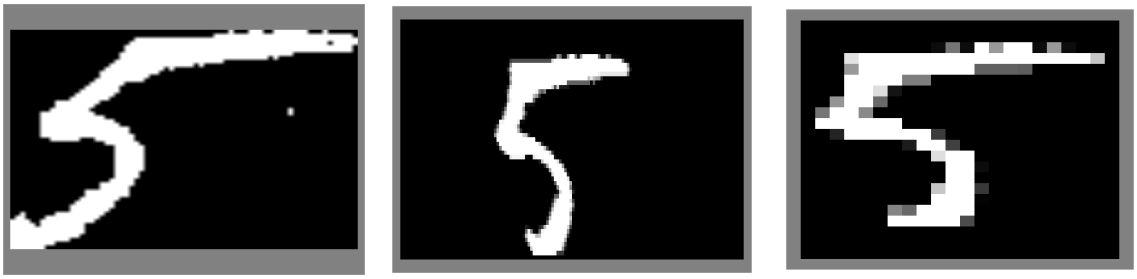
where $\bar{\mu}_{11} = \mu_{11}/\mu_{00}$, $\bar{\mu}_{02} = \mu_{02}/\mu_{00}$. The slant correction is finally applied by using an affine transformation:

$$S_\theta = \begin{bmatrix} 1 & 0 & 0 \\ \sin(\frac{1}{2}\pi - \theta) & \cos(\frac{1}{2}\pi - \theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

The moments are computed using the PRTool function `im_moments`, the transformation is applied with the PRTool function `imwarp`. The result of the slant correction can be seen in fig. 2.

2.1.3 Normalizing

To minimize the effect of different sized digits, the images are each normalized. The images are resized to the same size and finally a single pixel black box is added. The final result is shown in fig. 2.



(a) Raw image (b) Slant corrected image (c) Image after normalization of image (b)

Figure 2: The result of the slant correction and normalization. On the left side the raw image of a five digit. The raw image has first been boxed and filtered for noise. By calculating an offset angle using `im_moments` an transformation matrix is constructed and applied to the image. The middle image shows the result. After noise removal and slant correction the image is finally normalized by centering the image and adding a single pixel box. The most right image shows the final result.

2.2 Data representation

For the classification of data, some form of information has to be retrieved from it on which the classifier can perform calculations. How the data is stored and presented to the classifier is called data representation. As the performance of the classifier can rely on the way the data is represented, it is of great importance to do this thoughtfully. In this paper three methods are explored for the representation of the written digits, namely 1. *Pixel Representation*, 2. *Feature Representation* and 3. *Dissimilarities*. In the following section a short summary is given of these methods and their main characteristics.

2.2.1 Pixel Representation

A digital image consists of small boxes, or 'pixels', with a varying light intensity. Therefor a pixel representation can be seen as the most basic feature of a given image. Because this method uses every single pixel the computational burden can quickly rise, with possibly unwanted consequences like *the curse of dimensionality* (6). To prevent such problems, the images are resized to a smaller size. The data-set can be directly fed to the classifiers as it only uses a single feature.

2.2.2 Profile

The two dimensional image could also be represented by two one dimensional sets which hold information about the profile of the image. One method would be *pixel count*, which counts the number of pixels in a column or row and stores this inside a vector. This type of representation is called *profile representation*. The PRTool function `im_profile` is used to compute the horizontal and vertical images profiles.

2.2.3 Features

In feature representation the image data is transformed into a set of features and grouped inside a feature vector. This reduces the total amount of features as apposed to a full information representation like pixels and thus reducing the overall complexity of the classification. Features can be extracted from the data using the PRTool function `im_features`.

2.2.4 Dissimilarity

Lastly the dissimilarity representation will be used. This method searches for the differences between two objects. It can be preferred over features as every aspect of the object may be taken into account and so no important properties can be forgotten (7). The issue now arises that the dissimilarity measure, or proximity. The function `proxm` is used from the PRTool toolbox to compute the dissimilarity mapping which has multiple proximity options. Both a *euclidean* (option *distance* in the PRTool toolbox) and a *cosine* dissimilarity mapping was performed. The distance D_{euc} is given by:

$$D_{euc} = ||A - B||^P \tag{4}$$

Here A is the data-set used for representation, B the data-set applied to the representation set A and P the parameter of proximity. The distance D_{cos} is given by:

$$D_{cos} = 1 - \frac{AB'}{||A||||B||} \tag{5}$$

3 Dimensionality Reduction

Applying a dimensionality reduction has multiple advantages. The top reasons include faster training, reduce the complexity, improve the accuracy and reduce over fitting(8). The two methods are explored, namely *feature selection* and *feature extraction*, which, as the name implies, make a selection and extraction of certain features respectively. In *feature selection* the most relevant features are selected from the total data-set. In *feature extraction* the data-set is reduced in size by mapping it to a smaller amount.

3.1 Feature Selection

Feature selection was done using individual-, forward- and backward-selection. As some data-sets have a very high number of features, the feature selection was performed differently depending on the representation of the data.

Individual-, Forward- and Backward-Selection

In backward selection, at a given iteration the selected classifier is trained on n features. Then one feature is removed at a time and the same model is trained for $n-1$ features, n times. The feature which resulted in the smallest increase in the error rate is then removed, leaving $n-1$ features. This process is repeated. Each iteration k produces both a model trained on $n-k$ features and an error $e(k)$. The smallest number of features possible is then defined as the number necessary to reach the maximum allowable error $e(k)$ (9). Forward selection is the inverse process of the backward selection.

The feature selection was using the PRTool function `featseli`, `featself` and `featselb` for individual, forward and backward selection respectively. For the features represented data-set all three feature selection methods were used. The performance of each method was analysed and the best one was selected for further use. For the pixel represented and dissimilarity data-sets, only the individual selection up to 1300 features (out of 2500) was used. Forward and backward selection resulted in a very long execution time with the given hardware and was discarded.

3.2 Feature Extraction

A downside of feature selection would be the fact it reduces the dimensionality by dropping a certain number of features, thus losing information. Feature extraction is less affected by this. Principal Component Analysis (PCA) is a technique for feature extraction. PCA is a statistical procedure that transforms the original data set into a new, typically much smaller, data-set that still contains most of the information as the larger set(10). PCA does so by creating a covariance matrix from the original data and performs a singular value decomposition on it to obtain the so called *principal components* (PC's) which are the eigenvalues of the original data. The PC's are ordered from most- to least important. It can then be specified how much of the original data should be preserved. The smallest PC's are removed and then maps the original data to a new set with this new covariance matrix (11).

3.2.1 PCA approach

Implementation of PCA was done using the PRTool function `pcam`. In order to perform the PCA, we have computed the error at different percentages of retained variance (from 0.1 to 0.99 with 0.1 increments) and plotted the series for each targeted classifier. Once the desired percentage was roughly identified, a more granular search was performed around the value in order to detect the optimal solution.

4 Classification

Since the framework offers various classifiers to experiment with, using all of them through the entire evaluation process would not be practical, given the current time constraints. Therefore, before exploring

further on methods to improve the classifiers in order to reduce the error, we have used 10-fold cross validation (averaged on 2 runs) to test the error of each classifier, relative to the data-sets we have extracted from the original NIST data-set.

Each data-set contains 250 instances of each class, therefore totalling to 2500 entries. The number of features depends on the data-set: the smallest one is the one obtained through `im_features` (24 features), while the biggest is the one containing the dissimilarities (2500 features for a 2500 entries data-set). Due to the size differences, training the algorithms on the dissimilarity data-set took significantly longer. Where there is no value specified, it means that the execution time was too large (more than several hours) for the given hardware capacity (quad-core i7 processor, 16GB RAM). We have tested several dissimilarity algorithms, such as Hellinger, City-block and Minkowsky. However, the results were comparable, therefore we have only presented the most commonly used algorithms.

4.1 Classifiers

Initially, we have used 13 classifiers on all the unprocessed data-sets (features, profile, pixel, dissimilarity). The chosen classifiers are the following:

Linear Bayes Normal Classifier

This classifier belongs to linear classifiers category and does the computation assuming normal densities with equal covariance matrices. The MATLAB function to run such a classifier is `ldc`, notation also used in all the following tables.

Quadratic Bayes Normal Classifier

This is a quadratic classifier that also assumes normal densities. It has a more general approach to the linear classifier, because the separation of the class is made by a quadratic surface. The MATLAB function used in the project is `qdc`.

Naive Bayes Classifier

The Naive Bayes Classifier belongs to the simple probabilistic category and it applies the Bayes theorem with the naive assumption of independence between the features. The MATLAB function used is `naivebc`.

Logistic Linear Classifier

The logistic linear classifier, unlike the linear classifier, transforms the output using the logistic (sigmoid) function to return a probability value which can then be mapped to two or more discrete classes. The documentation specify that it becomes very slow for feature sizes bigger than 1000, which was validated by try to compute the classification using the dissimilarity features (size = 2500). The MATLAB function used is `loglc`.

Fisher's Least Square Linear Discriminant

Another classifier that belongs to the linear classifier category. The special characteristic of this classifier is that it uses the one-against-all strategy resulting in a set of linear based classifiers. The MATLAB routine used is `fisherc`.

Nearest Mean Classifier

The classifier uses the plain nearest mean computation between the classes and it is sensitive to scaling, but not to class priors. It is also part of the linear classifiers family. The MATLAB function is called `nmc`.

K-Nearest Neighbor Classifier

The classifier belongs to the category of non-parametric classifiers. The number of points k that falls inside the volume is specified and the size of the volume is adjusted every time to include the k points. The MATLAB routine is `knnc`. Besides that, we also used `knndc` for the dissimilarity matrices, which showed a significant improvement.

Optimisation of the Parzen classifier

This is another implementation of the non-parametric classifiers. Unlike K-Nearest Neighbor Classifier, Parzen classifier has a fixed volume size and the k number of points is left to vary. The MATLAB function used is `parzenc` and for the dissimilarity matrices `parzenddc`.

Back-propagation trained feed-forward neural net classifier

This is a feed-forward neural network that has a minimum of three layers of neurons, one input, at least one hidden and one output layer. The back-propagation represents the training method used for this type of neural network, which means that the neurons adapt the weights in order to learn new information. The MATLAB routine is `bpxnc`.

Trainable linear perceptron classifier

The *perlc* algorithm from PRTools uses a perceptron trained on the provided dataset. The perceptron belongs to the linear classifier type and has a relative simple approach: the classifier takes the input, aggregates it and returns a binary response according to a given threshold.

Verzakov Tree - Trainable Decision Tree Classifier

A decision tree divides repetitively the working area into sub parts, until a specific criteria is met or until the classes are pur (contin only members of one class). The Verzakov Tree discards the branches that do not improve the classification error. This results in a smaller tree and more stable algorithm. The MATLAB function is called `dtc`.

5 Results

5.1 Scenario 1

5.1.1 Initial analysis

The results we have obtained were surprising, as more complex classifiers proved to be less accurate than the simpler ones, such as 1-NN. However, there is no classifier that performs best on all the data-sets (fisher was extremely powerful on the dissimilarity data-set, while K-NN variations and parzen were perfoming best on the pixel data-set). Therefore, we have decided to pick several that performed the best overall: *1-NN, 3-NN* for good performance on the Pixel and Dissimilarity data-sets, *fisherc* for best performance on the Dissimilarity data-sets, *parzenc* for best performance on the Pixel data-set and Profile data-set and *loglc* for best performance on the Features data-set. We have also added *bpxnc*, as it exhibited promising results accross the board, especially when increasing the data-set size (with the exception of the Dissimilarity data-set). Moreover, neural networks can generally be improved through parameter tuning and bigger data-sets. The results of the initial analysis can be seen in table 1.

Classifier \ Representation	Features (im_features)	Profile (im_profile)	Pixel	Dissimilarity Euclidean	Dissimilarity Cosine
loglc	**0.1362**	0.2164	0.2516	-	-
fisherc	0.1900	0.2820	0.1982	**0.0432**	**0.1842**
ldc	0.1574	0.8984	0.8974	0.9	0.9
naivebc	0.285	0.1716	0.3572	0.9	0.9
nmc	0.6046	0.2994	0.2708	0.9	0.9
qdc	0.1894	0.8500	0.9000	0.9	0.9
1-NN	0.4198	0.1652	**0.0900**	**0.088**	**0.0684**
2-NN	0.4386	0.1768	0.1228	0.1236	0.0832
3-NN	0.4132	0.1598	**0.1066**	**0.1048**	**0.0804**
parzenc	0.4232	**0.1542**	**0.0880**	0.0896	0.0644
bpxnc	0.2258	0.1736	0.1172	0.5916	0.4852
perlc	0.1816	0.3808	0.1748	0.4896	0.4744
dtc	0.2394	0.2388	0.2734	0.2254	0.2360

Table 1: Scenario 1 - initial analysis results

5.1.2 Image pre-processing initial analysis

As detailed in section 5.1.1, we have chosen 6 classifiers to evaluate on the pre-processed data-sets. However, due to the operations that were necessary to pre-process, im_profile features were no longer available. Moreover, the performance degraded significantly for all the classifiers that were tested on the im_features. Nonetheless, for the other three data-sets, significant improvements can be observed across the board, as detailed in table 2. This shows that, for some classifiers, the required client performance was nearly met, just by performing a pre-processing step.

Classifier \ Representation	Features (im_features)	Pixel	Dissimilarity Euclidean	Dissimilarity Cosine
loglc	0.6946	0.2348	-	-
fisherc	0.7070	0.1430	**0.0356**	0.1426
1-NN	0.6514	0.0536	0.0548	0.0494
3-NN	0.6260	0.0636	0.0632	0.0518
parzenc	0.7058	0.0548	0.0520	**0.0444**
bpxnc	0.7026	0.0780	0.5162	0.3294

Table 2: Scenario 1 - Image pre-processing analysis results

5.1.3 Feature selection

The results for all three data representations can be seen in fig. 3. For the im_profile represented data-set, reducing the feature size does not show improvements in the error of any of the classifiers. The same conclusion can be drawn by looking at the error curve for the im_features data-set.

For the pixel data-set, two classifiers can be improved through feature selection, namely Fisher and Logistic Classifiers, which show that roughly 100 features (out of 484) are enough to get the best performance, as the figure shows signs of over-training after that number. This is a rather interesting finding as, intuitively, one might assume that all the pixels of an image are necessary to perform an accurate classification.

Dissimilarity data-sets were not evaluated in this step, as feature selection cannot be done on dissimilarity data. The detailed data regarding classifiers that have been improved through feature selection can be observed in table 3a.

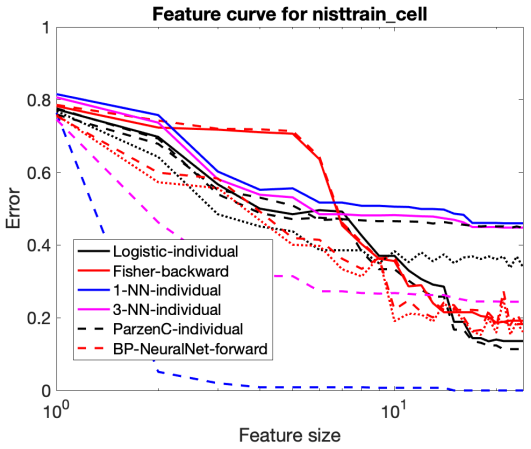
Classifier	Measurements	Pixel
loglc	Err	0.1840
	#feat	41
fisherc	Err	0.1536
	#feat	181

(a) Feature selection detailed results

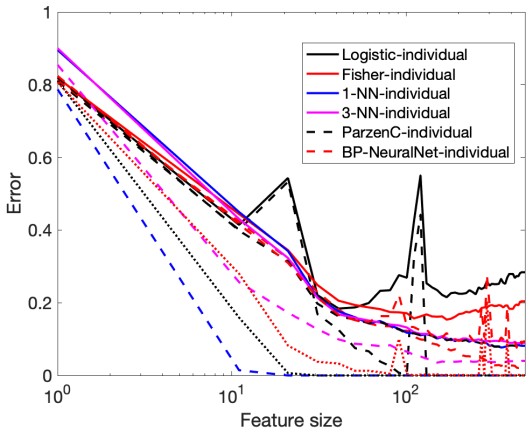
Classifier	Measurements	Pixel
loglc	Err	0.1036
	Std	0.0030
	#PC	37
fisherc	Err	0.1609
	Std	0.0043
	#PC	137
1-NN	Err	0.0574
	Std	0.0018
	#PC	30
3-NN	Err	0.0620
	Std	2.8284e-04
	#PC	27
parzenc	Err	**0.0416**
	Std	5.6569e-04
	#PC	29

(b) Feature extraction detailed results

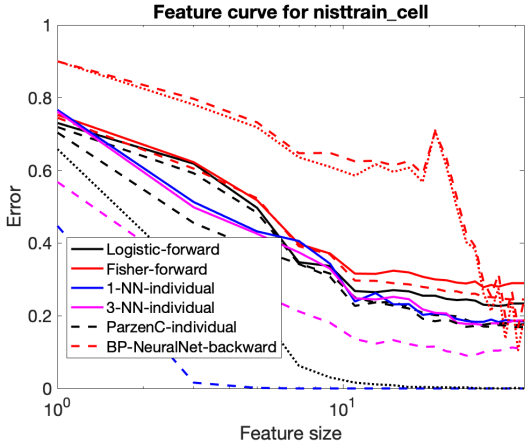
Table 3: Dimension reduction



(a) im_features



(b) pixel



(c) Im_profile

Figure 3: Error curves for the data-sets

5.1.4 Principal Component Analysis

The Principal Component Analysis (PCA) was performed as described in section 3.2.1 and resulted in the plots in fig. 4. As initial PCA on *bpxnc* yielded very high execution times even for smaller data-sets, this classifier was skipped. For the pixel data-set, significant improvements can be obtained for all classifiers by retaining a percentage of the variance. The results can be observed in table 3b. For all the other data-sets, no improvements can be obtained for any of the tested classifiers.

5.1.5 Combining classifiers

Although the performance of the system met the criteria imposed by the client, we wanted to verify if we can further improve it. Therefore, we tried combining the best classifiers on various data-sets. We performed a parallel and a stack combination, using *mean, min, max, prod, median* as the combining rule. The best results among all the combinations can be seen in table 4. Surprisingly, we have two classifier combinations that resulted in the same error value: the first one is *parzenc* with PCA on the Pixel data-set combined with *loglc* on the Features data-set using *prod* rule and the second one which is the same combination as the first one with the addition of *parzenc* on Profile data-set. Since the last addition did not improve the performance, we have decided to use the previous combination as it requires less complexity.

Classifiers	data-sets	Best Combiner	Error / Combination Type	
			sequential	parallel
parzenc (PCA) 1-NN (PCA) 3-NN (PCA)	pixel	PROD	0.0420	0.0424
parzenc (PCA) 1-NN (PCA)	pixel	PROD	0.0410	0.0416
parzenc (PCA) loglc	pixel features	MIN	-	0.0332
parzenc (PCA) loglc parzen	pixel features profile	PROD	-	0.0332

Table 4: Results of combining classifiers

5.1.6 Client evaluation

The client provided us with the evaluation function that he will use to verify if the system behaves as we specify. Therefore, we performed the evaluation on the selected classifier combination by running *nist_eval* for many variations of n and multiple iterations and computing the minimum, average and maximum value of the output. The results are presented in table 5.

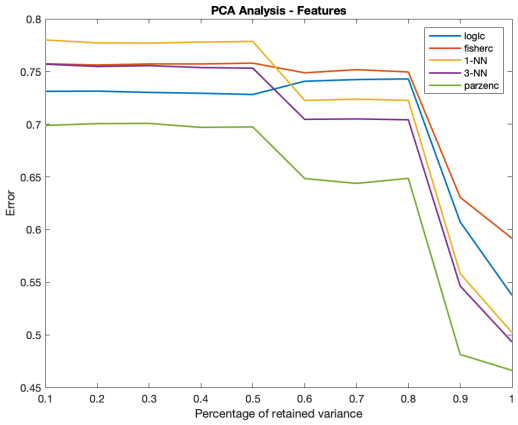
Classifier	nist_eval evaluation			n	iterations
	min	mean	max		
parzenc (PCA) - pixel loglc - features	0.023	0.023	0.023	100	1
	0.016	0.0224	0.028	50	5
	0	0.0245	0.05	10	20

Table 5: Nist˙eval results for scenario 1

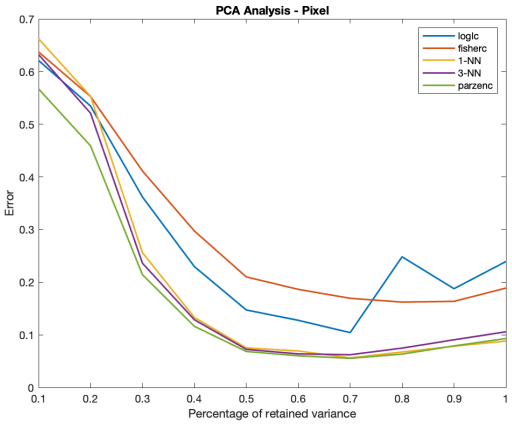
5.2 Scenario 2

5.2.1 Initial analysis

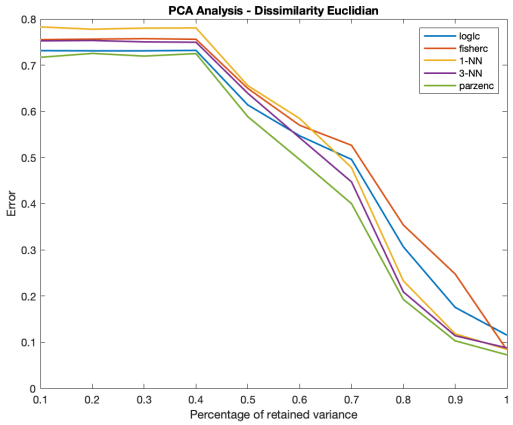
The same initial analysis was applied for Scenario 2 as was the case for Scenario 1, having the same classifiers and data-sets. In this case, the dissimilarity data-sets are less useful, due to the significantly



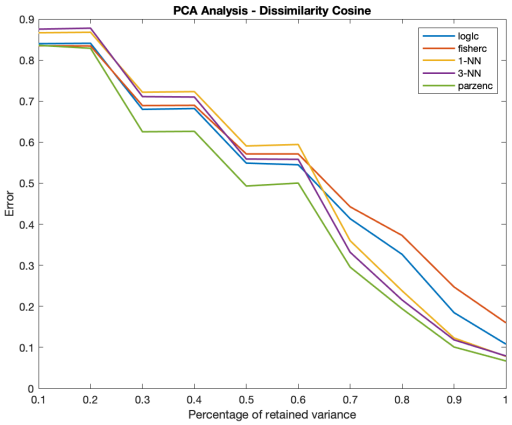
(a) im_features



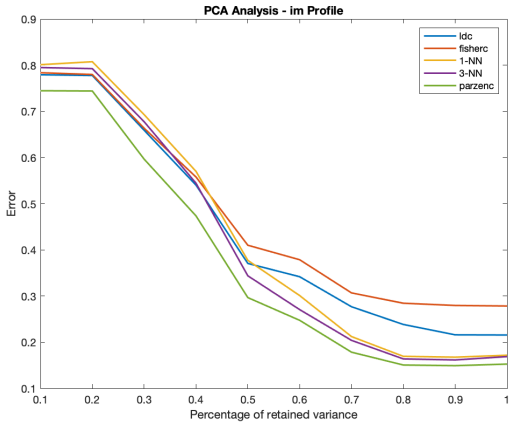
(b) pixel



(c) Euclidean Dissimilarity



(d) Cosine Dissimilarity



(e) im_profile

Figure 4: PCA Analysis

smaller size (for 250 objects per class, the dimension was 2500 x 2500, while for 10 objects per class the dimension is 100 x 100). As was the case for the first scenario, the classifiers that are usually considered less complex show better performance, especially given that now the size is an order of magnitude smaller. Some algorithms, such as *parzenc* and *k-NN* are again in the list of top performers, as can be seen in table 6. However, other classifiers have a good performance on Features data-set (*ldc*) and Pixel data-set (*naivebc*), contrary to Scenario 1.

Furthermore, given the fact that the data-set was smaller, the processing time decreased significantly and we managed to test the classifiers on different image sizes. Thus, we discovered that a smaller image is more suitable for scenario 2. Therefore, we have used an 18x18 image instead of 20x20, having on average a gain of 3% in error rate.

Classifier \ Representation	Features (im_features)	Profile (im_profile)	Pixel	Dissimilarity Euclidean	Dissimilarity Cosine
loglc	0.452	0.565	0.328	-	-
fisherc	**0.218**	0.42	0.326	0.366	0.414
ldc	**0.2**	0.365	0.9	0.276	0.272
naivebc	0.36	**0.25**	0.29	0.458	0.482
nmc	0.616	0.32	0.306	0.35	0.404
qdc	0.504	0.475	0.9	0.9	0.9
1-NN	0.602	0.285	0.258	0.286	**0.25**
2-NN	0.606	0.385	0.342	0.374	0.34
3-NN	0.612	0.385	0.312	0.346	0.366
parzenc	0.61	0.34	**0.25**	**0.27**	0.238
bpxnc	0.3	0.26	0.368	0.298	0.26
perlc	0.36	0.47	0.426	0.41	0.378
dtc	0.38	0.34	0.502	0.51	0.566

Table 6: Scenario 2 - Initial analysis

5.2.2 Image pre-processing initial analysis

As opposed to the first scenario, where the better performing classifiers were easier to distinguish from the others, that is no longer applicable here. For that reason, we have applied the pre-processing analysis on all the classifiers, instead of just for a few. As in the case of Scenario 1, the im_features data-set shows better performance when pre-processing is not applied. However, for the other data-sets, improvements can be seen across the board, as displayed in table 7.

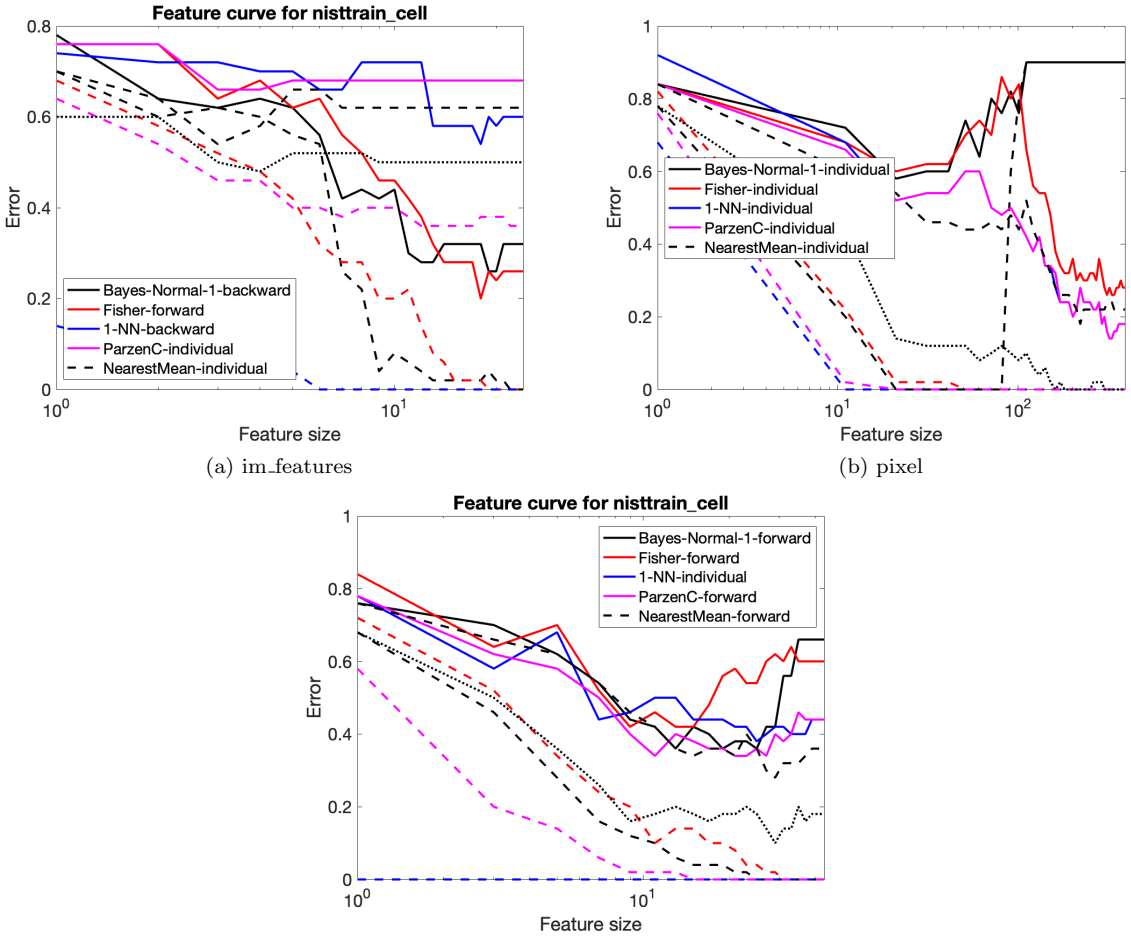
Classifier \ Representation	Features (im_features)	Pixel	Dissimilarity Euclidean	Dissimilarity Cosine
loglc	0.746	0.262	-	-
fisherc	0.764	0.244	0.254	0.326
ldc	0.734	0.9	0.226	**0.2**
naivebc	0.868	0.254	0.374	0.358
nmc	0.686	0.242	0.278	0.324
qdc	0.826	0.9	0.9	0.9
1-NN	0.692	**0.13**	**0.152**	**0.184**
2-NN	0.664	0.218	0.24	0.252
3-NN	0.682	0.222	0.254	0.256
parzenc	0.724	**0.128**	**0.13**	**0.2**
bpxnc	0.792	0.306	0.224	0.23
perlc	0.74	0.346	0.332	0.26
dtc	0.778	0.444	0.426	0.412

Table 7: Results of the pre-process analysis

5.2.3 Feature selection

Applying feature selection on a smaller data-set than Scenario 1 reveals a totally different behavior. By analyzing fig. 5, for the im_features only *ldc* and *fisherc* react to feature selection. However, the results are worse than initial analysis yielded. For the Pixel data-set, it can be seen that only *ldc* can be improved through feature selection. The results for im_profile data-set are fully different than those from Scenario 1. Here, *ldc* and *fisherc* resulted in potentially high improvements for feature selection, while the rest registered small deviations or none.

However, when redoing the experiment several times, we have observed large deviations in the overall results. This phenomenon is expected to happen as the training set is significantly smaller and more sensible to deviations. The results worth mentioning are summarised in table 8. Moreover, for all the classifiers that have been improved through feature selection, a better improvement was obtained through PCA as it can be seen in the next section.



(a) im_features

(b) pixel

(c) im_profile

Figure 5: Error curves for the data-sets

Classifier \ Representation	Measurements	im_features	im_profile	Pixel
ldc	Err	0.28	0.30	0.64
	#feat	19	15	51
fisherc	Err	0.26	0.34	N/A
	#feat	18	9	

Table 8: Feature selection detailed results

5.2.4 Principal Component Analysis

By analyzing table 6 and table 7, we have chosen the best 5 classifiers to perform the Principal Component Analysis (PCA): *ldc* for having the best performance on im_features, *fisherc* for good performance on im_features, *1-NN* for the second best performance on pixel and good performance on the dissimilarity data-sets, *parzenc* for best performance on the pixel data-set and *nmc* for good performance across the board.

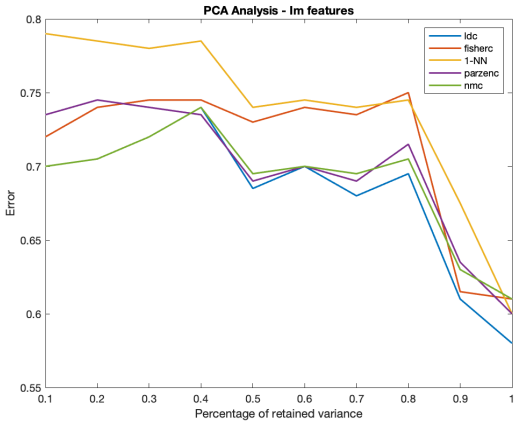
The results are displayed in fig. 6. For im_features, it can be observed that all the classifiers need all the features to obtain the best performance. For the pixel data-set, all the classifiers can be optimized by extracting the corresponding PC. For the Euclidean Dissimilarity data-set, *ldc* and *fisherc* can be vastly improved if only 0.9 of the variance is retained, while the Cosine Dissimilarity needs all the features to perform best, with the exception of *nmc*. For im_profile, small improvements can be obtained by extracting features for *ldc*, *fisherc* and *parzenc*. The final results of PCA can be observed in table 9.

Classifier \ Representation	Measurements	Im_profile	Pixel	Dissimilarity Euclidean	Dissimilarity Cosine
ldc	Err	0.228	0.148	0.1160	
	Std	0.0164	0.025	0.0219	N/A
	#PC	10	31	31	
fisherc	Err	0.324	0.1820	0.18	
	Std	0.0089	0.013	0.0122	N/A
	#PC	10	18	28	
1-NN	Err		0.116		
	Std	N/A	0.0055	N/A	N/A
	#PC		25		
parzenc	Err	0.3	0.11		
	Std	0.071	0.0	N/A	N/A
	#PC	10	25		
nmc	Err		0.238	0.25	0.268
	Std	N/A	0.0110	0.0089	0.0084
	#PC		28	13	8

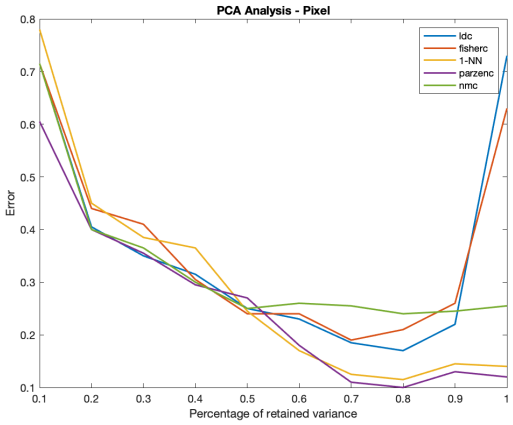
Table 9: PCA - Results for classifiers that can be improved

5.2.5 Combining classifiers

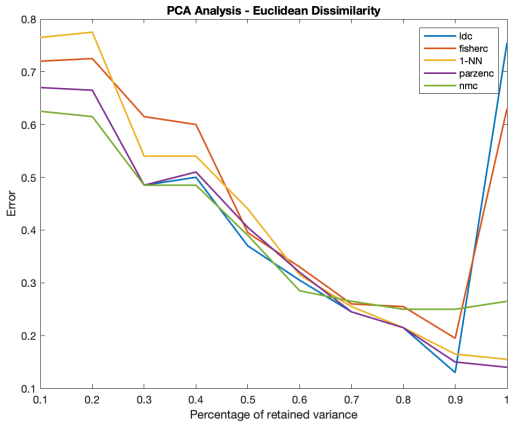
In an effort to increase the overall performance of the system from 0.1 which was obtained for *parzenc using PCA*, we have attempted to combine the best performing classifiers for various data-sets. As the performance for the dissimilarity data-sets was poor accross classifiers, we have directed our attention on the other data-sets. As for Scenario 1, we performed a parallel and stack classifier combination, using *meanc, minc, maxc, prodc and medianc* as combining rules. The results were close to the initial error and we obtained improvements just for one combination, as it can be seen in table 10. Therefore, we have decided to use the combination for the final system.



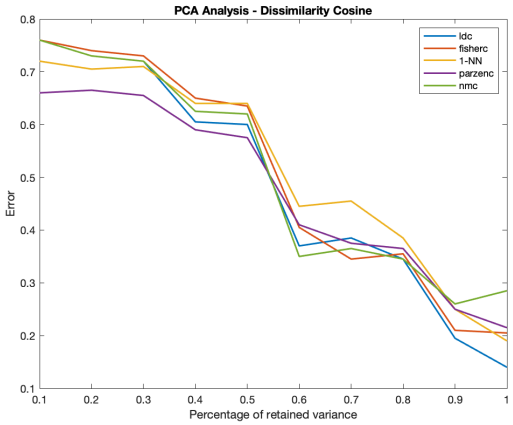
(a) im_features



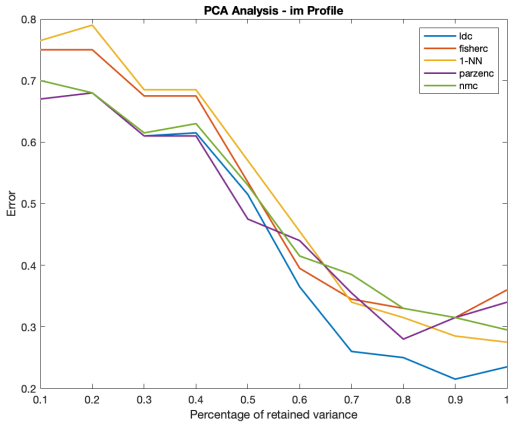
(b) pixel



(c) Euclidean Dissimilarity



(d) Cosine Dissimilarity



(e) im_profile

Figure 6: PCA at different percentages of retained variance

Classifiers	data-sets	Best Combiner	Error / Combination Type	
			sequential	parallel
parzenc (PCA) ldc	pixel im_features	MIN	-	0.12
parzenc (PCA) ldc (featseli)	pixel im_features	MIN	-	0.12
parzenc (PCA) 1-NN (PCA)	pixel	PROD	0.11	0.11
parzenc (PCA) ldc (PCA)	pixel	MEAN	0.1	0.105

Table 10: Results of combining classifiers

5.2.6 Client evaluation

The client provided us with the evaluation function that he will use to verify if the system behaves as we specify. The evaluation is slightly different for this scenario: for each of the 20 iterations, we have trained the classifier with a newly generated data-set, and used *nist_eval* with a test size of 10 instances per class. The results can be observed in table 11. By analyzing the results, we have noticed a significant difference between our 10-fold cross-validation and the client evaluation. Moreover, the variation between iterations was much greater than in the case of Scenario 1. However, the results are not surprising, given the fact that there is significantly more variation in such a small dataset than in the previous one.

Classifier	nist_eval evaluation			n	iterations
	min	mean	max		
parzenc (PCA) - pixel	0.12	0.1855	0.24	10	20

Table 11: Nist_eval results for scenario 2

6 Live Test

In order to perform the live test, we have scanned an image of 60 digits (6 instances for each digit), as can be seen in fig. 7 (a).

6.1 Processing the scan

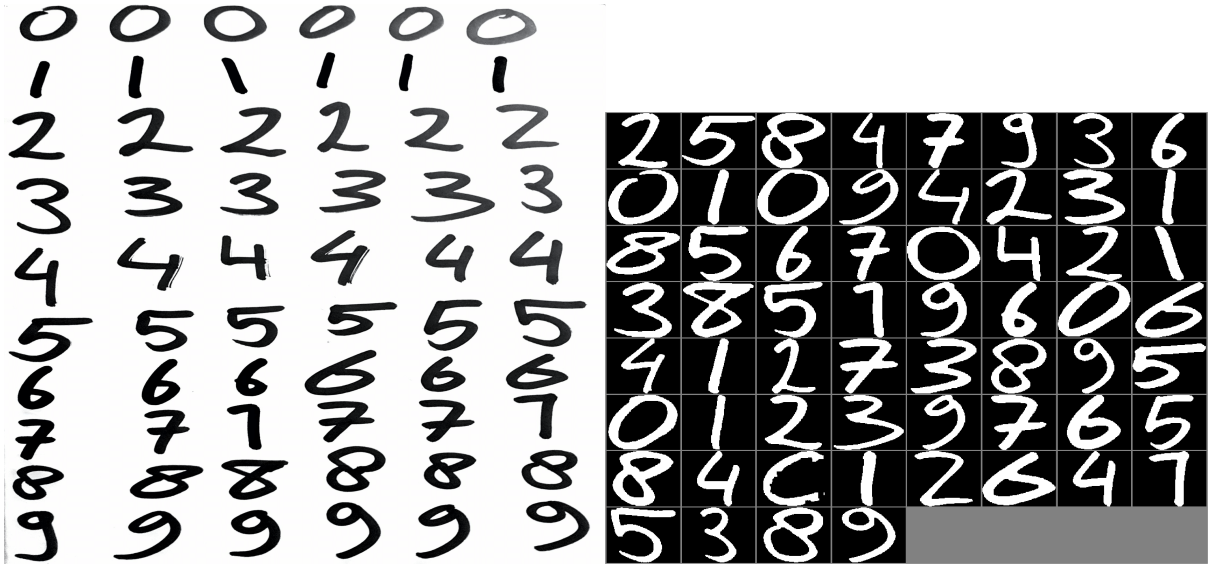
In order to add each individual digit into a data-set, the scan has to go through a digit separation process first. For this, the scan is converted to a gray scale image using MATLAB `rgb2gray`. The scan is filtered for noise in the same way as in the pre-processing process. To determine the correct bounding box which encloses a single digit MATLAB `regionprops` was used. This function searches for distinct objects in the image and creates a bounding box at the corresponding location. Using MATLAB `imcrop`, the digits can be cropped out of the scan. Finally the labels are added manually to every image.

6.2 Classification

Following this process, a *prdataset* is generated, as in fig. 7 b). However, at this step, the images are still very large. Therefore, they have to go through the same pre-processing as the NIST data-set. Given that the best classifier in Scenario 1 is a combination of classifiers on 2 data-sets (pixel and im_features), the same types have to be obtained from the live data-set. Since im_features has shown better performance on the standard, non-slanting pre-processing, two separate pipelines to obtain the data-sets have been devised. Once the live data-set has been obtained, the classifier that has been trained beforehand on the original NIST data-set (with 250 instances per class) can be tested. The result can be seen in table 12.

As can be observed, the classification error is significantly higher than when testing against the NIST data-set. There are several factors that have contributed to this deterioration. Firstly, although

detection of the borders of each digit works correctly most of the time, in some cases, it truncates the actual shape, as can be seen on the 7th row, 3rd image in fig. 7 b), where the 0 digit is not entirely maintained. Moreover, we have intentionally created some "difficult" digits, which can easily be confused with other. Such a case is two of the 7 digits, which can be confused with 1, or some of the 9 digits which are slightly rotated.



(a) Original image (b) Obtained data-set

Figure 7: Live test sample images

Classifier	Live test classification error
parzenc (PCA) - pixel loglc - features	0.1667

Table 12: Live test results

7 Conclusion

In this project we have researched the possibility of creating a system that recognises handwritten digits for a client that wants to process bank cheques. We explored the system capabilities on two scenarios provided by the client: one where the system is trained once and has a considerable amount of data (at least 250 instances per class) and another one where the system is trained for each batch of cheques and the data-set contains 10 objects per class.

For each scenario, we performed an initial analysis on 13 classifiers in order to assess the primary system performance. Then, we have enriched the image pre-processing algorithm, we have tried several dimension reduction techniques and finally we have combined the best classifiers in order to minimize the error.

The maximum error requested by the client for Scenario 1 was 5% and we managed to yield 3,3% using a parallel combination between parzen with PCA on the Pixel data-set and loglc on the im_feature data-set. Using the evaluation tool provided by the client, we have obtained an even smaller error (**2.8%**). This difference is due to a smaller amount of data used by the client tool, compared to the one used in our own evaluation.

For scenario 2, the maximum error requested by the client was 25%. Again, we managed to create a system that outputs a smaller error (**10%**) using parzenc(PCA), combined with ldc (PCA) on the Pixel data-set. Using the same client evaluation tool, we have yielded a bigger error, averaging around 19%.

In conclusion, we have reached the target specified by the client using one common classifier (parzen), with slightly different approaches and datasets for each scenario. Moreover, the system can be enriched by researching different processing and classification methodologies as it is presented in the next section.

8 Recommendations

In our experience with building the digit classification system, we have discovered several insights that might prove useful for the client with respect to the further development of the system. Firstly, we have found that, for the current system, the dimension of the data-set does not influence the performance after a certain threshold. We have found that a data-set containing 250 instances per class does perform significantly better than one containing 10 instances. However, when comparing 250 instances with 800 instances per class, no significant performance gain was observed. However, most of the classifiers we have studied in-depth, except for *bpxnc*, are characterized by a simpler structure and do not benefit from a very large data-set. Moreover, most of the complex classifiers are sensible to parameter tuning, which we did not research extensively. Therefore, under certain conditions, some classifiers might benefit from a larger data-set (especially Neural Networks, which require large data-sets to perform best).

Another issue that requires future work is combining the classifiers. In this research, we only focused on the basic parallel and serial combining, through 4 methods. However, another way of combining classifiers is through bagging or boosting, which are proven to improve the classification error over simple classifiers (12).

Through our research, we have shown that classifiers that are being trained on dissimilarity features can achieve high accuracy when the data-set is large enough. However, apart from time constraints (in the case of scenario 1), there was one characteristic that prevented us from properly testing a dissimilarity system against the client evaluation: since a dissimilarity data-set should always be square, training a classifier on a data-set that had more class instances than the client was not possible. Therefore, we were limited at 100 classes per instance, which had a severe impact on our data-set in scenario 1. Although we have performed several experiments using *distools*(13), further research in the matter is necessary.

Judging by the result from the live test, which is much higher than the results obtained with NIST data-set, we would recommend implementing a system based on active learning methodology. This way, the system can constantly adjust its accuracy, by learning new handwritten patterns. However, considering the high amount of time necessary for processing the images and retraining the classifier, the pipeline of the system must be carefully designed in order to overcome this challenge.

Another interesting finding from the live test is the quality of data. In our live data-set, we have one image that has missing parts from a digit. We realised that this can be a common processing error or even a writing mistake for a real life bank cheque. Therefore, such data should be discarded and the client should be notified for the presence of these outliers that might output erroneous results.

References

[1] MATLAB, *version 9.5.0 (R2018b)*. Natick, Massachusetts: The MathWorks Inc., 2018.

[2] DelftPR, *version 5.3.4*, 2018.

[3] S.-T. Bow, *Pattern recognition and image preprocessing*. Marcel Dekker New York, 2002.

[4] R. A. Huber and A. M. Headrick, *Handwriting identification: facts and fundamentals*. CRC press, 1999.

[5] M. Kozielski, J. Forster, and H. Ney, "Moment-based image normalization for handwritten text recognition," in *Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on*. IEEE, 2012, pp. 256–261.

[6] A. Hinneburg and D. A. Keim, "Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering," 1999.

[7] "Dissimilarities," 2018, accessed: 2019-01-19. [Online]. Available: \url{http://37steps.com/1153/dissimilarities/}

[8] L. Van Der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: a comparative," *J Mach Learn Res*, vol. 10, pp. 66–71, 2009.

[9] K. Z. Mao, "Orthogonal forward selection and backward elimination algorithms for feature subset selection," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 34, no. 1, pp. 629–634, 2004.

[10] I. Jolliffe, "Principal component analysis," in *International encyclopedia of statistical science*. Springer, 2011, pp. 1094–1096.

[11] J. Lever, M. Krzywinski, and N. Altman, "Points of significance: Principal component analysis," 2017.

[12] R. Maclin and D. Opitz, "An empirical evaluation of bagging and boosting," *AAAI/IAAI*, vol. 1997, pp. 546–551, 1997.

[13] "Distools," 2018, accessed: 2019-01-27. [Online]. Available: \url{http://37steps.com/distools/}