THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF MATHEMATICS

On PCA and robust PCA for handwritten digit classification

DO Van Thuat (SID: 20405634)
LO Yi-Su (SID: 20399988)

Mar 13, 2017

(Updated on April 3, 2017)

CONTRIBUTION OF WORK

We had many discussions to determine the problem for our work and the appropriate method to solve it.

- DO Van Thuat: responsible for writing report.
- LO Yi-Su: responsible for experiments.

Our problem: How do PCA and RPCA methods help to classify handwritten datasets?

1 INTRODUCTION

This section provides a brief on principal component analysis (PCA) and robust principal component analysis (robust PCA/ RPCA). Section 2 and 3 show our experiments and results. There we apply PCA and RPCA methods to recognize handwritten digit datasets.

Principal component analysis (PCA) was introduced by Pearson (1901) and Hotelling (1933) [lect]. The idea is using a plane (or hyperplane) to separate two classes of points in an affine space. It is helpful for dimensionality reduction.

Let $X = [X_1|X_2|...|X_n] \in R^{p\times n}$. We will find a $k-$dimensional affine space $\mu + U\beta$ in R^p to approximate these n samples, where $U = [u_1, ..., u_k]$ consists of $k-$columns of an orthogonal basis of the affine space. Then the optimization problem is:

$$\min_{\mu,\beta,U} I = \sum_{i=1}^{n} \|X_i - (\mu + U\beta_i)\|^2. \tag{1.1}$$

In Equation (1.1), $U \in R^{p\times k}, U^T U = I_p, \sum_{i=1}^{n} \beta_i = 0$.

Robust principal component analysis (robust PCA or RPCA) is introduced by Candes E.J. et al. [Candes]. It is a modification of the well-known PCA. RPCA is useful with noised observations. Robust PCA decomposes a corrupted observed matrix into a low-rank and sparse ones $M = L + S$, so the low-rank matrix L is recovered.

The classical PCA looks for a decomposition of the data matrix $X \in R^{p\times n}$ into

$$X = L + E \quad \text{by} \quad \min_{s.t.\quad \text{rank}(L)\le k} \|X - L\|,$$

where the norm is matrix or Frobenius norm, and E is the error matrix with a small norm. However, PCA doesn't work well if the outliers are sparse and far from concentration. This makes PCA sensitive with outliers. To resolve this problem, RPCA decomposes:

$$X = L + S \quad \text{by} \quad \min_{s.t.\quad \text{rank}(L)\le k} \|X - L\|_0,$$

where $\|A\| = \#\{A_{ij} \neq 0\}$, L is a low-rank matrix and S is a sparse matrix.

SVM classification. *Support Vector Machine* (SVM) is invented by Cortes C. et al. (1995). In the experiment, we use SVM classifier based on feature vectors computed by the (kernel) PCA. The idea of SVM is to define a hyperplane in a multi-dimensional space to classify two categories of objects. Originally, SVM is a binary linear classifier. However, it can be generalized for multi-classification (more than two categories).

The (canonical) separating hyperplane is determined by

$$\begin{cases} |W^T X + W_0| = 1 & \text{(scaling)}, \\ \min \|W\| & \text{(maximized margin)} \end{cases} \tag{1.2}$$

In some cases, ones may not have a linear kernel. Instead, non-linear classification works. Here, *radial basis function* (RBF) kernel is such a method. Let x and x' be two samples (so they are feature vectors). According to [Jean], the RBF kernel is defined as

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right), \tag{1.3}$$

where σ is a free parameter (sometimes, choosing $\sigma = 1$).

2 EXPERIMENTS

2.1 TARGET HANDWRITTEN DIGIT DATASET

Throughout this section, the handwritten digit dataset at

`http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/zip.digits/`

will be of interest. The dataset is a training set consisting of hundreds of observations/samples (see Table 2.1) for every single decimal digits 0, 1, 2, ..., and 9. Every sample is a soft-copy image scanned from handwritten envelopes and contains 16×16 pixels. Every pixel takes value as the greyscale from 0 to 1. As a matter of fact, we can consider every sample a data point in the space $[0,1]^{256}$.

Before getting into experiments, Figure 2.1 gives us an impression how the samples in this dataset look like. Indeed, every sample is quite different from the other samples presenting the same digit.

Digit	0	1	2	3	4	5	6	7	8	9	Total
# samples	1194	1005	731	658	652	556	664	645	542	644	7291

Table 2.1: The number of samples for every digit in the dataset.

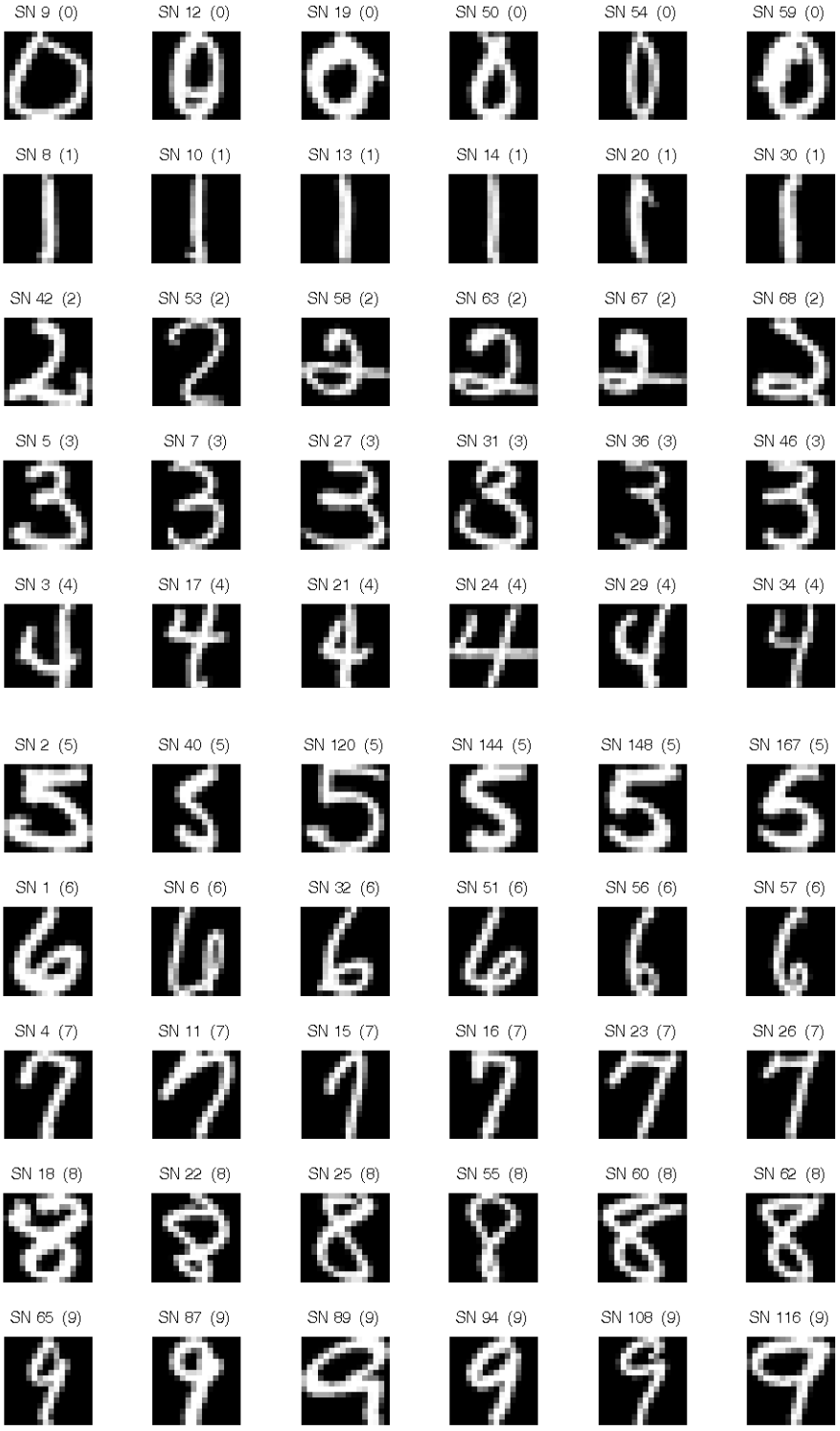

Figure 2.1: Some samples in the captioned dataset with their own sample number (SN) and label (i.e. the digit it presents.)

2.2 DIMENSION REDUCTION WITH PCA

As mentioned, the captioned dataset contains more than seven thousands 256-dimensional data points. In this case, an efficient dimension reduction approach, such as PCA, can play a key role. In this section, we conduct PCA over the training set, and investigate how the number of principal components influences the image quality if we want to recover the original handwritten image with data of much less dimension. The PCA is implemented with our own code, but the results are validated by `Matlab` built-in function `pca`.

As the result of applying PCA over the dataset, the explained variance ratio of the top 16 principal components are shown in Figure 2.2. We can see the top 8 principal components explain over 50% of variance, while the top 16 principal components explain around 70%. To see how lower-dimension data could be effective alternative, we adopt a certain number of principal components (i.e. U in (1.1)) and the projections of every sample onto them (i.e. β_i in (1.1)), combined with the sample mean (i.e. μ in (1.1)), to recovery the sample. Some results are presented in Figure 2.3.

In Figure 2.2 and 2.3, we can observe that the use of more principal components would benefit the recovery quality. However, for recognizing digits, a limited number of principal components might be enough for most samples. It indicates that by selecting a certain number of key principal components, we may be able to reduce the dimension of the dataset without losing too much feature of it.

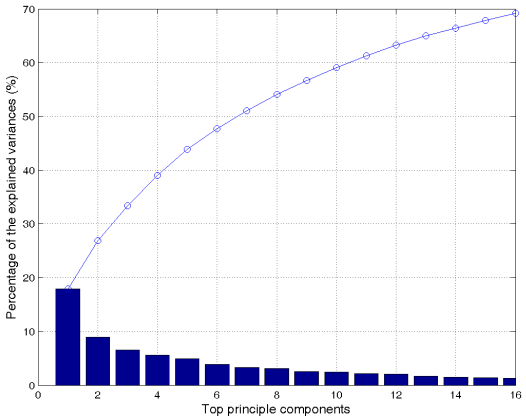

Figure 2.2: The explained variance of the top 16 principal components of the dataset.

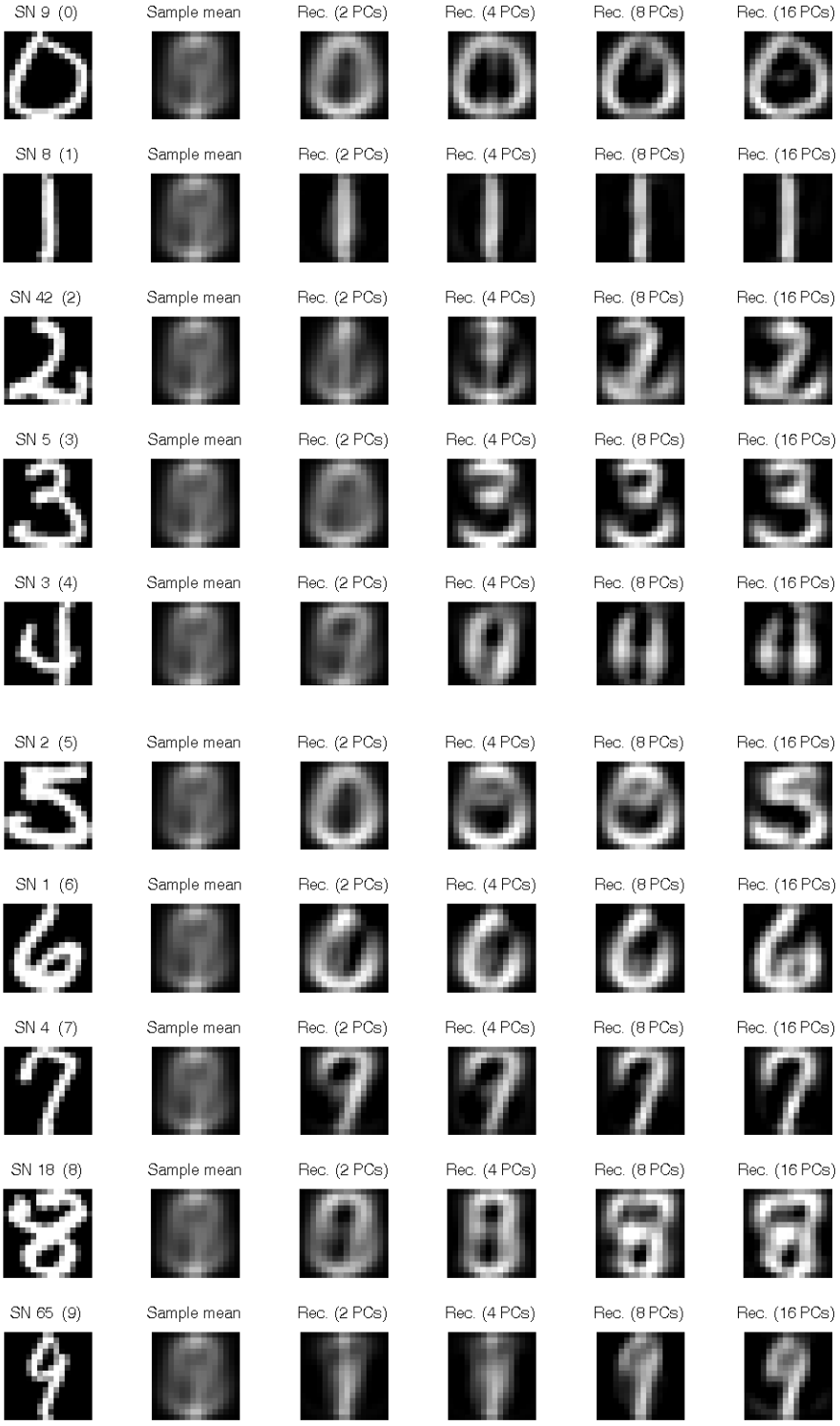

Figure 2.3: The recovery (Rec.) of some samples with only a few top principal components (PC).

2.3 Handwritten digit classifier training with SVM

As a follow-up application and further investigation to PCA, we try to train handwritten digit classifiers for the captioned dataset. In the following experiments, we first carry out PCA over the target dataset and use the projections of every sample onto several top principal components as a new data point to represent the sample. Then we adopt SVM with linear kernel (1.2) or RBF kernel (1.3) to train two-categories classifiers for digits. In implementation, we employ the `Matlab` built-in functions `svmtrain` and `svmclassify` for training a classifier with SVM and applying the classifier, respectively.

Experiment 1: Classifier for binary digits

We start with a simple case that involves only digits 0 and 1 in the captioned handwritten dataset. There are, as Table 2.1 presents, totally 2,199 samples of digit 0 and 1.

By applying PCA, we can see from the Figure 2.4 that the first two principal components already explain more than 60% of the variance. If we plot the projections of all data points onto these two principal components, as the left hand side of Figure 2.5 shows, we can see two separated groups. Moreover, with the help of SVM with linear kernel, we can easily find a hyperplane (which is a straight line in this case) as a classifier of the two digits, see the right hand side of Figure 2.5. There are totally 2 misclassified samples, on the right hand side of the straight line, leading to an error rate $\frac{2}{2199} \approx 0.09\%$ in this classifier training.

In the later subsection, we will show some misclassified samples as a closer observation.

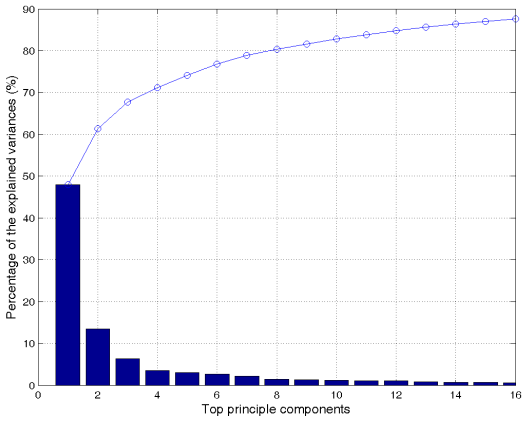

Figure 2.4: The explained variance of the top 16 principal components in the 0&1 subset.

Experiment 2: Classifier for one digit and the rest

Now we consider a slightly more challenging task in which we attempt to separate the digit 0 from the other digits, i.e. $1, 2, \ldots, 9$. That is, we have two categories/classes in this task: digit 0 and the rest. Similar to the previous subsection, we first conduct PCA over the whole dataset

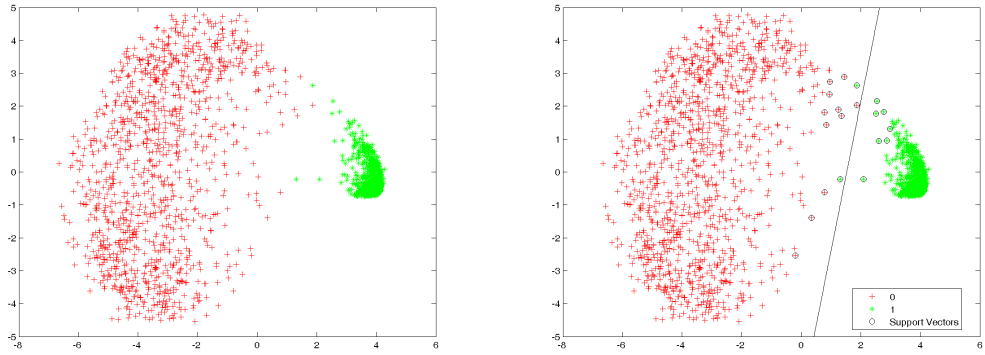

Figure 2.5: The projections onto the top two principal components in the 0&1 subset (left) and the result of SVM with error rate 0.09% (right). (Green dots: samples of digit 0; Red dots: samples of digit 1)

and choose the first two principal components as the coordinates. Then as we can see in Figure 2.6, the two groups of digit 0 and the other digits are no longer far away from each other; instead, they intersect much. In this case, the linear SVM suffers and results in an error rate 11.99% in the training. In comparison, SVM with RBF kernel function improves the error rate by 1%, see Figure 2.7.

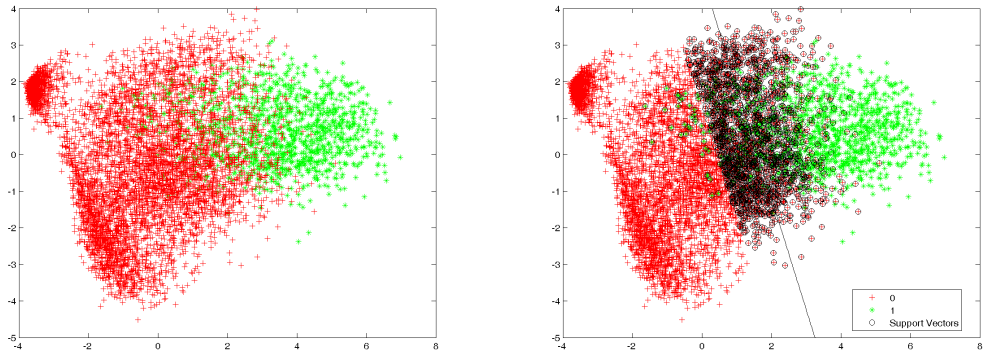

Figure 2.6: The projections onto the top two principal components of the whole set (left) and the result of SVM with error rate 11.99% (right). (Green dots: samples of digit 0; Red dots: samples of the other digits)

To illustrate such a failure, we can look back at Figure 2.2. Among the principal components concerning the whole dataset, the top two components explain less than 30% of the total variance. It indicates that the explanation ability of only two components might be too weak to facilitate a proper classifier. Thus, we increase the number of principal components engaging in the SVM (with RBF kernel) and then obtain an immediate improvement in the error rate, see Table 2.2. Note that the use of 16 principal components is already enough for finding a

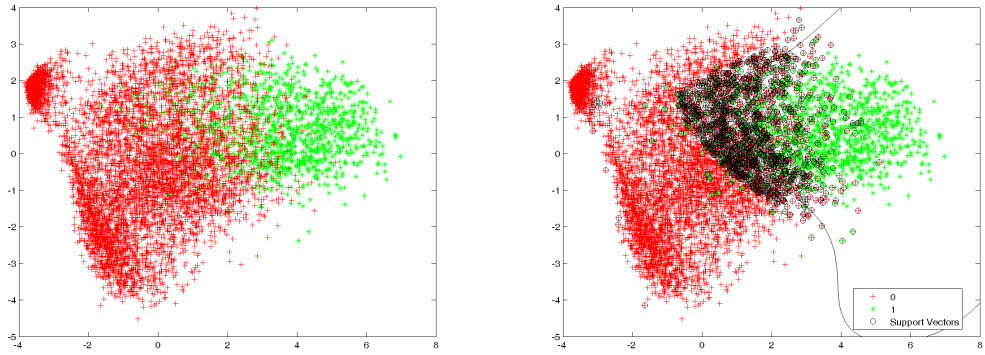

Figure 2.7: The projections onto the top two principal components of the whole set (left) and the result of SVM with error rate 10.92% (right). (Green dots: samples of digit 0; Red dots: samples of the other digits)

perfect classifier.

# Principal components	2	4	8	16
Training error (%)	10.92	3.84	0.62	0

Table 2.2: The error rates under the use of different numbers of principal components in the training of a classifier for 0 and the other digits.

EXPERIMENT 3: CLASSIFIER FOR ALL DIGITS AND THE CLASSIFICATION IN THE TEST DATASET

In this subsection, we try to find a classifier for all digits $0, 1, \cdots, 9$. Since SVM is a two-category classifier training approach, we adopt a strategy called *one versus the rest*, which is essentially a repetition of the last experiment. That is, we first find a hyperplane for digit 0 and the other digits, and then the hyperplane for digit 1 and the other digits, and so on. In the subspace spanned by the top 16 principal components in Figure 2.2, we carry out the strategy and obtain a set of hyperplanes for separating every digit and the rest. By applying these hyperplanes to classify the training set, the resulting training error rate is shown in the first row of Table 2.3. We also summarize the results of every one-versus-the-rest classification and show the error rate in the last column. It is worth mentioning that when we summarize the results, there are three types of misclassification:

1. samples misclassified as the other digits,
2. samples classified as more than one digit,
3. samples not classified throughout all one-versus-the-rest classification.

For the second case, we further classify those samples into the first digit they were classified as.
Next on, for evaluating the classifier, we introduce a test handwritten digit dataset at

`http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/zip.test.gz`

This dataset is actually associated with the captioned training dataset. Similar to what we did over the training set, we first compute the differences of every sample in this test set and the sample mean μ of the training set, then the projections of them onto the 16 chosen principal components. Finally, the trained classifier is applied. The error rates of every one-versus-the-rest classification, as well as the summary, are shown in the second row of Table 2.3.

Digit	0	1	2	3	4	5	6	7	8	9	Summary
Training error (%)	0.00	0.04	0.00	0.00	0.04	0.00	0.00	0.03	0.00	0.05	0.1
Test error (%)	4.33	0.80	4.83	4.53	4.19	5.78	2.09	1.44	4.38	1.94	33.98

Table 2.3: The error rates in the training and test of the classifier for handwritten digit when there are 16 principal components in consideration.

As Table 2.3 presents, the error rates in training are excellent, while the error rates in test are unacceptable. In fact, most of the error samples in the test set are the third type of misclassification listed above (so the error rate in summary is near to the sum of error rates for every digit.) It may suggest that the hyperplanes of this classifier are *overfitting*. In order to resolve this problem, we further employ the technique of *cross validation* in the classifier training. In particular, we first produce 5 folds and in every fold we hold out 20% samples of the training set as the *validation set*. Then we insert an additional step to tune parameters before every one-versus-the-rest digit classifier training, including the σ in (1.3) and the box constraint in the problem of finding the optimal hyperplane. As a result, Table 2.4 present the error rates of the new classifier trained with the participation of cross validation scheme. As one can observe, this run receive a slightly worse error rate in training, but a much better accuracy in test.

Digit	0	1	2	3	4	5	6	7	8	9	Total
Training error (%)	0.03	0.11	0.25	0.22	0.11	0.21	0.05	0.10	0.26	0.12	0.84
Test error (%)	0.80	0.70	1.69	1.25	1.59	1.79	1.10	0.70	1.69	1.05	7.32

Table 2.4: The error rates in the training and test of the classifier for handwritten digit when there are 16 principal components in consideration.

Moreover, we collect some error samples in the test set which are counted in the Table 2.4, see Figure 2.8. It may give us a visual clue for why those samples are misclassified.

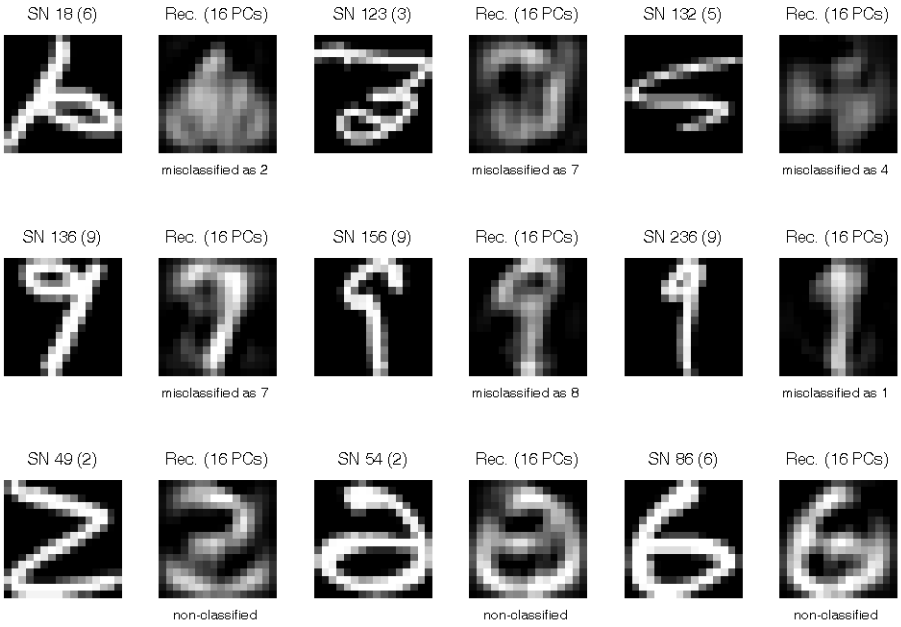

Figure 2.8: Some error samples in the classification of the test set which are counted in Table 2.4. The i-th row is of the i-th type of misclassification.

2.4 DENOISING WITH RPCA

In this section, we assume the samples in the captioned test dataset in section 2.3 is contaminated from the source or in the process of scanning, and thus lead to dot noise on every sample. In particular, we randomly set up to 20% of the pixel values of every sample to be 1, regardless what the original value is. Some samples got contaminated in this way are exhibited in Figure 2.9.

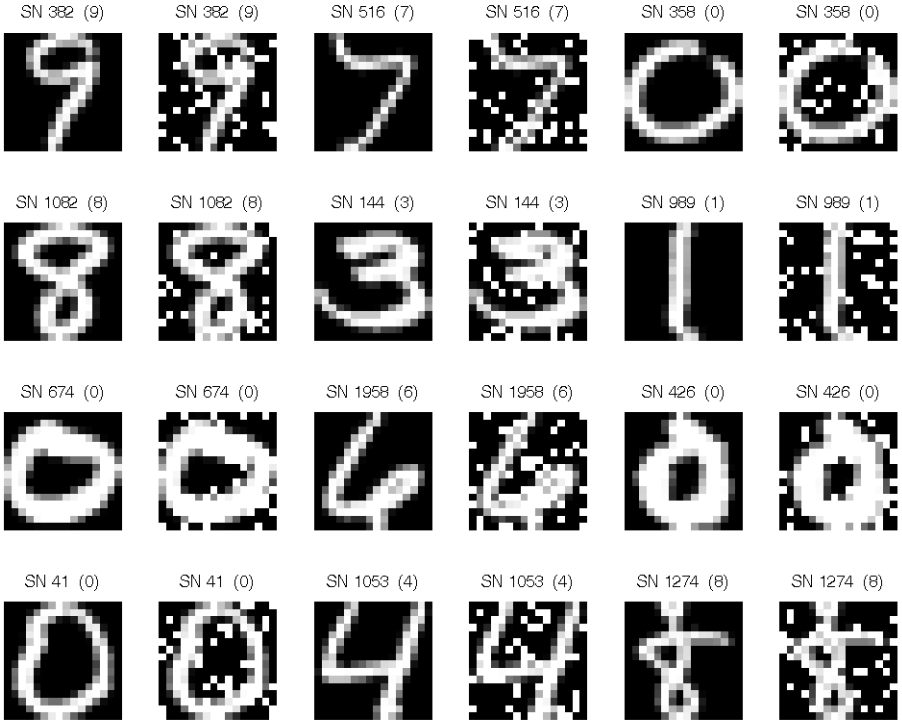

Figure 2.9: Some examples for the noise. (In each pair, the left hand side: the original sample; the right hand side: the sample contaminated by noise.)

It is not surprising the noise become a barrier for classification. In our experiment, the classifier for all digits trained in section 2.3 suffers from the noisy data and result in a higher error rate in the classification of the test set, see Table 2.5.

Digit	0	1	2	3	4	5	6	7	8	9	Total
Test error (%)	1.44	3.14	3.84	2.29	3.34	2.99	1.49	3.64	6.48	3.54	21.28

Table 2.5: The error rates in the classification of test set when there are 16 principal components in consideration.

Since the noise seems to be a sparse structure, we try to apply RPCA as a denoising procedure before the classification. We employ the code at

http://perception.csl.illinois.edu/matrix-rank/sample_code.html

which is written by Z. Lin, M. Chen, L. Wu, and Y. Ma for implementing RPCA with a core solver using inexact augmented Lagrange multiplier method. Some outcome low-rank part and sparse part of noisy samples are placed in Figure 2.10. Overall, it shows that RPCA successfully separate the noise and leave a low-rank part which keep most ingredient of the original sample.

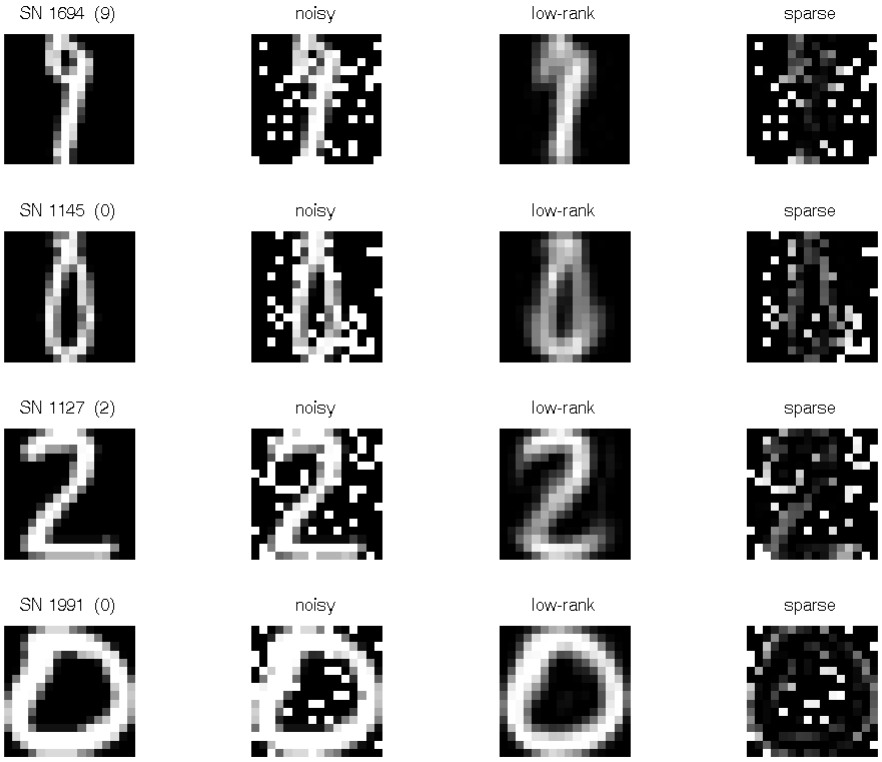

Figure 2.10: The results of RPCA. (1st column: original samples; 2nd column: the samples with noise; 3rd and 4th column: the low-rank and spare parts obtained through RPCA.)

Finally, we apply the classifier again for the classification of the low rank part of the test set. The resulting error rate is presented in Table 2.6. In the comparison with Table 2.4 and Table 2.5, we can see that the the application of RPCA make a significant improvement, but the error rates are still higher than the classification of the clean dataset.

Digit	0	1	2	3	4	5	6	7	8	9	Total
Test error (%)	1.05	0.80	2.14	2.49	3.04	2.54	1.15	1.10	3.09	2.69	10.66

Table 2.6: The error rates in the classification of test set when there are 16 principal components in consideration.

3 CONCLUSION

In this report, we investigate the usage of PCA and our results show that PCA is an effective tool for feature searching and dimension reduction. As an application of PCA, we also try to perform SVM to train classifiers for handwritten digits and conclude that a certain rate of successful classification can be achieved with only a small number of principal components. In addition, we also show that RPCA is good at separating a sparse part and then is helpful for denoising the data.

APPENDIX

The associated `Matlab` code and raw `tex` file of this report are available at

`https://drive.google.com/open?id=0B3zEmYEa2Wc5R3FCLTg4Y3hvXzg`

REFERENCES

List of references

[Candes] Candes E. J., Li X., Ma Y., and Wright (2011). Robust principal component analysis? J. ACM 58 , 3, Article 11.

[Cortes] Cortes C., Vapnik V. (1995). Support-vector networks. Machine Learning. 20 (3), 273-297. doi:10.1007/BF00994018.

[Jean] Jean-Philippe Vert, Koji Tsuda, and Bernhard SchÃűlkopf (2004). A primer on kernel methods. Kernel Methods in Computational Biology.

[lect] Yuan Yao, A Mathematical Introduction to Data Science, Lecture note.