IN4085 Pattern Recognition
Final Assignment

Helmiriawan (4623835)
Priadi Teguh Wibowo (4625862)

1 Introduction

Bank cheques are developed as a way to do financial transactions without the need to carry a large amount of money. It is still widely used all over the world, thus there are lots of cheques which have to be processed every day. In order to reduce the time and effort, a system that can automatically process a large number of bank cheques can be beneficial.

In this report, a study about using pattern recognition techniques to make an automatic bank cheques processing system will be discussed. A dataset of handwritten digits from the US National Institute of Standards & Technologies (NIST) was used and two different scenarios were explored as the study cases. The results show that the proposed system can be used to recognise the handwritten digits with an acceptable error rate.

2 Methodology

The objective of this study is to recognise handwritten digits on bank cheques by using pattern recognition techniques in two different scenarios. In the first scenario, the pattern recognition system was trained by using a large number of training set (at least 200 and at most 1000 samples per class). While for the second scenario, the size of the training set was much smaller (at most 10 samples per class). The test error in the first scenario should be less than 5%, while in the second scenario it should be less than 25%.

Similar dataset of handwritten digits from NIST was used in both scenarios. The proposed system in this study is developed by using a pattern recognition tool in Matlab named PRTools [2]. Furthermore, a pipeline from [4] for designing a classification system was followed, as discussed in the following subsections.

2.1 Dataset

As mentioned previously, NIST handwritten digits dataset was used in this study as the dataset. It consists of 10 class numbers (between 0 to 9) with 1000 samples per class. Thus, there are 10 thousand samples in the dataset. In the original form, each sample has a different size for space saving. Therefore, data preprocessing was performed at the initial step to make all samples have a similar size. First, empty pixels were added to make all samples have a similar dimension. After that, the samples were also resized to a specific dimension to speed up the processing time at the next step. A subset of the preprocessed dataset is depicted in Figure 1.

There are three types of representation or feature set used in this study: pixels, image properties, and dissimilarity. Creating representation from image properties can be done by using *im_features* function in PRTools. The properties are the shape measurements and pixel value measurements, as can be seen in the following list.

- Area
- Bounding box
- Centroid
- Convex area
- Eccentricity
- Equivalent diameter
- Euler number

- Extent
- Filled area
- Major axis length
- Minor axis length
- Orientation
- Perimeter
- Solidity
- Max Intensity
- Mean Intensity
- Min Intensity
- Weighted centroid

In this study, three different pixel sizes were investigated: 10x10, 20x20 and 30x30. Each of them provides a different number of generated features per sample, which are 100, 400 and 900 respectively. The first feature is the first pixel, the second feature is the second pixel, and so on.

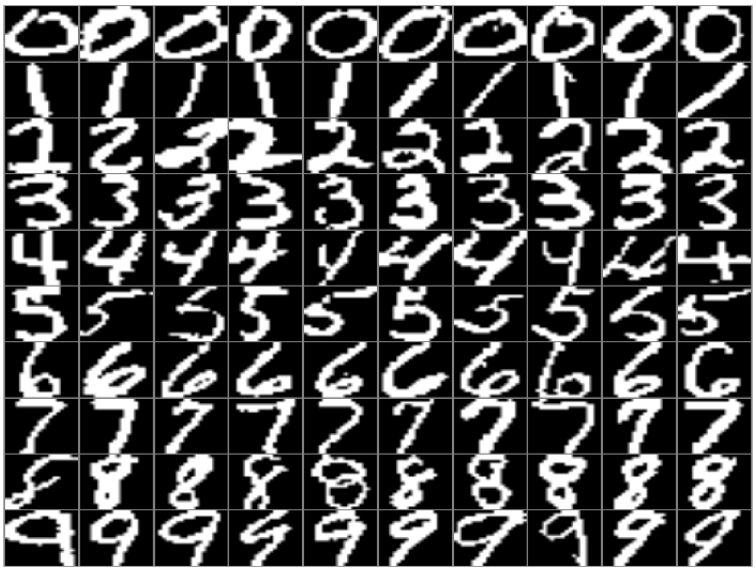


Figure 1: Sample of preprocessed dataset

Furthermore, two proximity measurements were used for the dissimilarity representation, i.e. Euclidian distance and City-Block. The selected proximity measurements were selected based on [3], since it is mentioned that summation-based distances (built from many components) tend to be approximately normally distributed thus give simpler classification problem. Each proximity has an equation like:

1. Euclidean distance: $||A - B||$
2. City-block: $SUM(|A - B|)$

2.2 Feature Reduction

Typically, a large number of samples is required to obtain good performance accuracy when a large number of features is used. This common issue in pattern recognition domain is commonly called "curse of

dimensionality" [4]. To tackle this issue, typically the number of features needs to be reduced by selecting the relevant features in the modeling step, which is known as feature reduction.

In order to get an appropriate number of features, two things are required: criterion function and search algorithm. In the process of selecting the best features, the dataset will be evaluated by using a criterion, such as Euclidean distance, Mahalanobis distance, or inter-intra distance. Euclidean distance was selected in this study since it is frequently used in traditional feature selection techniques [5]. While for the search algorithm, there are several options, such as sequential forward selection, sequential backward selection, and floating search methods. In this study, the sequential forward selection was selected due to its simplicity.

2.3 Classifier Design

There are several classifiers that can be used to classify handwritten digits. In general, they can be classified into three categories: parametric, non-parametric, and advanced. In the experiment, the classifiers investigated in this study were:

1. Parametric: Nearest Mean Classifier (*nmc*), Linear Bayes Normal Classifier (*ldc*), Quadratic Bayes Normal Classifier (*qdc*), Fisher's linear Classifier (*fisherc*), and Logistic Linear Classifier (*loglc*).
2. Non-parametric: K-Nearest Neighbor Classifier (*knnc*) and Parzen classifier (*parzenc*).
3. Advanced: Back-propagation Trained Feed-forward Neural Network Classifier (*bpxnc*) and Support Vector Machine Classifier (*svc*).

The classifier that can give the best performance between the scenarios might be different since the number of samples in the training set is different as well. The path that has been followed in this study was applying the classifiers mentioned above to several representations in each scenario.

In addition, hyperparameter optimisation was also performed in this study. The hyperparameter that can be optimised in each classifier are different, as described in the following list:

1. Number of the nearest neighbors in K-Nearest Neighbor Classifier.
2. Number of units in hidden layer Back-propagation Trained Feed-forward Neural Network Classifier.
3. Regularisation parameter in Support Vector Machine Classifier.

2.4 System Evaluation

Evaluation is one of the most important parts of designing the system. This part will determine whether a pattern recognition technique or a classifier is suitable or not for the system. There are several techniques that can be used to evaluate the system. But, in general, the classification error is estimated by counting the number of wrongly classified samples with known true class labels in the test set.

A common approach to estimate the true error is by using 50-50 split of the design set in equal portions for training and test set, followed by counting the wrong classification in the test set by using the classifier. This will result in a slightly pessimistic estimation of true error [2]. Therefore, in order to obtain higher confidence in estimating the true error, another approach called n-fold cross-validation was used in this

study. The number of n that has been used in this study was 5, as it commonly used yet small enough for this method.

In general, cross-validation was used to evaluate the classifiers. However, due to long training time, the hold-out method was used to evaluate Support Vector Machine Classifier in a few experiments. In addition, feature curves were also used in the feature reduction step. There is a function in PRTools that can be used to analyse the feature curves. This function already provides an evaluation method that can be used to do the evaluation, which is by simply repeating the number of testing. Therefore for the sake of simplicity, we used this built-in functionality instead of cross-validation. However, the number of repetition that has been used was 5, similar to the number of n in cross-validation.

3 Experiment

As mentioned previously, there were two scenarios performed in this study. In the first scenario, the system should use a large number of training set (at least 200 and at most 1000 samples per class). While in the second scenario, the system should use a smaller number of training set (at most 10 samples per class). All experiments for pixels were performed in a computer with Intel Core i7 CPU, while for dissimilarity and image properties were performed in a computer with AMD Radeon R7 CPU.

3.1 Scenario 1

In the beginning, a subset of NIST handwritten images were taken from the whole dataset. In this scenario, the number of training samples can be between 200 and 1000 per class. However, it was decided that only 200 samples per class used in this scenario to speed up the evaluation process using cross-validation. However, there were only 100 samples per class used to benchmark the system.

In this experiment, three representations were used in separated test cases. They are pixels, image properties, and dissimilarity. The experiment and result of using these representations will be described in following subsections.

3.1.1 Pixels

There were a lot of pixels in the original dataset, which made the number of features very large if each pixel is considered as a single feature. In order to tackle the curse of dimensionality issue [4], the original dataset needs to be modified. However, it was still unknown what is the suitable pixel size for this scenario. Therefore, in the first step, the suitable pixel size for this scenario was investigated. Three different pixel sizes were investigated: 10x10, 20x20, and 30x30.

Seven different classifiers were applied to investigate the suitable pixel size, which consisted of parametric and non-parametric classifiers, as can be seen in Table 1. Cross-validation method was used to evaluate the classifiers. Advanced classifiers were not investigated for time-saving.

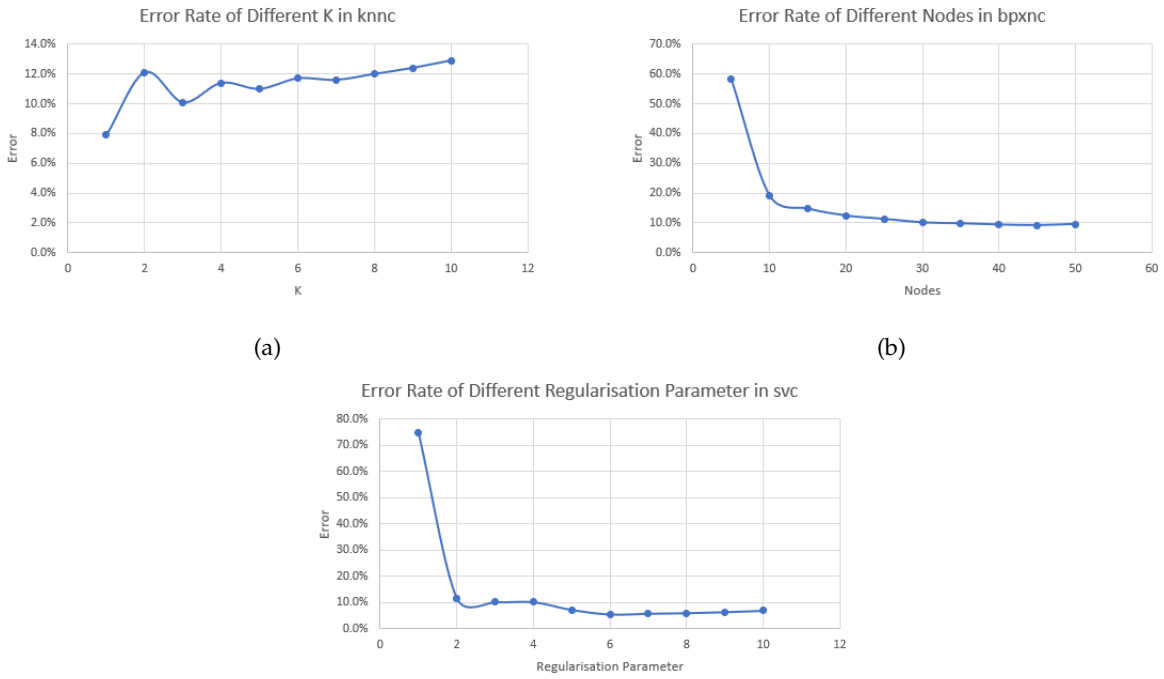
Table 1: Pixel Size Investigation

Classifier	10x10		20x20		30x30	
	Err	Std	Err	Std	Err	Std
nmc	21.9%	0.3%	20.0%	0.2%	90.0%	0.0%
ldc	15.8%	0.2%	17.1%	0.5%	27.9%	0.5%
qdc	47.2%	2.5%	90.0%	0.0%	90.0%	0.0%
fisherc	19.1%	0.3%	21.4%	0.4%	30.3%	0.9%
loglc	49.1%	6.2%	27.1%	0.9%	38.1%	0.7%
1-nnc	12.8%	0.2%	8.1%	0.3%	7.5%	0.2%
parzenc	11.1%	0.3%	8.2%	0.3%	90.0%	0.0%

It can be seen from Table 1 that several classifiers, such as *nmc*, *ldc*, *qdc*, and *fisherc*, were suffered on having larger pixel size or number of features. Only *1-nnc* that had better performance with larger pixel size.

However, the performance of *1-nnc* with 20x20 and 30-30 pixel size were quite similar. Therefore, 20x20 was selected as the appropriate pixel size for this scenario.

After knowing the suitable pixel size, hyperparameter optimisation was performed. Three classifiers were selected in this investigation: *knnc*, *bpxnc*, and *svc*. The reason is because from the previous investigation, it can be seen that complex classifiers, such as *knnc* or *parzenc*, can provide lower error rate compared with simple classifiers, such as *nmc* or *ldc*. As mentioned previously, the hyperparameters were number of nearest neighbours, hidden nodes, and regularisation parameter in *knnc*, *bpxnc*, and *svc* respectively, as can be seen in Figure 2.



(a) (b)

(c)

Figure 2: Results from hyperparameter optimisation of (a) *knnc* (b) *bpxnc* (c) *svc* for pixel representation in scenario 1

The result from hyperparameter optimisation showed that *svc* with regularisation parameter 6 provides the best result with around 5% error rate, as can be seen in Figure 2c. Radial basis kernel was used in the *svc* since it is commonly used [1]. Cross-validation method was used to evaluate *knnc* and *bpxnc*, while hold-out method was used to evaluate *svc*.

Since the number of generated features from using the pixels as the representation was very large, feature reduction was investigated for this scenario. As mentioned previously, sequential forward selection with Euclidean distance as the criterion was selected in this process. The result of feature selection was investigated by using feature curve with the best previous approach in this scenario and representation, which was *svc* with radial basis kernel and regularisation parameter 6, as can be seen in Figure 3.

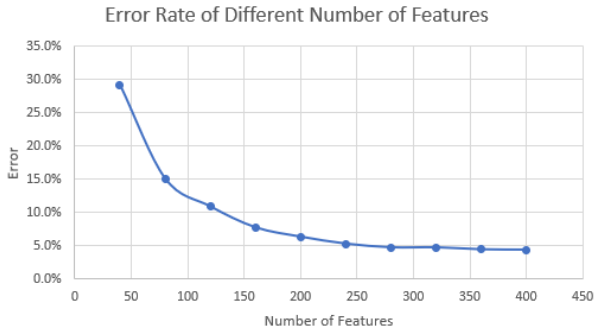


Figure 3: Results of *svc* with feature reduction for pixel representation in scenario 1

It can be seen that the performance of the classifier with 280 features was almost the same as 400 features. Therefore, it can be concluded that feature reduction was helpful in reducing a lot of features with only small performance degradation.

After the best approach was found, the benchmark was performed for this scenario with pixels as the representation. Unfortunately, the error rate was still a little bit higher than the target. In order to achieve the target, which is below 5% error rate, the number of samples in the training data was added from 200 to 400 per class. As a result, the error rate was dropped to 3.8%, which met the target. However, as a trade-off, the training time required to train the classifier was increased from 2552 to 3788 seconds.

3.1.2 Image Properties

Table 2: Test error using features as representations in scenario 1

Classifier	**Err (%)**	**Std (%)**
nmc	61.5	0.1
ldc	15.4	0.1
qdc	16.0	0.6
fisherc	17.7	0.1
loglc	13.0	0.2
1-nnc	55.5	0.4
parzenc	53.6	0.3

In this representation, 20x20 pixel size was used to construct the dataset, based on the previous experiments. After a subset was selected, 24 features were extracted by using *im_features* function in PRTools. However, the average time required to get the features of 400 samples by *im_features* in this scenario was quite long, which was 2759 seconds.

Seven classifiers from parametric and non-parametric types were used for this representation, as can be seen in Table 2. Like in the previous experiments, 5-fold cross-validation was used in the evaluation step. The result showed that no classifiers achieved the target error rate, which is below 5%.

Like the previous representation, the advanced classifiers were also investigated, as can be seen in Figure 4. Unfortunately, none of these classifiers achieved the error rate target. In general, these advanced classifiers performed worse than the previous types of classifiers.

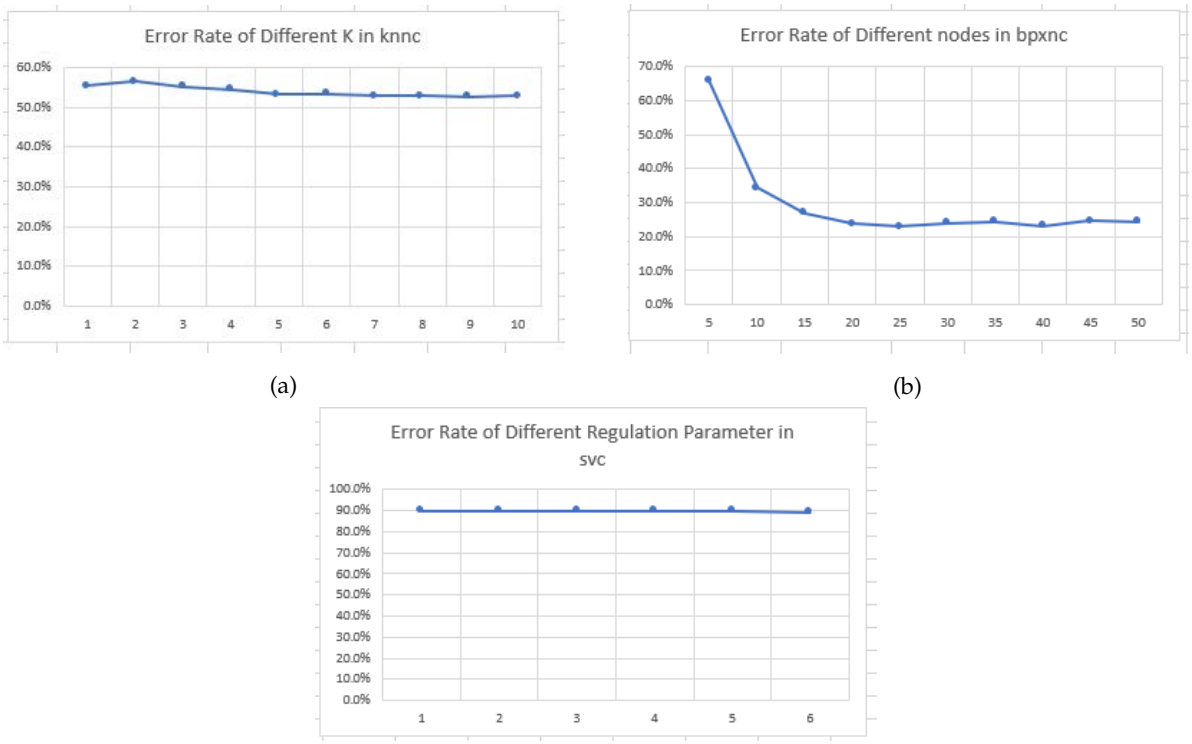


Figure 4: Results from hyperparameter optimisation of (a) *knnc* (b) *bpxnc* (c) *svc* for feature representation in scenario 1

In summary, *loglc* showed the best performance with this representation with 13% error rate. The result from benchmarking was also similar with 13.2% error rate. No further investigation was performed in this case as no classifier achieved the target error rate.

3.1.3 Dissimilarity

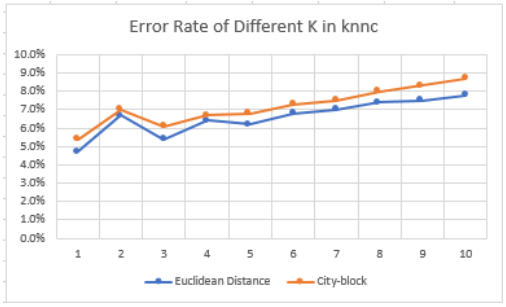
In this representation, 20x20 pixel size was still used to construct the dataset. All training set itself was used as the representation set. After the dissimilarity matrix was obtained, the matrix was mapped into 30-dimensional pseudo-Euclidean subspace. The total time required to get dissimilarity matrix with Euclidean distance and city-block from 400 samples or images was 33.07 and 215.87 seconds respectively.

The result showed that *qdc*, *1-nnc*, *parzenc* and *svc* achieved the target error rate, as can be seen in Table 3 and Figure 5. As expected, dissimilarity representation produced normal distribution. Thus, normal Bayesian classifiers tend to work very well in this representation. One also can see that the classifier using Euclidean distance as its proximity give slightly better result than ones using City-block. The four classifiers were investigated further in the benchmark with Euclidean distance criterion to seek the best one.

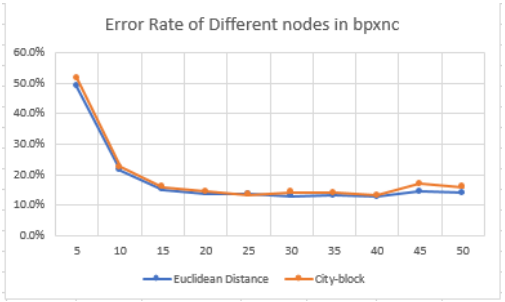
Table 3: Test error using dissimilarity representations in scenario 1

Classifier	Euclidian Distance		City Block	
	Err	Std	Err	Std
nmc	20.8%	0.2%	20.7%	0.2%
ldc	14.1%	0.2%	13.8%	0.3%
qdc	4.6%	0.2%	4.7%	0.1%
fisherc	17.5%	0.2%	16.3%	0.3%
loglc	12.0%	0.3%	12.2%	0.2%
1-nnc	4.7%	0.4%	5.3%	0.3%
parzenc	4.8%	0.2%	5.2%	0.2%

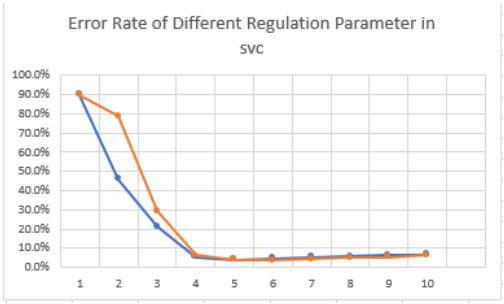
It turned out that *svc* achieved the best result in the benchmark among the best four classifiers in the experiment, as can be seen in Table. 4. However, the result of the benchmark was still quite similar compared with the experiment.



(a)



(b)



(c)

Figure 5: Results from hyperparameter optimisation of (a) *knnc* (b) *bpxnc* (c) *svc* for dissimilarity representation in scenario 1

Table 4: Benchmark testing of dissimilarity representation in scenario 1 using 100 samples

Classifier	Err	Time (s)
svc	3.2%	4745.56
qdc	4.1%	2.768
1-nnc	5%	0.12
parzenc	4.7%	10.855

Since we found more than one good classifiers in this representation, we were interested in combining all of them with advanced classifiers: combiner. However, *svc* was not included in this experiment due to the long training time. Therefore, only *qdc*, *knnc* and *parzenc* that were included in this experiment. Two types of combiners were investigated in this experiment, as described in the following list:

1. Fixed Combiners : Mean (*meanc*), Minimum (*minc*), Maximum (*maxc*), Product (*prodc*), Median (*medianc*), and Voting (*votec*).
2. Trained Combiners : *nmc*, *ldc*, *qdc*, *fisherc*, *loglc*, *knnc*, and *parzenc*.

Table 5: Hold-out Testing for several fixed combiners

Fixed Combiner	Error
meanc	4.62%
minc	4.64%
maxc	4.83%
prodc	4.67%
medianc	4.18%
votec	5.72%

The fixed combiners are evaluated using the hold-out method as *prcrossval* function can receive the fixed combiner classifiers. But the trained combiners is evaluated using cross-validation. The results of several combining schemes is shown in Table 5 and Table 6. It can be seen that *1-nnc* and *parzenc* combiners provide the best result in this experiment. Therefore, these two combiners were tested in the benchmark.

Table 6: Cross validation of several trained combiners

Trained Combiners	Error	Standard Deviation
nmc	5.30%	0.3%
ldc	25.30%	0.9%
qdc	38.30%	1.8%
fisherc	5.60%	0.4%
loglc	4.90%	0.2%
1-nnc	3.90%	0.2%
parzenc	3.70%	0.1%

From the benchmark, it was found that *1-nnc* achieved better result compared with *parzenc*, as can be seen in Table 7. Therefore, it can be concluded that the best approach for dissimilarity representation is *1-nnc* combiners with 3 base classifiers, which are *qdc*, *1-nnc*, and *parzenc*.

Table 7: Benchmark testing on combiners

Combiners	**Error**
parzenc	4.1%
1-nnc	3.6%

3.2 Scenario 2

Like the first scenario, a subset of NIST handwritten images were taken from the whole dataset at the beginning of the experiment. As mentioned previously, the allowed number in this scenario is only 10 samples per class. However, since cross-validation with 5 folds was used to evaluate the system, only 8 samples per class were used to train the classifier in this experiment (4/5 from the maximum of 10 samples per class). Three representations were also used in separate test cases, which were pixels, image properties, and dissimilarity. The experiment and results from using these representations will be described in the following subsections.

3.2.1 Pixels

Firstly, pixel size investigation was also performed for this representation, like the first scenario. Similar number and type of classifiers were also used to investigate the suitable pixel size, as can be seen in Table 8. To evaluate the result, cross-validation method was also performed in this scenario.

Table 8: Pixel Size Investigation

Classifier	10x10		20x20		30x30	
	Err	Std	Err	Std	Err	Std
nmc	29.6%	1.3%	26.9%	1.8%	90.0%	0.0%
ldc	46.2%	6.5%	90.0%	0.0%	90.0%	0.0%
qdc	90.0%	0.0%	90.0%	0.0%	90.0%	0.0%
fisherc	56.8%	2.6%	28.0%	2.7%	28.2%	1.1%
loglc	52.0%	2.7%	26.6%	2.3%	29.7%	2.5%
1-nnc	26.1%	2.1%	25.1%	2.1%	22.1%	2.0%
parzenc	26.1%	1.3%	26.1%	3.0%	90.0%	0.0%

It can be seen from Table 8 that several classifiers, such as *ldc*, *qdc*, and *fisherc*, were also suffered on having larger pixel size or number of features. Like the first scenario, only *1-nnc* that had better performance with larger pixel size. However, the performance of classifiers with 20x20 pixel size was generally good (25%-26%) in a few classifiers. While in 30x30 pixel size, only *1-nnc* that could achieve the error rate target. Therefore, 20x20 was selected as the appropriate pixel size for this scenario.

After having the suitable pixel size, hyperparameter optimisation was performed. Similar classifiers like in the first scenario were also selected, which were *knnc*, *bpxnc*, and *svc*. The hyperparameters that can be optimised were also the same, which were nearest neighbours, hidden nodes, and regularisation parameter for *knnc*, *bpxnc*, and *svc* respectively, as can be seen in Figure 6.

The result from hyperparameter optimisation shows that *svc* with regularisation parameter 4 achieved the best result with around 20% error rate, as can be seen in Figure 6c. Radial basis kernel was still used like in scenario 1. However, the cross-validation method was used to evaluate all classifiers in this scenario, since the training set was smaller, which made the training time much faster compared with the first scenario.

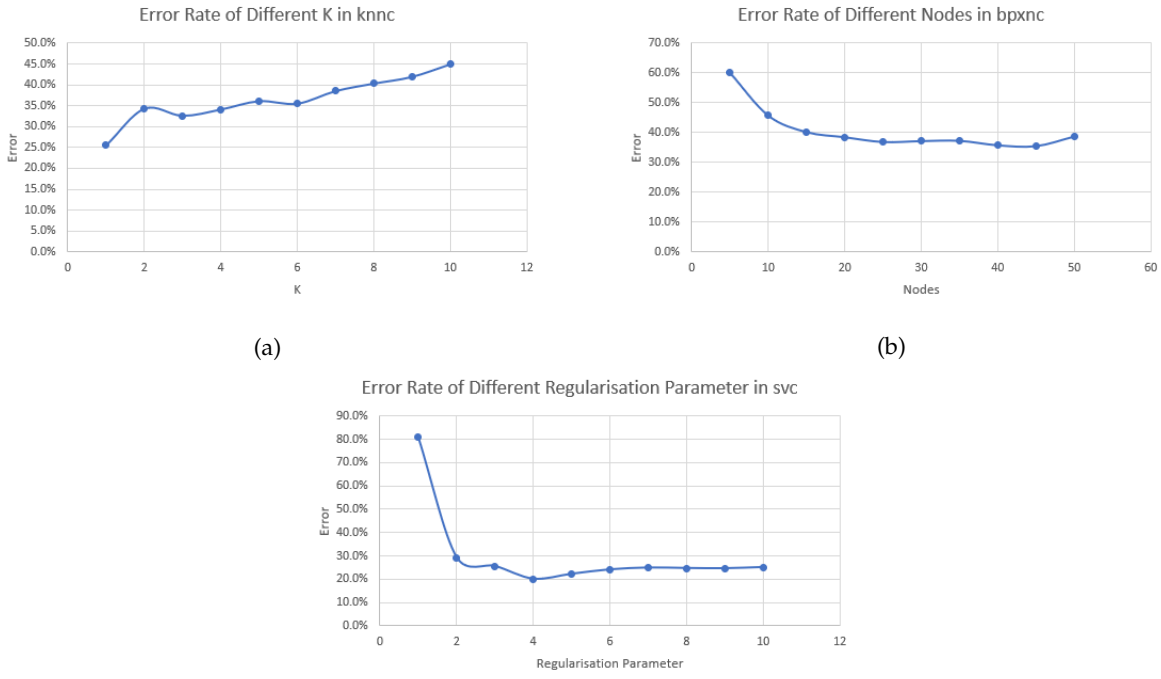


Figure 6: Results from hyperparameter optimisation of (a) *knnc* (b) *bpxnc* (c) *svc* for pixel representation in scenario 2

Like the previous scenario, feature reduction was also performed. A similar approach, which was the sequential forward selection with Euclidean distance as the criterion, were selected. The result of feature selection was then investigated by using feature curve with the best previous approach in this scenario and representation, which was *svc* with radial basis kernel and regularisation parameter 4, as can be seen in Figure 7.

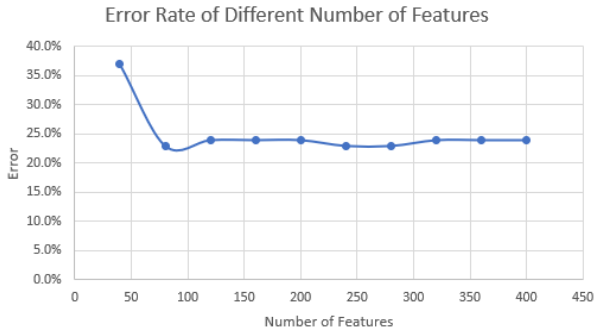


Figure 7: Results of *svc* with feature reduction for pixel representation in scenario 2

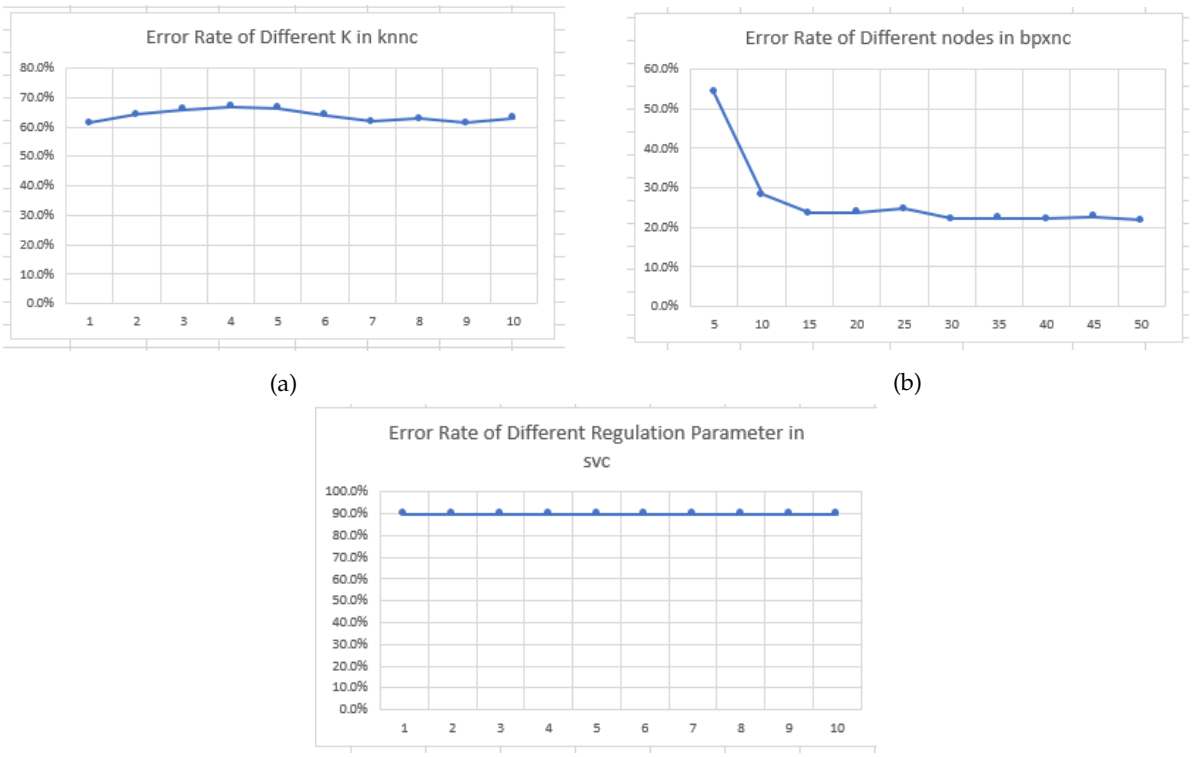
It can be seen that with only 80 features, the error rate was almost the same as 400 features, even better with around 23.0% error rate. However, the result from the benchmark with these selected features was not similar, which was 35.3% error rate. Therefore, it can be concluded that the performance of our approach

Table 9: Test error using features representations in scenario 2

Classifier	Err (%)	Std (%)
nmc	60.6	2.3
ldc	12.7	1.6
qdc	46.8	3.4
fisherc	22.4	3.7
loglc	22.1	3.1
1-nnc	61.0	2.4
parzenc	61.8	1.7

for scenario 2 with pixel representation was not good enough. It seemed that the training data was not sufficient to represent the whole data since the allowed number was only 10 samples per class.

3.2.2 Image Properties



(a) (b)

(c)

Figure 8: Results from hyperparameter optimisation of (a) *knnc* (b) *bpxnc* (c) *svc* for Feature Representation in Scenario 2

In this representation, 20x20 pixel size was used to construct the dataset based on the result of previous experiments. The steps in this experiment were similar compared with the Image Properties investigation in scenario 1. Since the training data in this scenario was smaller, the average time required to get the

features of 10 samples by *im_features* in this scenario was reduced significantly to 66.9 seconds.

It can be seen in Table 9 that *ldc*, *fisherc*, and *loglc* were successfully achieved the target with less than 25% error rate. The *ldc* was even outperform the best approach in pixel representation, with 12.7% error rate. Furthermore, another experiment, which was hyperparameter optimisation in advanced classifiers, was also investigated, as can be seen in Figure 8.

Unlike the first scenario, feature reduction was performed in this scenario with image properties as the representation. However, *ldc* was the only classifier that has been investigated in this experiment since it was the best classifier in this representation. It can be seen in Figure 9 that with 12 features, the performance of *ldc* was almost similar compared with the full features.

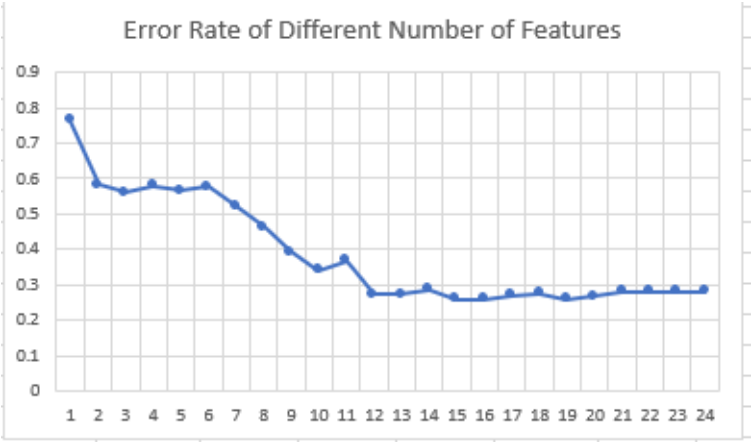


Figure 9: Results from feature reduction of *ldc* for features representation in scenario 1

Unfortunately, from the benchmark which used 100 samples per class, it was found that the performance of *ldc* with only 12 selected features did not achieve the target, which was 34.0% error rate. After that, the full features were tested in the benchmark, and the result showed that *ldc* achieved the target with 23% error rate. Therefore, the best approach for scenario 2 with image properties as the representation was *ldc* without feature reduction.

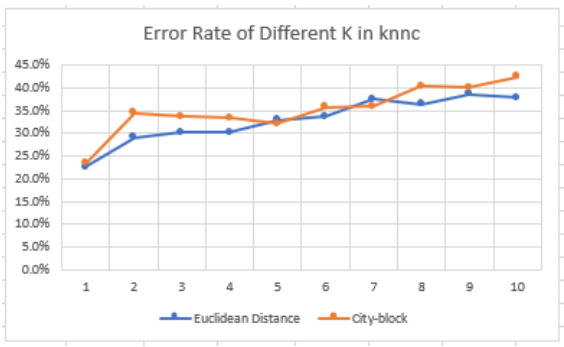
3.2.3 Dissimilarity

In this representation, 20x20 pixel size was used to construct the dataset. The experiments in scenario 2 with dissimilarity representation were similar like dissimilarity experiments in scenario 1. The differences were the number of training samples (10 samples per class) and error rate target (maximum 25%). The total time required to get dissimilarity matrix with Euclidean distance and city-block of 10 samples per class was 0.078 and 0.976 second respectively.

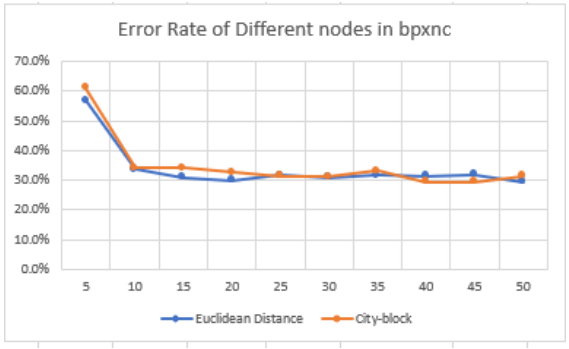
Table 10: Test error using dissimilarity representations in scenario 2

Classifier	Euclidian Distance		City Block	
	Err	Std	Err	Std
nmc	28.2%	2.3%	28.9%	1.7%
ldc	21.0%	2.4%	23.2%	2.1%
qdc	81.2%	1.8%	83.0%	2.4%
fisherc	27.6%	2.3%	28.7%	2.8%
loglc	34.6%	2.9%	29.1%	3.1%
1-nnc	20.8%	2.7%	22.9%	1.4%
parzenc	23.1%	2.5%	25.0%	1.2%

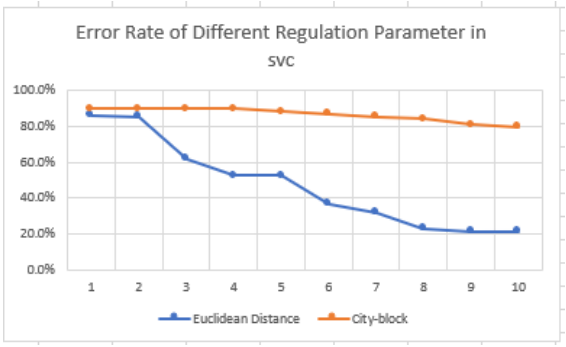
It can be seen in Table 10 and Figure 10 that classifiers *ldc*, *1-nnc*, *parzenc*, *svc* with Euclidean distance proximity and classifiers *ldc* and *1-nnc* with city-block proximity achieved the target with error rate below 25%. However, none of these classifiers with dissimilarity representation outperformed the *ldc* classifier with image properties as its representation.



(a)



(b)



(c)

Figure 10: Results from hyperparameter optimisation of (a) *knnc* (b) *bpxnc* (c) *svc* for dissimilarity representation in scenario 2

The result from the experiment was then validated in the benchmark, as can be seen in Table 11. Unfortunately, the performance of the classifiers was degraded, and none of them achieved the target. Therefore, it

Table 11: Benchmark testing for dissimilarity representations in scenario 2 using 100 samples

Classifier	Error	
	Euclidean Distance	City Block
ldc	25.6%	29.7%
1-nnc	37.1%	35.2%
svc	34.4%	-
parzenc	35.2%	-

can be concluded that dissimilarity representation is not suitable to achieve the target in this scenario.

4 Live Test

In addition to the previous experiment, a live test was also conducted in this study. In the live test, 100 of other handwritten digits from the authors (10 digits per class) were scanned, processed, and used as the test set [1], as can be seen in Figure 11.

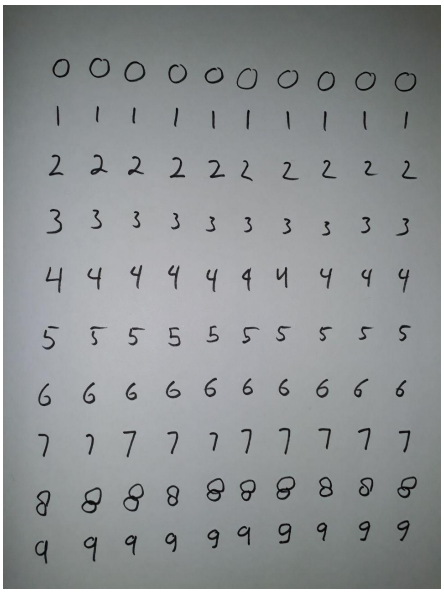


Figure 11: Sample of scanned handwritten digits

To make 100 images of digits, each of them were cropped manually with an image editing tool. After that, the images were imported into Matlab, processed into PRDataset, given label according to its image, and used as the test set. The performance of the classifiers was measured by estimating the classification error.

For live test in scenario 1, 400 samples per class from NIST dataset were used as the training set and the 10 scanned handwritten digit image per class were used as the test set. In this scenario, there were two solutions: using trained combiner (using *1-nnc*) of ***qdc***, ***parzenc***, and ***knnc*** with dissimilarity representation

[1] Available at https://github.com/helmiriawan/IN4085

and *svc* combined with feature reduction. The result showed that the classifier can identify the handwritten digits using combiner and *svc* with error rate of 13% and 17% respectively. Table 12 and Table 13 illustrates the confusion matrix of the live test classification result in scenario 1.

Table 12: Confusion matrix of live test in scenario 1 using *svm* and feature reduction

Actual Label	**Predicted Label**										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
0	7	0	0	1	0	1	0	1	0	0	**10**
1	0	10	0	0	0	0	0	0	0	0	**10**
2	0	1	7	0	0	0	1	0	1	0	**10**
3	0	0	0	10	0	0	0	0	0	0	**10**
4	0	0	0	0	6	0	0	0	0	4	**10**
5	0	0	0	1	0	9	0	0	0	0	**10**
6	0	0	0	0	0	1	9	0	0	0	**10**
7	0	0	0	0	0	0	0	10	0	0	**10**
8	0	0	0	1	2	1	0	0	6	0	**10**
9	0	0	0	1	0	0	0	0	0	9	**10**
Total	**7**	**11**	**7**	**14**	**8**	**12**	**10**	**11**	**7**	**13**	**100**

In scenario 2, the optimal performance was obtained by using ***ldc*** as the classifier and ***image properties*** as the representation. Therefore, in this test, 10 NIST images per class were used as the training set and 10 scanned handwritten digits were used as the test set. Compared to live test in scenario 1, the classifier performed poorly in the live test in scenario 2 with a classification error of 32%. The confusion matrix of the live test is available in Table 14.

Table 13: Confusion matrix of live test in scenario 1 using combiners

Actual Label	**Predicted Label**										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
0	10	0	0	0	0	0	0	0	0	0	**10**
1	0	10	0	0	0	0	0	0	0	0	**10**
2	0	0	9	0	0	0	1	0	0	0	**10**
3	0	0	0	10	0	0	0	0	0	0	**10**
4	0	0	0	0	8	0	0	1	0	1	**10**
5	0	0	0	0	0	9	0	0	1	0	**10**
6	0	1	0	0	0	1	8	0	0	0	**10**
7	0	0	0	0	0	0	0	7	0	3	**10**
8	0	1	2	0	0	0	0	0	7	0	**10**
9	0	0	0	0	0	0	0	0	1	9	**10**
Total	**10**	**12**	**11**	**10**	**8**	**10**	**9**	**8**	**9**	**13**	**100**

With the same scenarios, the classification error from the live test with scanned handwritten digits increases compared with the predicted performance and the result of benchmarking. The different characteristics

Table 14: Confusion matrix of live test in scenario 2

Actual Label	Predicted Label										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
0	4	0	5	0	0	0	1	0	0	0	**10**
1	0	9	0	1	0	0	0	0	0	0	**10**
2	0	0	3	0	0	1	6	0	0	0	**10**
3	0	0	3	7	0	0	0	0	0	0	**10**
4	0	0	0	0	6	0	0	1	0	3	**10**
5	0	0	1	0	0	5	1	3	0	0	**10**
6	1	0	0	0	0	0	8	0	1	0	**10**
7	0	0	0	0	0	0	0	8	0	2	**10**
8	0	0	0	0	0	0	0	0	10	0	**10**
9	0	0	0	0	1	0	0	0	1	8	**10**
Total	**5**	**9**	**12**	**8**	**7**	**6**	**16**	**12**	**12**	**13**	**100**

between NIST digit images and scanned handwritten images or the way we constructed the live-test data might be the reason why the performance of proposed solution degraded. For example, one of digit 4 image has a head with a shape of a triangle, which is looked like 9, as can be seen in Figure 12.

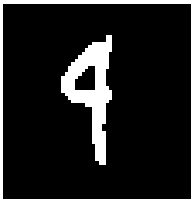


Figure 12: Sample of single digit in scanned handwritten digits

In most tests that were conducted, selected classifier in scenario 1 (with more training data) performed better and yielded lower classification error compared to the classifier in scenario 2. This result is expected as the training process in scenario 1 covers more variance of data compared to scenario 2.

5 Recommendation

There are several things that can be concluded from this study as recommendations. First, more training data can help us to obtain better performance of the classifier. It can be seen that the error rate on scenario 1 was much lower compared with scenario 2. In scenario 1, 200 and 400 samples per class were used in the training phase, while in scenario 2 only 10 samples per class were used.

Furthermore, in both scenario, there was more than one representation that can be used. For scenario 1, the target of test error can be achieved either by using pixel or dissimilarity as the representation. However, the trade-off between the two solutions was time. More time is required on the training phase when using *svc* with pixel representation. If the user has a time constraint in training the classifier, then dissimilarity

representation and combiners approach is more preferred. However, the deployment will require more time from the computational perspective because it needs to do some data processing. Furthermore, in scenario 2, the error rate target can only be achieved by using image properties as the representation. Since the target has been achieved on both scenarios, using rejection for the system was not required.

It can be seen from all experiment that a single classifier can be used to achieve the target. However, especially in scenario 1, the performance for dissimilarity representation could be increased further by using combiners.

To conclude the recommendation, for scenario 1, it is recommended to use pixels as the representations with *svc* as its classifier and feature reduction scheme. If dissimilarity representation is more preferred for scenario 1, it is recommended to use *1-nnc* combiners that combine *qdc*, *1-nnc* and *parzenc*. Meanwhile, for scenario 2, it is recommended to use image properties as the representations with *ldc* as the classifier.

References

[1] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin. Training and testing low-degree polynomial data mappings via linear SVM. *Journal of Machine Learning Research*, 11:1471–1490, 2010.

[2] R. P. W. Duin, P. Juszczak, P. Paclik, E. Pekalska, D. De Ridder, D. M. J. Tax, and S. Verzakov. *PRTools4 - A Matlab Toolbox for Pattern Recognition*, 2007.

[3] E. Pekalska and R. P. Duin. Dissimilarity representations allow for building good classifiers. *Pattern Recognition Letters*, 23(8):943–956, 2002.

[4] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Academic Press, 2009.

[5] H. Vafaie and K. De Jong. Genetic algorithms as a tool for feature selection in machine learning. In *Proceedings Fourth International Conference on Tools with Artificial Intelligence TAI '92*, pages 200–203, 1992.