Handwritten Digits Image Classification

Septian Gilang Permana Putra (4609328)
Yadong Li(4608283)
Runfeng Tian (4595912)

January 30, 2017

Contents

1 Introduction

Handwriting is one of the personal skill for communicating with the other individual. It has some standard rules related to the language which is used for the communication process[9]. During the last century, people have invented some technologies which can be alternative to it. All these inventions have led to the reinterpreting of the handwriting role in our daily activity. The reason that handwriting persists in the age of the digital computer is the convenience of paper and pen as compared to other tools for numerous situations[8].

Previously, humans read the information presented in handwriting and enter the data into a computer manually to make it digital. However, as the number of data increases rapidly, people need the ability to automatically recognize handwriting and process it into electronic data which can be utilized for many purposes, including to recognize account numbers or the monetary amount for automatic bank cheques processing applications[8]. A significant amount research has been done covering the recognition of handwritten digits and characters area[6]. Those research mainly focused on comparing different classifiers to decrease the classification error.

In this report we will provide the results of our handwritten digit recognition system that we have developed. Our system will be used to classify individual digits in bank account numbers and the monetary amount. We focused on two scenarios to test the accuracy of our system with different target for each scenario:

1. We train the pattern recognition system once with 1000 objects per class. For this scenario, system should have lower than 5% test error.

2. We train for each batch of cheques with 10 objects per class. The test error for this scenario should be below 25%.

This report is organized as follows: We first present the representation of data we chose in Section 2 and explain about feature extraction and reduction which is used in Section 3. Section 4 will elaborate the optimal classifier and how it has been tested when building our system. In Section 5, we explains the result of evaluation to our systems. Section 6 goes into detail about the "("Live Test), which test performance of our system on actual handwritten digits. In Section 7, we provides recommendation regarding the actual implementation of handwritten digit recognition system. Finally, we present our conclusion in Section 8.

2 Representation

2.1 Choice of Representation

In order to classify each object properly, we should bring them in the same domain which represent themselves first. There are many choices of how we can represent the picture to be compared, such as pixel, feature, and dissimilarity representation. The process of finding a good representation is the most important step in the design of the system as it is possible to make mistakes here which can not be corrected in the next step. A good representation will make the difference between objects from difference classes is big enough to be separated while the difference between objects within the same classes is small enough to be generalized[3].

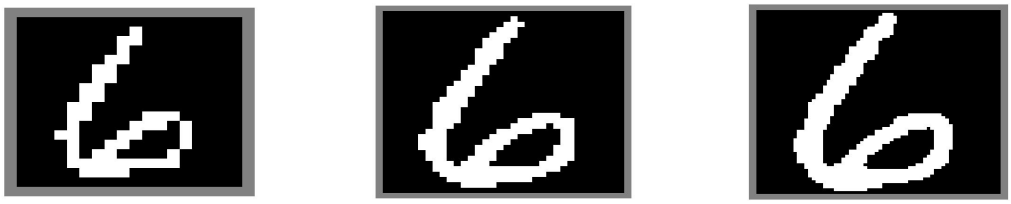
Feature are well suited to represent objects by numbers if it is known where to look for and if this knowledge can be formed in measurable quantities. For example, if we want to classify oranges and watermelons, the color and weight can be a good feature to represent each class. However, if it is not certain what the characteristics and properties of each class are, a number of irrelevant ones might be taken and also the essential ones may be overlooked[1]. As human knowledge is not always specified in measurable properties, choosing features to represent the object can be problematic in some cases, including ours. So, we are not considering the feature representation to be used in this case.

Dissimilarity can be formed directly on raw or preprocessed measurements. The use of dissimilarities as representation is advantageous when characteristics and properties of objects are difficult to obtain or when they have a little discriminative power. Such situations happen when there is no straightforward way to define features which are caused by either the data is highly dimensional or the features consist of both continuous and categorical measurements. For a small set of training data, the dissimilarity will outperform the other two representation[4]. However, there is an issue when selecting a good dissimilarity measure for the classification problem which caused by not only there are plenty of different possibilities, but also there is no convincing argument to prefer one measure over another. Moreover, in order to get a good enough classification error, the objects in the representation set should be selected carefully as as it can affect the classification performance and will require huge storage and computational cost if the size is too big/citepkekalska2002dissimilarity.

An image can be described by a set of pixels organized in a square grid. Pixel is a true representation of a 2-dimensional object like a handwritten digit as the pixel values are all there is to describe the object. Another almost identical set of pixels represents an almost identical object. Given that condition, it is impossible that such objects are entirely different if there are no hidden features like color or weight[2]. Pixels based classifiers do not use the spatial connectivity of the pixels in the images, so the pixel representations can only be used for large enough collections of images. If there are large enough of data, the pixel representation will perform the best compared to other two representation[4]. In this project, we chose pixel over both feature and dissimilarity representation considering that the small training set which is given in Scenario 2 come with the higher classification error as its target. We believe that the classification error for both scenarios can be achieved by using pixel representation.

2.2 Actual Representation Used

In the previous chapter, we have decided that the pixel representation will be used in our system. However, the images inside the NIST dataset contains various pixel size which means it does not have the same feature size to be compared. So, in order to be able to work with the dataset, we need to preprocess every image in the dataset so that all of them has an equal size. Previous work which also uses this dataset resized every image into 32x32 pixels. The choice of using 32x32 pixels size seem reasonable as if we compare it to the other pixel sizes as shown in Figure 1, it still clear enough in representing the digits compared to the 16x16 and also has less feature size compared to the 64x64. After we resized the picture in the data set, there is another problem which comes from the quality of the images. Some



(a) 16x16 pixel size (b) 32x32 pixel size (c) 64x64 pixel size

Figure 1: Comparison of image with the different pixel size

images in the dataset have noises, which may cause the images from the same label have different profiles. So to deal with the noises, we applied the uniform filter to the images before thresholding it to get the 32x32 size binary image, which means 1024 features for every image. All of these features may not have the same level of importance in discriminating one class from the others. In the next section, we will try to reduce the dimensionality of the data, so that every image only holds important feature in our classification problem.

3 Dimensionality Reduction

3.1 Feature extraction

As we discussed before, if we choose to represent features by pixels, the total dimensions of the feature space will be considerably large. What's more, the pixels at the various positions in the image might be largely correlated, thus a large amount of redundant information might be include if we use all pixels as features.

To avoid this large amount of data and potential redundancies, dimensionality reduction and feature extraction technique is needed. We apply Principal Component Analysis (PCA) here to reduce dimensionality and extract new features. Basically, what PCA does is to generate a set of new features and maintain as much variance between pixel features as possible. Meanwhile, all the newly extracted features are sorted according to how much variance retained. The first dimension retains the most of the total variance, as the i-th feature is the i-th principal component.

The sorted order of the features allows us to do feature reduction easily. Figure 2 shows the relationship between dimensionality and variance retained for a certain dimensionality. The PCA is based on 2000 samples, each sample is represented as 32*32 pixels and the original dimensionality is 1024. From Figure 2, we can tell that if 100 features are chosen, nearly 80% of variance is retained. 90% of variance can be caught if we choose around 240 features. This indicates that PCA can reduce dimensionality dramatically without losing much variance.

3.2 Dimensionality reduction

In order to decide to what extent dimensionality reduction should be, we will introduce feature curve to reveal the relationship between feature size and estimated classification error. In PR tools, we can build the feature curve conveniently using "levalf", which can compute the classification error by cross-validation for each size of feature and for different classifiers simultaneously. In the next discussion, large

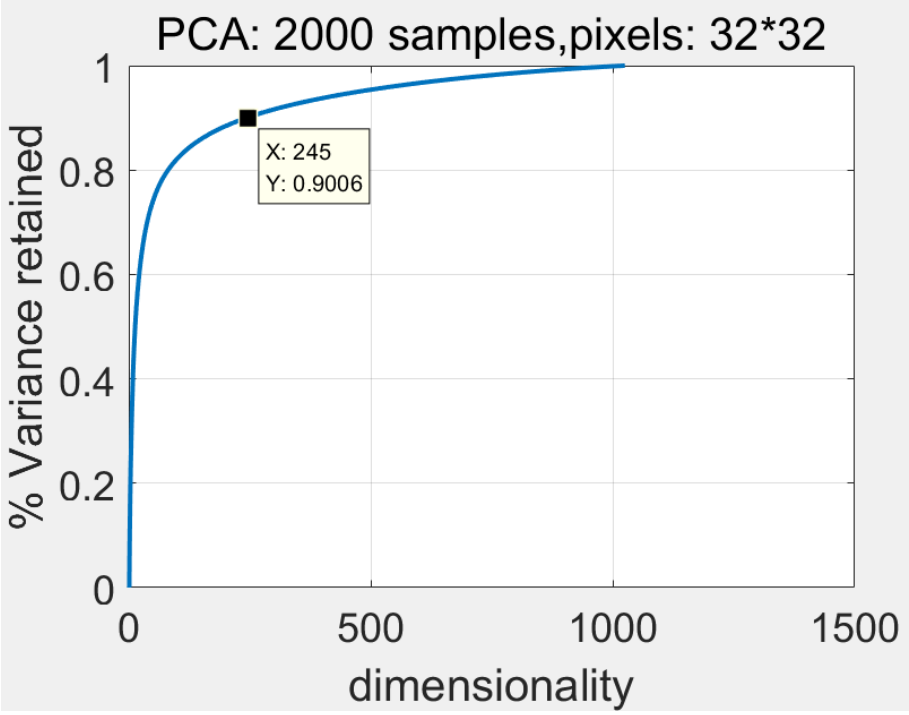


Figure 2: PCA features: dimensionality against variance retained

training data and small training data will be discussed separately.

3.2.1 Large training data

For large training data, we've build three feature curves for different types classifiers: parametric, non-parametric and advanced ones. A brief description of these classifiers will be provided in the next chapter. The training data here is quite large: 500 objects per class and 5000 objects in total.Figure 3 shows the feature curve for parametric classifiers. We can find that Quadratic Discriminant Classifier (Bayes-Normal-2, QDC) have the best performance when feature size is about 30. The general trend here is that as feature size increases, errors go down greatly at first and then into a phase of plateaus. For more complex classifiers like QDC, the curve goes up when feature size is too large. Similarly, feature curves for non-parametric and advanced classifiers are shown in Figure 4 and Figure 5 . We can tell that knn and parzen classifiers act similarly expect that knn is more stable when feature size gets large. For advanced classifiers shown in Figure 5 , they are more likely to be affected by the increase of feature size.

All these feature curves reveal that more features is not always better. When feature size is in the range of 20 to 30, classifiers generally get the best result. The error rate decreases at first, due to the increase of information gained from the added features. However, if there is too much information added, advanced or more complex classifiers are more likely to "overfit" the training data, which means that classifiers tend to fit the unnecessary pattern in training data and thus testing error gets increased.

For the large training set scenario, an arbitrary decision is made here: we will use the first 30 features. By choosing 30 features extracted by PCA, we successfully reduced 97.07% of dimensionality but retained 69.54% of variance, as shown in table

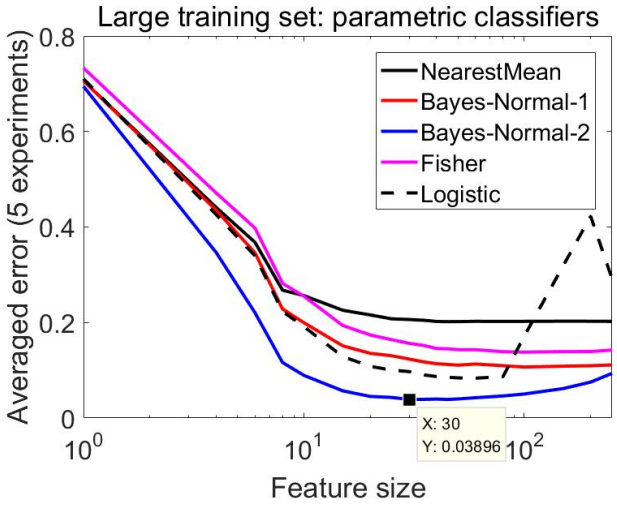


Figure 3: Large training set: feature curve for parametric classifiers

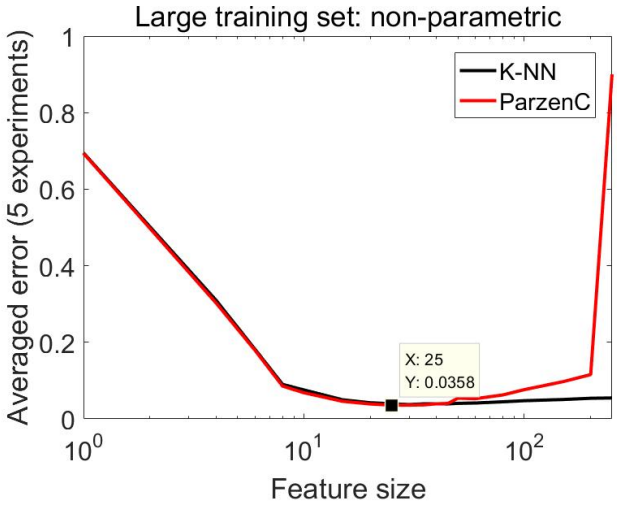


Figure 4: Large training set: feature curve for non-parametric classifiers

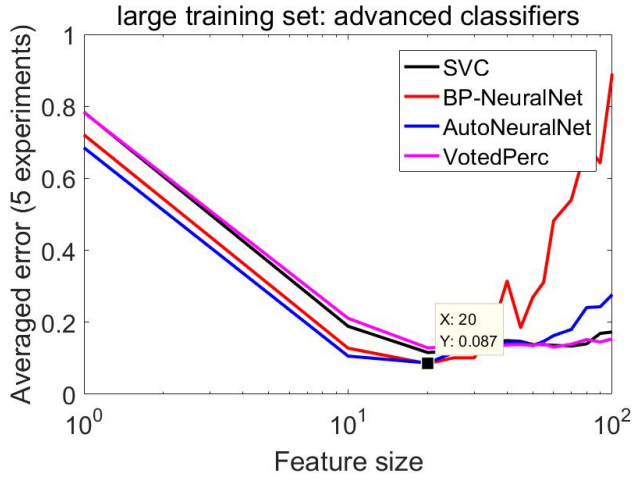


Figure 5: Large training set: feature curve for advanced classifiers

1 .

Feature	Dimensionality	Reduced	Variance retained
32*32 pixels	1024	N.A	100%
PCA(245 features)	245	76.07%	90.06%
PCA(30 features)	30	97.07%	69.54%

Table 1: Comparison among raw pixel data, PCA with 245 features and 30 features.

3.2.2 Small training data

As the size of training data is relatively small, classifiers may have a different performance comparing to large training data. This can be evaluated by drawing feature curves for different classifiers, as we did before for large training data.

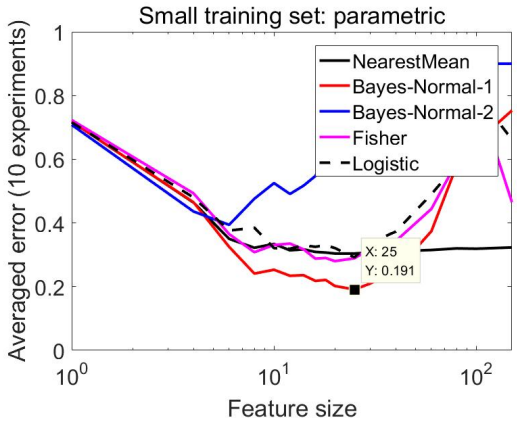


Figure 6: Small training set: feature curve for parametric classifiers

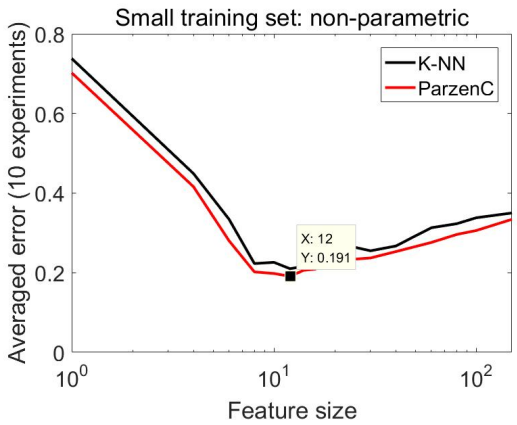


Figure 7: Small training set: feature curve for non-parametric classifiers

Figure 6 7 8 shows that when feature size is really small, the performances for almost all the classifiers we tested before degrade a lot. Another point worth noticing is that if the training size is small, most of the classifiers overfit when the feature size is relatively large. There are some exceptions though, such as Nearest Mean classifier and , it's performance is stable when feature size is getting large.

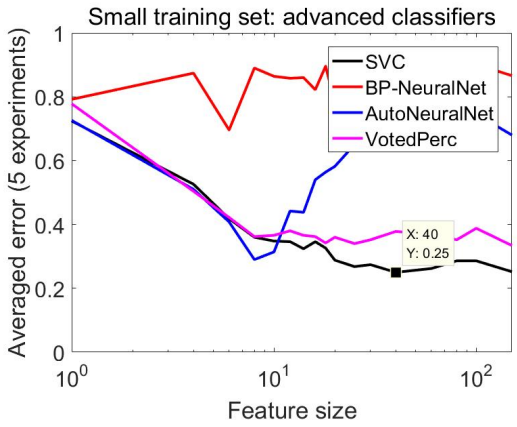


Figure 8: Small training set: feature curve for advanced classifiers

For the small training set scenario, an arbitrary decision is made here: we will use the first 20 features extracted by PCA.

4 Classifiers

4.1 Analysis in principle

4.1.1 Baye's classifiers

Baye's classifiers are based on Baye's rule,

$$P(\omega|x) = \frac{P(x|\omega)P(\omega)}{P(X)},$$

where $P(x)$ is the data distribution, $P(\omega)$ is the class prior, and $P(x|\omega)$ is the class distribution. The data distribution is the same for all classes so it is not taken into account. And we assume that the a priori probabilities are known - in this case, it is 10% for all classes. So the mean goal is to estimate the likelihood function $P(x|\omega)$. Basically, there are two ways, make a parametric model (e.g. ML, MAP), or use non-parametric model (e.g. Parzen, kNN, Naive-bayes, Bayesian network). Following is the analysis and test for each classifier.

Parametric: ML,MAP

For both the maximum likelihood (ML) and the maximum a posteriori (MAP) method, an assumption of the type (e.g. Normal, Gaussian) of the probability density function should be made. The ML method estimates the parameters θ so that the likelihood function takes its maximum value; while the MAP estimate $\hat{\theta}_{MAP}$, at the point where $P(\theta|x)$ becomes maximum. These two methods are both easy to implement. While in real case, it is hard to make a correct estimation of the pdf model.

Non-parametric: Parzen classifier and k Nearest Neighbor classifier

The Parzen windows method assume the pdf as an average of N Gaussians, with each one centered at a different point of the training set. So $p(x)$ is then,

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

where h is the smoothing parameter. For fixed training set, the smaller the h the higher the variance. And for a fixed h, the variance decreases as the number of sample points N increases. So the choice of h is very important for Parzen windows classifier. The normal way is to start with an initial estimate of h and then modify it iteratively to minize the resulting misclassification error. In the case of scenarios 1 with 1000 objects per class, the value of h varies from 0.1 to 2 to test the misclassification error. The result is shown in Figure 9. From the result, we can

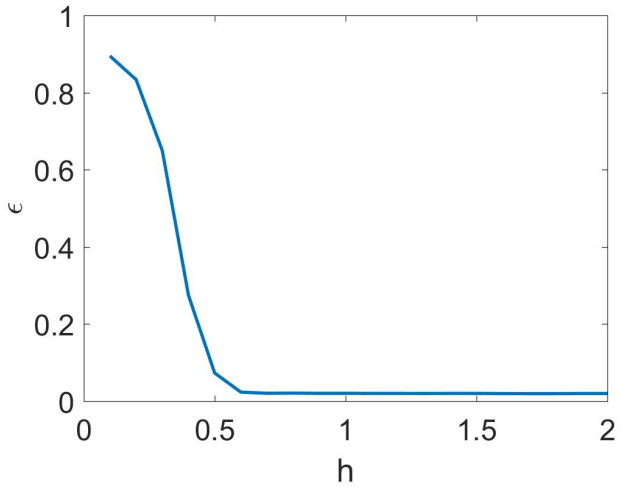


Figure 9: Misclassification error of Parzen Windows classifier with various h

see that h larger than 0.6 gives the best result, with misclassification error equals to 2.0%. This classifier satisfies the requirements for scenarios 1.
The k-Nearest Neighbor classifier predict the pdf by measuring the distance between the data point and training points. The distance could be Euclidean, Mahalanobis or others. The estimator can be written as:

$$\hat{p}(x) = \frac{k}{NV(x)},$$

where k is the parameter indicates how many nearest neighbor wanted to be included; N is the total number of training data and $V(x)$ is the volume of the hypercube contain k nearest neighbor. Basically, there are two ways to implementation the k-Nearest Neighbor algorithm: One way is to take grid in the hyperspace and estimate the pdf on every point; another way is just storing all the training examples without any learning algorithm. The selection between these two method depends on the size of the training set and the hyperspace. The performance of k-NN classifier with varying k from 1 to 10 is shown in Figure 10. The best result appears at k equals to 1, with the misclassification error 2.1%. This classifier satisfies the requirements for scenarios 1, but the performance is worse than Parzen windows classifier.
The Parzen classifier and k-NN classifier works in the similar way. Both of them need relatively large training sets, and are very sensitive to the scaling of the features. Also, they need to store the whole training set, and to compute predictions is more costly than for say linear models. The advantages of these two classifiers are that they are very simple and can give good performance. Also, the complexity of the classifiers is adaptive by tuning the parameters h and k.

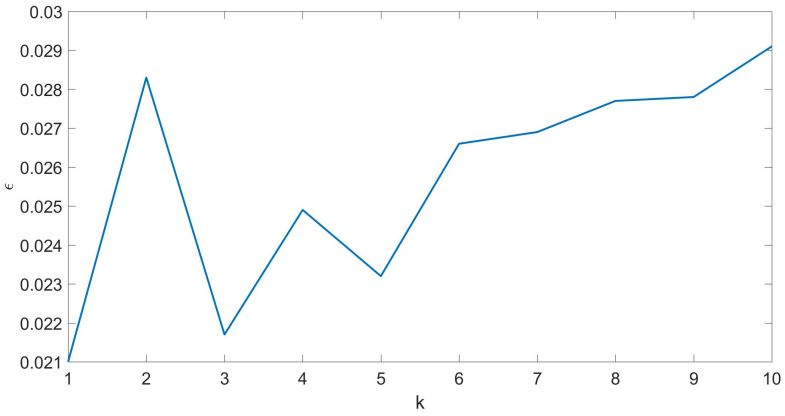


Figure 10: Misclassification error of k-NN classifier with various k

Non-parametric: Naive-Bayes and Bayesian network
The Naive-Bayes classifier assume all the features are independent. So we can estimate $P(x_i|\omega)$ per feature and multiply them. By using this way, we can avoid the curse of dimensionality. However in this case, it is almost impossible that two features are totally independent, no matter which representation is used.
The Bayesian network works as follows: " The parents of a feature,x_i,are those features with directed links toward x_i and are the members of the set A_i. In other words,x_i is conditionally independent of any combination of its nondescendants, given its parents. There is a subtle point concerning conditional independence"(Sergios 2011). The performance of Bayesian network should be better than Naive-Bayes for the task in principle. However, it's hard and time consuming to find the network topology, which determines the performance of Bayesianed not to use this classifier.

4.1.2 Linear Classifiers

Another way besides using Bayes' rules is to assume the decision boundary directly. The hyper-plane that divided the classes could be assume linear or nonlinear. Both cases will be introduced and tested.
Perceptron
The perceptron classifier assume the classes are linearly separable and tuning the decision hyperplane step by step after assuming an arbitrary initial condition. The perceptron classifier is easy to implement and will get a "correct" result for linearly separable classes. While for linearly non-separable cases, although a tolerance factor can be set, the perceptron classifier will not give a good result. So it is not used in this case.
Fisher,Least squares,Logistic,SVM
The fisher classifier tries to maximize the separability indicated by the fisher criterion, $J_F = \frac{\sigma^2_{between}}{\sigma^2_{within}}$. The least squares classifier tries to minimize the cost function $J(w) = E[|y - w^T x|^2]$. The logistic classifier tries to maximize the likelihood function. And the SVM classifier tries to maximize the distance between the support vectors. All these linear classifiers can be changed into nonlinear form by applying the 'kernel-trick'.

4.1.3 Non-linear Classifiers

Neural Networks
Neural networks is a good way to solve linear unseperatable problems (e.g. XOR). And there are examples of recognize hundwritten digits by neural networks, which gave good results.[7] However, it is hard to determine the proper topology of the neural networks and the automatic neural networks function contained in PRTools didn't give a good result. So we decided not to use neural networks in the project.
Combining, Boosting
Suppose there are several classifiers both perform well and you don't know which one to choose. One way to exploit their individual advantage is to combine them. In scenario two, we will try to combine different classifiers to get a better performance and stability.
Boosting is another powerful technique to train several base classifiers into a strong classifier. It basically awards classifiers which has a good performance and focus on the data which is misclassified. It is proven to be effective to solve two-class classification.

4.2 Choice of classifiers

4.2.1 Methodology

The general idea here is to find the "best" classifier for each scenario. By saying "best" we mean the classifier which has the least estimated error in our experiments. In both scenarios we will split our data into training set and validation set and do cross validation to estimate the performance. After we've find our "best" classifiers, we will use all the data available to train them and they will be ready for testing.

4.2.2 Large training set

The Table 2 shows the classification error of each classifier mentioned before, evaluated by cross validation. The training set size is 10*1000 and the number of folds is 50. The standard deviation was not considered because it decreases with the increasing of data set. And it's time consuming to get the standard deviation of such a large training set.
We can see that the kNN classifier with k equals to 1 gives the best result, so we decided to use 1NN classifier at last in scenario 1.

	Classification error
loglc	6.0%
fisherc	12.5%
ldc	8.6%
nmc	17.4%
nmsc	15.2%
qdc	2.6%
parzenc	2.4%
knnc	2.0%
treec	30.3%

Table 2: Classification error for different classifiers

4.2.3 Small training set

From the previous figure, it is worth noticing that the performance of classifiers varies for large training data and small one. For a scenario in which we just have 10 objects per class for training, we should build our classifiers differently. Boosting is another effective way to

Individual Classifiers

As we discussed in section 3.2.2, the first 20 features extracted by PCA will be used in this scenario. The size of training set will be 10 objects per class. Due to the small size, it is not wise to split the training data into training set and validation set directly. In this scenario, cross-validation is a good choice to estimate the error made by classifiers. The Table 3 shows the classification error of each classifier mentioned before, evaluated by cross validation. The training set size is 10*100 and the number of folds is 10.

	Classification error	Standard deviation
loglc	19.80%	0.027
fisherc	18.80%	0.016
ldc	15.50%	0.024
nmc	25.54%	0.027
nmsc	20.10%	0.0152
qdc	60.60%	0.026
parzenc	15.40%	0.019
knnc	18.80%	0.024
treec	66.40%	0.039
vpc	26.60%	0.030
svc	23.00%	0.028

Table 3: Classification error for different classifiers(small training data)

Combined Classifiers

For this particular scenario, all of the classifiers perform worse comparing to the previous large training data, as shown in table 3. What's more, the variation for a single classifier indicates that the classifier is not stable. Combining different classifiers could lead to better performance and stability[5]. To combine different classifiers, we need to decide what are the base classifiers to combine and the combining rule. In order to ensure that base classifiers are different, ldc, parzenc and svc are chosen, each of which represents parametric, non-parametric and advanced classifiers. In consideration of combining rules, there are plenty of rules to use and generally rules are often heuristic[5]. To evaluate the performance of the combined classifiers using different combining rules, cross validations are applied. The number of folds is 10 and is repeated 10 times. Table 4 shows the result. Comparing to the previous result of individual classifiers, combined classifiers could improve the performance, but not dramatically, at least based on this experiment. More importantly, it does stabilize the performance as the standard deviation is dropping. In this simple experiment for combining, we just tested combining different classifiers in the same feature space. Future work could be done to test combining classifiers using different feature space.

Based on these experiments, for small training data scenario, we decide to com-

	Classification error	Standard deviation
min	16.5%	0.011
max	17.1%	0.019
median	14.9%	0.019
product	16.4%	0.016

Table 4: Performance of Combined Classifiers:ldc+parzenc+svc

bine linear discriminant classifer, Parzen density classifier and support vector machine using median rule.

5 Evaluation

To benchmark the classifiers we trained for both scenarios, the *nist_eval* function is used. The result is shown in Table 5. As we can see, both of the classifiers meet the requirements under certain scenario. The classification error for scenario 1 is 2.9% compared with 2.4% we got from cross-validation; while that for scenario 2 is 18.6% compared with 14.9% we got from cross-validation. The result is reasonable since the smaller the training set, the larger the variance. And both the results are fallen in the confident interval.

	Classification error
Scenario 1	2.9%
Scenario 2	18.6%

Table 5: Benchmark the classifiers for both scenarios

6 Live Test

6.1 Data preparation

In order to better understand the performance of the classifiers that best apply to the domain of digit recognition, a live test has been conducted. In this test, we will evaluate the best classifier for both scenarios which has been chosen before using our handwriting image. To do that, we have written down 10 digits for each class and then scanned it. The original scanned image is shown in Figure 11.

Afterward, we invert the color and slice the image into 100 pieces, so that each of them contains only one digit with the edge of digit touching the 4 sides of image. In order to match the digits of the NIST data set that is used for the training phase of our classifiers, a function is created which read the images above into prdatafile and binarized each digit image. An example of a digit is illustrated in Figure 13. For each scenario, the best classifiers found are trained and tested with the handwritten data set to measure the classification error.

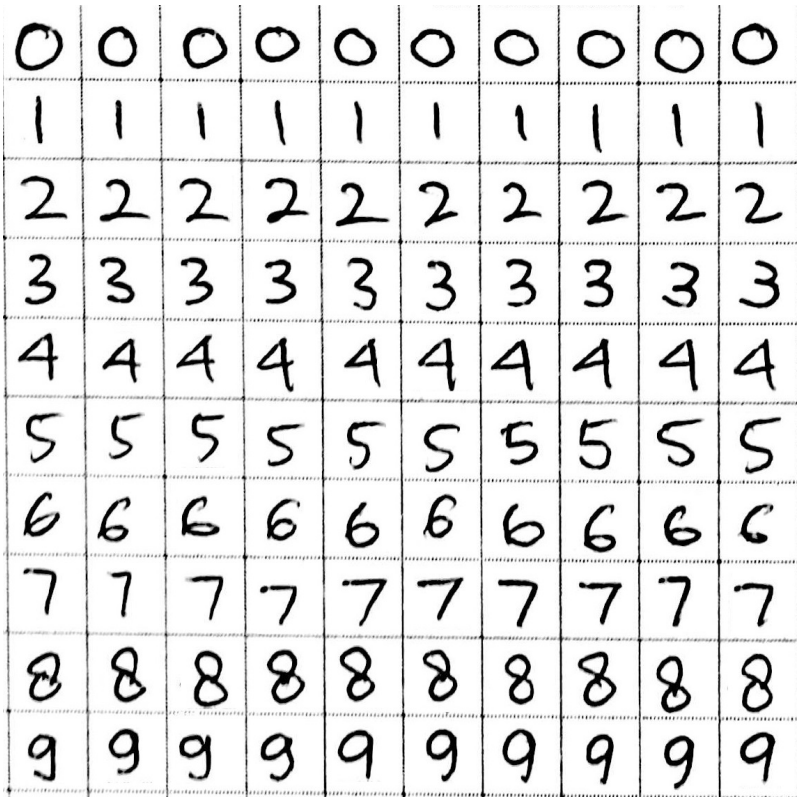


Figure 11: Handwritten digit for the live test

Figure 12: An example of digit image from the live test dataset

6.2 Result

In Section 4 and Section 5, we have chosen and evaluated the best classifier for both Scenario 1 and Scenario 2. For scenario 1, the best performance is obtained when using 1-NN, while the scenario 2, the median combination between Parzen, SVM, and Linear Bayes Normal Classifier appear to be the best. So we will test those classifiers using handwriting dataset which we created before. The results are shown in Table 6.

	Classification error
Scenario 1	3%
Scenario 2	18%

Table 6: Live test result for both scenarios

According to the result shown by Table 6, the amount of classification error in quite similar to the previous result. For the Scenario 1, in which 1-NN is used, the error is higher than both the test we have conducted before using the cross-validation method and the evaluation using nist_eval. If see into more details, the difference of error in this live test to both result is not significant as it is not higher than 1%. It is understandable as the amount of handwritten data which is used in the live test is lacking, which makes one misclassification would cost larger percentage compared to our previous test. As we only have 100 handwritten data, one error might cause 1% in classification error. If we have more handwritten data for the live test, the result would be more accurate, and closer to our previous result. For the Scenario 2, which

(a) in live dataset (b) in NIST dataset

Figure 13: Comparison of how the way majority of digit 4 is written

we combined Parzen, SVM, and Linear Bayes Normal Classifier, the performance, again, is not as good as our test using cross-validation method, but it is slightly better than the evaluation using nist_eval. Besides we only have limited data to do the test as we mentioned before, there is also another cause which may cause the increase of misclassification error, which is the way the digits were written. As we only collect the digits for the live test from one person, the way they were written is specific to that person and it might be different to the way other people write the digits. If you see at Figure 13, it is shown that the way this person write digit 4 is different with how the majority of the digit is represented in the NIST dataset. It is proven by the fact that 7 out of 10 digits of four is misclassified in this live test, which contributes to 7% to overall error. All in all, according to the result of test and evaluation that have been done, our digit recognition system has been proven to be able to fulfill the requirements in both scenarios.

7 Recommendation

7.1 Training time for different classifiers

The average training time (10*1000 data) of 10 repetitions for different classifiers is measured. The result is shown in Table 7. From the result, we can see that for linear classifiers, the stronger the assumption, the shorter the training time. And for the Parzen and kNN classifier, the training time is almost zero, since they just store the data and no need for training as explained before. In this case, there is no need to worry about the training time. But for a larger training set, the training time and the storage space must be considered.

	Training time (s)
loglc	1.74
fisherc	0.76
ldc	0.25
nmc	0.07
nmsc	0.10
qdc	0.09
parzenc	0.02
knnc	0.02
treec	1.3

Table 7: Training time for different classifiers

7.2 Learning curve: Will having more training data help?

It is intuitive to think that as we add more training data and more information is provided, it will help to increase the performance. For scenario 1, actually we have the flexibility to choose from 200 to 1000 training data per class. Next we will calculate so called "learning curve" to reveal the relationship between classification error and number of points in the training data set. The figure 14 shows learning curves for two classifiers. One is 1-Nearest Neighbor classifier(1NN), which is our "best" choice for this scenario. The other, Quadratic Discriminant classifier(QDC), is a representative for parametric classifiers. Also, dotted lines represent for training errors. From the learning curve we can find that as training size grows, the error of 1NN drops very quickly at first then drops slowly while the error of QDC stays very high until the training size reaches about 250. This can be explained by the parametric nature of the classifier. Generally, parametric classifiers need more data to estimate all parameters. In conclusion, will having more training data help for this scenario? Yes, it will. For classifiers like QDC, it will help a lot. But for our "best" classifier 1NN, it will help a lot at first, but it helps just a little as the size continues to grow. It's also worth to mention that the upper bound for every classifiers is the Bayes error, hence having more training data can never lead to a decrease of it.

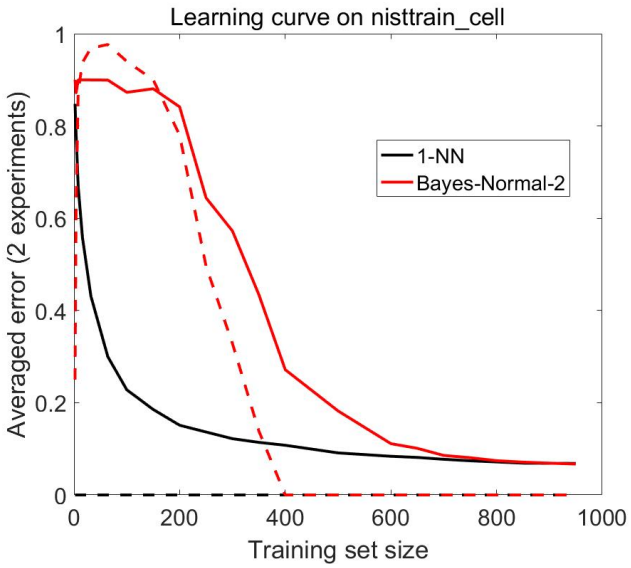


Figure 14: Learning curve for two classifiers in scenario 1, dotted lines represent for training errors.

7.3 Optimal feature size

In Section 2, we have chosen the pixel size to be 32x32 based on the visual of the digit, which makes us having 1024 features for each image. However, this feature size is not guaranteed to be the optimal choices. So, we did some trial to find out the minimum pixel size which still satisfies the requirements of both scenario 1 and 2. As we apply PCA for feature extraction in our system, using less feature size will make the computation for searching the eigenvector become faster. Based on our experiment, we found that the 16x16 pixel size will produce less feature, but still perform well enough to satisfy the requirements. So, we recommend the client to use this feature size in order to get faster runtime. The evaluation result using nist_eval for 1x1 feature size is provided in Table 8.

	Classification error
Scenario 1	4.2%
Scenario 2	24.8%

Table 8: Evaluation result (nist_eval) for 16x16 pixel size

8 Conclusion

We have proved that our system is able to perform well to satisfy the requirements for both scenario 1 and scenario 2. For both scenarios, we decided to classify the handwritten digit using pixel representation with every digit image is represented by 32x32 pixels (1024 features). PCA is used to extract the features for both scenarios. We employed PCA to extract 30 features in the scenario 1 and 20 features in the scenario 2. In the scenario 1, we found that 1-NN will perform the best with the 30 extracted features from PCA. While in the scenario 2, we found that the median combination of Parzen, SVM, and Linear Bayes Normal Classifier outperform the

other classifier. We have tested and evaluated the performance of both classifiers for each scenario they appear to be the best using cross-validation, nist_eval function, and also actual handwriting (live test), which all of them satisfy the requirements for each scenario (<5% for scenario 1 and <25% for scenario 2).

We also have three recommendations for the client who want to implement our digit recognition system. The first one is regarding the training time for classifier. We recommend the client to make a balance between training time and storage space.

Our second suggestion is to use choose the size of training data according to the classifier used. If 1 Nearest Neighbor classifier,as we recommend for scenario 1, is implemented, then it will not help much. However, if classifiers with many parameters to estimate are chosen, then increasing training size is a good choice.

Finally, we suggest client to use 16x16 pixel size for the minimum feature size which still satisfy the requirements for booth scenario, but will lead to faster feature extraction run time.

References

[1] 37steps.com. Features need statistics as they reduce, 2012.

[2] 37steps.com. Pixel representation, 2012.

[3] 37steps.com. Representation and generalization, 2012.

[4] Robert PW Duin and Elżbieta Pękalska. The dissimilarity space: Bridging structural and statistical pattern recognition. *Pattern Recognition Letters*, 33(7):826–832, 2012.

[5] Josef Kittler, Mohamad Hatef, Robert PW Duin, and Jiri Matas. On combining classifiers. *IEEE transactions on pattern analysis and machine intelligence*, 20(3):226–239, 1998.

[6] Cheng-Lin Liu, Kazuki Nakashima, Hiroshi Sako, and Hiromichi Fujisawa. Handwritten digit recognition: investigation of normalization and feature extraction techniques. *Pattern Recognition*, 37(2):265–279, 2004.

[7] Back-Propagation Network. Handwritten digit recognition with. 1989.

[8] Réjean Plamondon and Sargur N Srihari. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):63–84, 2000.

[9] Gerard P Van Galen. Handwriting: Issues for a psychomotor theory. *Human movement science*, 10(2):165–191, 1991.