Pattern Recognition 2018/2019
Classifying Individual Handwritten Digits

Sharon Grundmann - 4973720
Zheng Liu - 4798406
Yan Zhu - 4939980

Group 30

1 Introduction

Handwritten digit recognition is an active topic in optical character recognition (OCR) applications as well as pattern recognition and machine learning research. In OCR applications, digit recognition is dealt with in postal mail sorting, bank check processing, form data entry, etc [4]. For these applications, the performance of digit recognition is crucial to the overall performance. While in pattern classification and machine learning communities, the problem of handwritten digit recognition is a good example to test the classification performance [3].

In this paper, we design and evaluate a system that makes use of existing pattern recognition techniques to classify individual digits in bank account numbers and monetary amount for automatic bank cheque processing. We are interested in two scenarios:
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.

The paper is structured as follows. Section 2 describes the dataset we use for training and testing the system, section 3 discusses the possible representations of the dataset and the actual one used. We explain how we reduce the dimension of the resulting dataset in section 4 and describe the classifiers used in building the classification system in section 5. We evaluate our system in section 6 and present a live test of the system in section 7. Finally, we suggest recommendations for improvement and conclude in sections 8 and 9 respectively.

2 Data

The dataset to be used for training and testing the system described in this paper is constructed from the NIST[1] dataset consisting of 2800 images of handwritten digits for each of the 10 classes "0","1", ..., "9". These were scanned from forms filled out by volunteers, converted into binary images and automatically segmented into images of 128 x 128 pixels.

[1] US National Institute of Standards & Technology

For scenario 1, we select a subset of the data containing 5000 images to be used. That is, we obtain 500 objects from each class to train and test the system. In scenario 2 however, the system is supposed to be trained for small batches of cheques and thus, we choose 10 objects per class resulting in 100 images to train and test the system.

3 Representation

The first essential step of the design process is to select an appropriate way to represent the images. The task of this step is to make the objects quantitatively comparable so that we can distinguish groups of similar images from dissimilar images[2]. This implies that we have to bring all the images to a domain that is open for computations on object differences. Some representations possible include pixels, features and dissimilarities. The chosen representation of objects and the definition of classes determine how the act of learning should be modeled and the performance of the system.

Pixel representation An intuitive way to represent the objects is to use the pixels the images are given in. However, it is not a good idea to feed the pixel-based images directly into a classifier because of the high level of redundancy present in the dataset. Instead, normalization methods like binarization, scaling, segmentation and alignment have to be used to improve the image data. In particular, it is necessary that all object representations have the same size in pixels. Pixel representation generally needs a sufficiently large training set [1].

Feature representation In this approach, objects are represented as feature vectors which contain discriminating information that allow one class to be distinguished from another. In image processing, features can take many forms. It may be the color components, length, area, gradient magnitude, gray-level intensity value, etc [6]. As features can be based on entirely different physical phenomena, their scaling may be different. Normalizing their variabilities may be needed to avoid not-intended built-in preferences for some features [5].

Dissimilarity representation Dissimilarity representation is an alternative for feature representation and it requires us to define a measure that estimates the dissimilarity between pairs of objects. It is useful when the characteristics of objects are not obvious. The dissimilarity representation tries to be sensitive for all object differences, but treat neighboring pixels different than remote pixels unlike pixel representation where neighborhood relations are lost.

In this paper, we make use of pixels together with HOG features as our form of representation. We implemented multiple normalization methods like resizing and centering on the images as this is necessary to enhance recognition of the

[2] `http://37steps.com/720/representation-and-generalisation/`

digits. Figure 1 shows a sample digit after this normalization process with different size parameters. It is clear that 32 x 32 pixel size is a good compromise because it has the edges of the digit better defined than the 16 x 16 resolution and contains 1024 features which is significantly less than that of the 64 x 64 image. Therefore, we choose this together with the bounding box normalization which results in 34 x 34 pixels as our pixel representation.

Fig. 1. Pixel image of digit "0" showing different image resolutions

Furthermore, the resulting pixelated images are merged with their Histogram of Oriented Gradient (HOG) features. HOG features extraction steps mainly involve gradient calculation, histogram generation and block normalization. First, the image is divided into predefined equal sized smaller regions called cells. Each cell has a one dimensional histogram of oriented gradient direction. The gradient orientation is quantized into 9 bins spaced from 0°to 180°. The cells are grouped into bigger units called blocks. These blocks are overlapped and the cells are shared among the neighboring blocks which is helpful for better performance [6]. In figure 2, the HOG features of a sample digit is shown for three different cell sizes. By varying the HOG cell size parameter and visualizing the result, the effect the cell size parameter has on the amount of shape information encoded in the feature vector is clear. Basically, the cell size must be small enough to capture the digit but large enough so that the analysis is efficient.

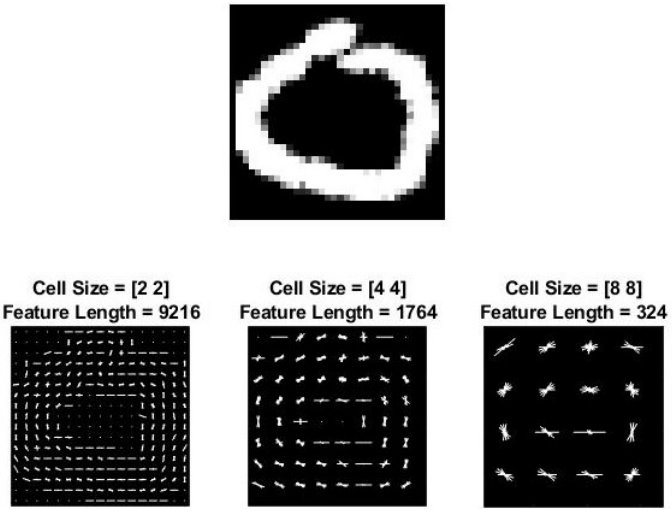

Fig. 2. Extracted HOG features of digit "0" for different cell sizes

The resulting images in both small and large datasets have a total of 1480 features. The pixel representation including boundaries account for 1156 features (34 x 34 pixels) while the HOG features with [8 8] as the cell size parameter generate 324 features for each image.

4 Dimension Reduction

Due to the large number of features derived from the representation, it is necessary to reduce the dimension of the dataset in order to increase classification performance and reduce computational complexity. We realize this by using feature selection to select a subset of relevant features and feature curves to obtain the optimal number of features for both large and small datasets.

Feature selection This approach selects a subset of features that maximizes class separability instead of searching for good combinations like feature extraction does. It can be seen as a combination of a search technique for proposing new feature subsets, along with an evaluation criterion which scores the different feature subsets[3]. In our implementation, we use `eucl-m`[4] which computes the minimum of squared Euclidean distance between features as our selection criterion and forward selection as our search algorithm. Forward selection starts with the best single feature and extends the selection iteratively with the feature that maximizes the performance of the selected set. The result is a ranking of all features in the dataset from the most relevant to the least relevant.

[3] `https://en.wikipedia.org/wiki/Feature_selection`
[4] from Matlab pattern recognition toolbox PRTools

Feature curves The behavior of a classifier's performance as a function of the dimensionality is significant for selecting features and their number. Feature curves show an estimate of the true classification error as a function of the number of features. For classifiers trained by a small training set, feature curves may show a minimum for a low number of features. The larger the training set, the more features can be used but this does not necessarily mean that the performance of the classifier improves. With the help of feature curves shown later in this paper, we obtain the optimal number of features to be used for classification in both the large and small datasets.

5 Classification

After deriving the feature set by dimension reduction, we able to build classifiers that make use of these to perform the classification task. There are a number of classifiers available for this but we only consider a few of them. These classifiers are divided into single types and combined types. Moreover, single type classifiers can be broken down into parametric and non-parametric ones. In the project, we will apply six single type classifiers and one combined type classifier with six different rules.

5.1 Parametric classifiers

A parametric model is a learning model that summarizes data using a set of parameters of fixed sizes[7]. This means that parametric classifiers assume that the data could be fit in a function for classification with a certain set of parameters. As an example, the functions can be Gaussian or Rayleigh function, and the parameters can be mean or the variance of the data. After picking a function, we learn the coefficients from the training data. That is how parametric classifiers are built. Having certain functions make these classifiers simple, fast and also requires less data. However, it will lead to a poor performance when the data does not fit the model.

Nearest Mean Classifier (`nmc`) This classifier assigns an unknown feature vector $\overrightarrow{x}$ to the label of the class of training data whose mean the vector $\overrightarrow{x}$ is nearest to[5]. The algorithm is as following. Suppose we have labeled traning dataset $(\overrightarrow{x}_1, y_1), \ldots, (\overrightarrow{x}_n, y_n)$ with class labels $y_1 \in Y$. Then we can compute mean vector,

$$\overrightarrow{m_l} = \frac{1}{|C_l|} \sum_{i \in C_l} \overrightarrow{x}_i \tag{1}$$

where C_l is the set of indices of samples belonging to class $l \in Y$. Finally, we assign $\overrightarrow{x}$ to label

$$\hat{y} = argmin_{l \in Y} ||\overrightarrow{m_l} - \overrightarrow{x}|| \tag{2}$$

[5] `https://en.wikipedia.org/wiki/Nearest_centroid_classifier`

which means the distance from $\overrightarrow{x}$ to $\overrightarrow{m_y}$ is the smallest.

Fisher's Discriminant (`fisherc`) This is a linear classifier which maximizes the separability among different classes. Suppose we have a training set with two classes with means $\overrightarrow{\mu_0}, \overrightarrow{\mu_1}$ and co-variances Σ_0, Σ_1. The separability of two classes is represented as

$$S = \frac{\sigma^2_{between}}{\sigma^2_{within}} = \frac{(\overrightarrow{w} \cdot \overrightarrow{\mu_1} - \overrightarrow{w} \cdot \overrightarrow{\mu_0})^2}{\overrightarrow{w}^T \Sigma_1 \overrightarrow{w} + \overrightarrow{w}^T \Sigma_0 \overrightarrow{w}} \tag{3}$$

By maximizing S, we obtain the resulting vector $\overrightarrow{w}$, which is used for classification.

Normal Densities Based Linear Classifier (`ldc`) This classifier assumes the data of each class could fit in the Gaussian function,

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4}$$

with identical co-variance matrices. After obtaining the coefficients by training the data, we compute a linear classifier like Fisher's Discriminant.

Normal Densities Based Quadratic Classifier (`qdc`) This classifier is very like `ldc`.It is assumed the data of each class is correspond to Gaussian distribution but the co-variances of the classes are not necessarily identical. Similar to `fisherc`, the resulting vector x can be computed by maximizing

$$S = \frac{\sqrt{|2\pi\Sigma_{y=1}|}^{-1} exp(-\frac{1}{2}(x-\mu_{y=1})^T) \sum_{y=1}^{-1}(x-\mu_{y=1})}{\sqrt{|2\pi\Sigma_{y=1}|}^{-1} exp(-\frac{1}{2}(x-\mu_{y=0})^T) \sum_{y=0}^{-1}(x-\mu_{y=0})} < t \tag{5}$$

for some threshold t.

In order to use these parametric classifiers for training, we first have to know the number of features that yield the best classification performance given both large and small datasets. We do it by analyzing features curves that show an estimate of the true classification error as a function of the number of features in the dataset.

Scenario 1: Large dataset

Figure 3 shows the feature curves of the previously mentioned parametric classifiers for the large dataset. The trend among the classifiers is that there is an abrupt drop in error rate in the beginning which signifies the relevance of the first few features as ranked by the forward selection but this plateaus for most of

the classifiers with the addition of more features. It is also clear that the Normal Densities Based Quadratic Classifier performs best and the lowest classification error is achieved when the feature size is about 50. This however changes when the feature size grows beyond 250 features.

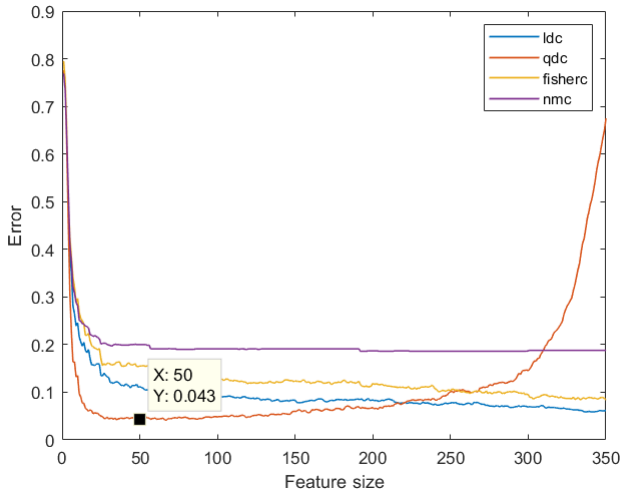

Fig. 3. Large dataset: feature curve for parametric classifiers

Scenario 2: Small dataset

Similarly, figure 4 shows the feature curves of the parametric classifiers for the small dataset. Unlike the feature curves for the large dataset, these show a lot of instability. The huge drop in classification error for all classifiers happen around when the feature size is 10. The complex Normal Densities Based Quadratic Classifier has the worst performance contrary to its behavior in figure 3 which supports the notion that a simple classifier usually performs better when the training set is small. The lowest classification error in this case is achieved by the Normal Densities Based Linear Classifier when the feature size is between 40 and 55. It is also worthy to note that the Nearest Mean Classifier stabilizes best of all with no increase no increase in classification error as the feature size increases.

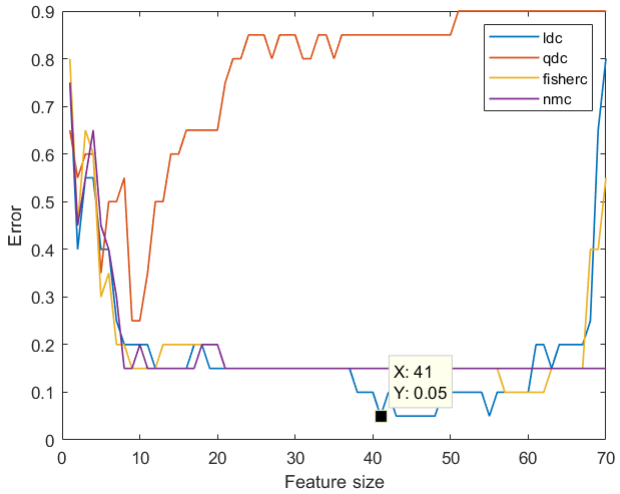

Fig. 4. Small dataset: feature curve for parametric classifiers

5.2 Non-parametric classifiers

Contrary to parametric classifiers, non-parametric classifiers do not make assumptions that the data would fit in a function with a certain set of parameters. This gives the classifiers more freedom about what to learn and we do not need worry about if the data is good enough to fit in the classifiers. Still they are not perfect. They are slow and required a large set of training data.

k Nearest Neighbor Classifier (`knnc`) The input of this classifier consists of k nearest neighbors out of N training vectors and an unknown feature vector $\overrightarrow{x}$ and the output is based on the vote of its k neighbours. That is, $\overrightarrow{x}$ will be assigned to the class most common among its k nearest neighbors. Also distant functions such as Minkowsky, Euclidean and Chi square are needed to calculate the k nearest neighbors [2]. As an example, the Euclidean function can be determined as

$$dist(A,B) = \sqrt{\frac{\sum_{i=1}^{m}(x_i - y_i)^2}{m}} \tag{6}$$

where $A = (x_1, x_2, \ldots, x_m)$ and $B = (y_1, y_2, \ldots, y_m)$ are feature vectors with dimension m.

Parzen Density Based Classifier (`parzenc`) This classifier is similar to `knnc` but the input consists of N training vectors, an unknown feature vector x and a constant h which determines the distance. The vector x is assigned to the

class most common among the neighbors of x in region around x with distance h. The probability of a vector x being in a class $X = (x_1, x_2, \ldots, x_N)$ is

$$\hat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{(2\pi)^{\frac{l}{2}} h^l} exp(-\frac{(x - x_i)^T (x - x_i)}{2h^2}) \tag{7}$$

.

Just as in the previous subsection, we plot feature curves for the non-parametric classifiers described to obtain an estimate of the optimal number of features to use for training both large and small datasets.

Scenario 1: Large dataset

Figure 5 shows the feature curves for the non-parametric classifiers for the large dataset. We can tell that the k Nearest Neighbor Classifier and the Parzen Density Based Classifier have similar behavior. They both achieve a low classification error with `parzenc` achieving an error rate of 0.045 when the feature size is around 50. However, the error rate begins to rise when the feature size gets to 350.

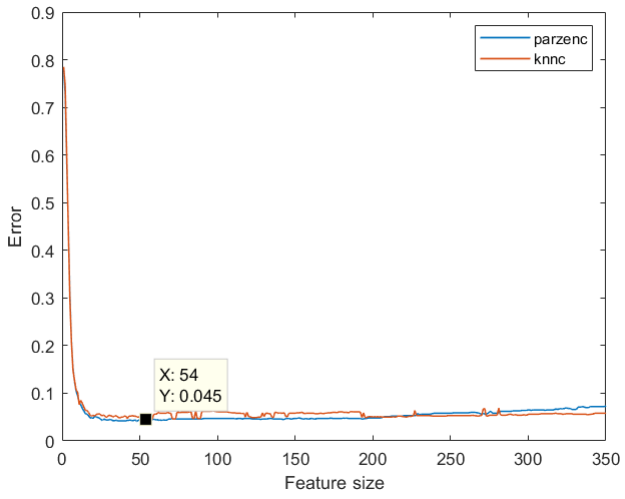

Fig. 5. Large dataset: feature curve for non-parametric classifiers

Scenario 2: Small dataset

In figure 6 however, the non-parametric classifiers exhibit a slight erratic behavior when the feature size is less than 55. This changes afterwards where they both plateau with k Nearest Neighbor Classifier reaching the lowest classification error of 0.05.

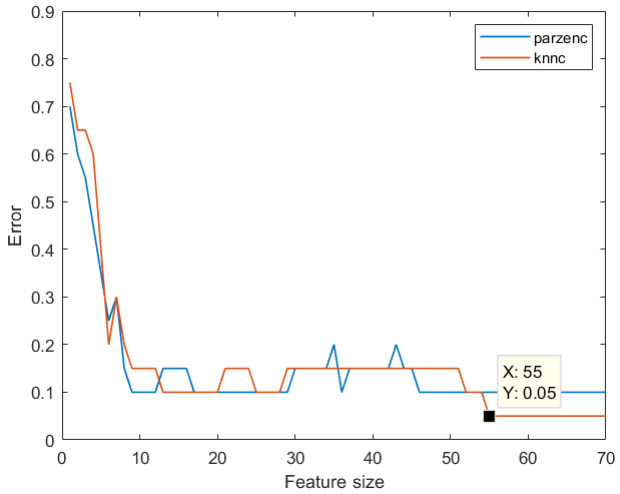

Fig. 6. Small dataset: feature curve for non-parametric classifiers

5.3 Advanced classifier

Combination classifier It is a classifier that combines several classifiers. It aims to achieve a higher performance than each individual classifier by exploiting and applying the advantages of its component classifiers. This type of classifier provides users a high level of designing flexibility. In order to reach the final conclusion with regards to performance, we have to pick a rule to combine the results of the individual classifiers. The rules we pick are the product, sum, min, max, vote, and the median rule. Using these rules, we evaluate combinations of the single classifiers described above to build a combined classifier.

6 Evaluation

Evaluation is an essential part of the design of any pattern recognition system. With our classifiers chosen and our datasets in both scenarios split into training and test sets as shown in tables 1 and 4, we train and test our classifiers. According to the feature curves in figures 3-6, we select 50 from 55 as feature sizes for scenario 1 and scenario 2 respectively. This is because the values are within the range in which the classifiers achieve the lowest classification error.

6.1 Scenario 1: Large dataset

For scenario 1, the data set is divided according to the values shown in table 1. There are 400 objects per class for training and 100 objects per class for testing with 50 as the feature size as mentioned previously. The test results of the six

classifiers are shown in table 2. It shows that `parzenc`, `qdc` and `knnc` perform best with error rates 4.4%, 4.38% and 5.2% while the error rates of the other classifiers being around 15%.

We choose 2 out of 3 of the classifiers which perform the best as the combined classifier. After some testing and evaluation, we find the combination of `parzenc` and `knnc` performs the best and the result is shown in table 3. The combination classifier with product rule has the lowest error rate, so we use it as the final classifier for Scenario 1.

	Objects per Class
Training Data	400
Testing Data	100
In Total	500

Table 1. Data Set Size (Scenario 1)

	Classification Error
nmc	20.0%
ldc	11.2%
qdc	4.38%
fisherc	15.5%
parzenc	4.4%
knnc	5.2%

Table 2. Classification Error of Individual Classifiers (Scenario 1)

	Classification Error
maxc	5.2%
minc	4.4%
meanc	5.2%
prodc	4.3%
medianc	5.2%
votec	5.1%

Table 3. Classification Error of Combined Classifiers (Scenario 1)

6.2 Scenario 2: Small dataset

The small data set size with feature size 55 is presented in table 4. The results in table 5 prove that the best three classifiers are `ldc`, `knnc` and `parzenc` with error rates 5%, 5%, and 10%.

Just as mentioned in Scenario 1, we selected `knnc` and `parzenc` as our combination classifier with different rules. Since the results for the max, mean, median rule are the same, our final combination classifiers are all three of them.

	Objects per Class
Training Data	8
Testing Data	2
In Total	10

Table 4. Data Set Size (Scenario 2)

	Classification Error
nmc	15%
ldc	5%
qdc	9%
fisherc	15%
parzenc	10%
knnc	5%

Table 5. Classification Error of Individual Classifiers (Scenario 2)

	Classification Error
maxc	5.0%
minc	10.0%
meanc	5.0%
prodc	10.0%
medianc	5.0%
votec	10%

Table 6. Classification Error of Combined Classifiers (Scenario 2)

6.3 Benchmark

To test our system and verify whether the performance we obtain is actually valid, we are provided a test function to benchmark our system. For scenario 1, our system should have lower than 5% test error and for scenario 2, our target is 25% test error. From the results presented in 7, it is clear that our `knnc` and `parzenc`combined classifier meets the targets. The error rate for Scenario 1 is 4% which is under 5%, and other three classifiers for Scenario 2 are all under 25%. And it is worthy to note that we have done the benchmark for the combined

classifiers with all different rules. Surprisingly, the combined classifiers with the vote rule for both Scenario 1 and 2 have the lowest error rates of 1% and 12%.

	Classification Error
Scenario 1	4%
Scenario 2 with maxc	13.0%
Scenario 2 with meanc	13.0%
Scenario 2 with median	16.0%

Table 7. Benchmark of Combined Classifiers

7 Live Test

For testing our classifier in a more general situation, we attempted to do a live test on hand-written digits. We wrote a 10×10 digits grid. In the coding part, a cutting function was designed for cutting that grid into 100 separate digits pictures and attach labels to the digits. Then, the hand-written digits were transformed into the `prdataset` for further processing.

We used the trained classifier of scenario 1 to test the error rate of our hand-written digits. However, the result was not as good as the performance in scenario which was expected. The error rate reaches to 90%, for which we think it is rather unreasonable but we did not get the expected result yet currently. The feature of the pre-processed figures are shown in figure 7. The code is still submitted which describes about the work we did in this part in detail.

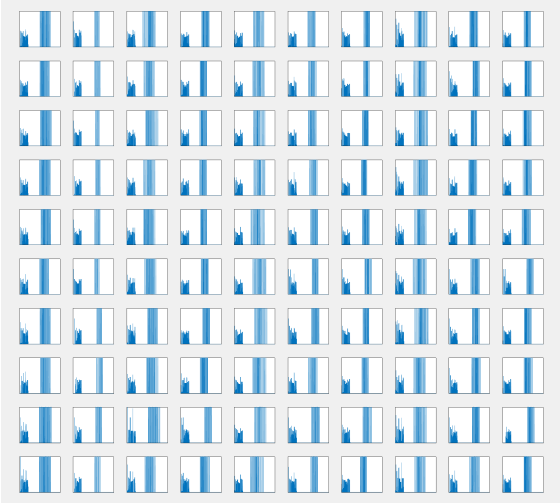

Fig. 7. Features of the pre-processed digits

8 Recommendation

Although our system meets the requirements of the client, we suggest the following recommendations to help improve the overall performance of the system. The first has to do with the relationship between the size of the training set and the performance of the system. For small training sets, it is good to use simple classifiers. Using complex classifiers in this case might result in a worse error rate. Another problem that needs attention has to do with computational complexity of training the classifiers. Time and storage space must be kept in mind in order to run the system efficiently. Another classifier that could be considered for the recognition task is the Support Vector Machine. It is known to perform well and does not suffer from the curse of dimensionality although it has a high time complexity.

9 Conclusion

In this paper, we applied our pattern recognition knowledge to create a system that recognizes hand-written digits provided by the NIST dataset. Using representation and dimension reduction techniques, we transformed the set of images into datasets containing relevant features used for training in two scenarios. The first scenario had to with training a large set of images and using this for testing while the second only used 10 objects per class for training and testing. We evaluated the performance of different individual classifiers and picked the best pair, `knnc` and `parzenc`to create our combined classifier. We were able to achieve an error rate of 4% and 13% in scenarios 1 and 2 respectively, meeting the target of our clients.

References

1. R. P. Duin and E. Pekalska. The dissimilarity space: Bridging structural and statistical pattern recognition. *Pattern Recognition Letters*, 33(7):826–832, 2012.
2. L.-Y. Hu, M.-W. Huang, S.-W. Ke, and C.-F. Tsai. The distance function effect on k-nearest neighbor classification for medical datasets. *SpringerPlus*, 5(1):1304, 2016.
3. Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, et al. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60. Perth, Australia, 1995.
4. C.-L. Liu, K. Nakashima, H. Sako, and H. Fujisawa. Handwritten digit recognition: benchmarking of state-of-the-art techniques. *Pattern recognition*, 36(10):2271–2285, 2003.
5. D. R. PW and P. Elzbieta. *Pattern Recognition: Introduction and Terminology*. 2015.
6. H. Rehana. Bangla handwritten digit classification and recognition using svm algorithm with hog features. In *Electrical Information and Communication Technology (EICT), 2017 3rd International Conference on*, pages 1–5. IEEE, 2017.
7. S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.