Mini-Project 2 - MATH6380J
SOME CLASSIFICATION METHODS ON HANDWRITING DIGITS AND DRUG SENSITIVITY SAMPLES

DO Van Thuat (SID: 20405634), LO Yi-Su (SID: 20399988)
Department of Mathematics, School of Science
The Hong Kong University of Science and Technology
tdovan@connect.ust.hk

Abstract

In this work, we deal with problems on image classification and drug sensitivity prediction. In Section 1, we attempt to train a model capable of classifying the handwritten digits. In the training set, we first carry out principal component analysis as an approach of dimension reduction. Then we train classifiers with several tree-based algorithms and test for accuracy. In Section 2, we try to classify real-valued responses to drug of cancer cell lines. It is interesting that testing error is rather small, although training accuracy is extremly bad (30-50%).

Contribution 50-50. *Each person contribute a half of the work, not only in experiments but also in report writing.*

1 Handwritten Digit Classification with Tree-based methods

In this section, we attempt to train a model capable of classifying the handwritten digits. In the training set, we first carry out principal component analysis as an approach of dimension reduction. Then we train classifiers with several tree-based algorithms, such as full and pruned decision tree, bagging, and random forest. Finally, we apply those classifiers into a test set and investigate the accuracy of classification.

The `Matlab` source code for experiments in this section are accessible at

`https://drive.google.com/open?id=0B3zEmYEa2Wc5R09CY3ZPRlFRRzQ`

1.1 An introduction to dataset

The data set of interest is a set of soft-copy images for handwritten digits $0, 1, \ldots, 9$ scanned from US envelopes. One can download them at

Training set:	`http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/zip.digits/`
Test set:	`http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/zip.test.gz`

Table 1 and Figure 1 below show some information and ingredient of the captioned data set. Note that every sample consist of 16×16 pixels which take value as the grey scale from 0 (black) to 1 (white). As a matter of fact, we can consider every sample a data point in the space $[\,0, 1\,]^{256}$.

1.2 Principal component analysis

Principal component analysis (PCA) was introduced by Pearson (1901) and Hotelling (1933) [lect]. It could be helpful for dimension reduction and feature extraction. As the result of applying PCA over

Digit	0	1	2	3	4	5	6	7	8	9	Total
Training set	1194	1005	731	658	652	556	664	645	542	644	7291
Test set	359	264	198	166	200	160	170	147	166	177	2007

Table 1: Number of samples for every digit in the training and test set.

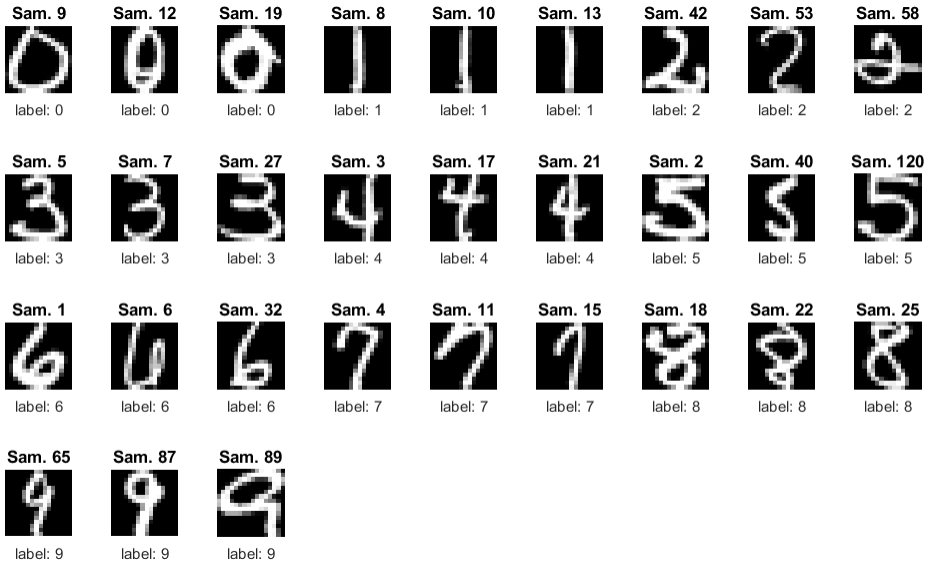

Figure 1: Some example samples in the training set.

the training set[1], the explained variance ratio of the top 16 principal components are shown in Figure 2. The top 8 principal components explain over 50% of variance, while the top 16 principal components explain around 70%. In the figure we also show some sample recoveries with lower-dimensional data along a few top principal components.

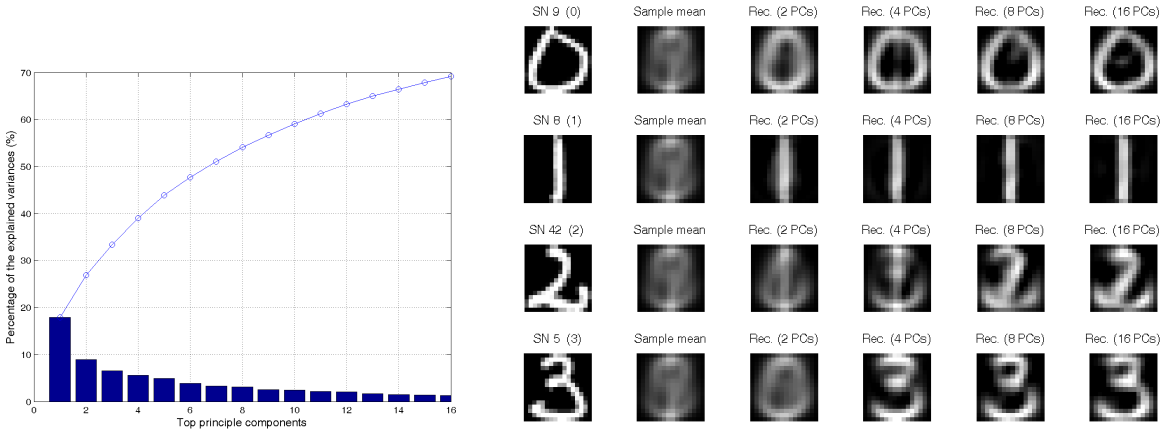

Figure 2: The explained variance of the top 16 principal components of the training set.

1.3 An illustration to tree-based methods

In order to understand the operation of tree-based algorithms, we first consider a simpler classifier training task which involves only digits 0, 1, and 2 in the training set. We extra the score/projection

[1]In implementation, we invoke the `Matlab` (2015b) buit-in function `pca`.

onto the top two features/principal components as the data point representing every sample. See Figure 3.

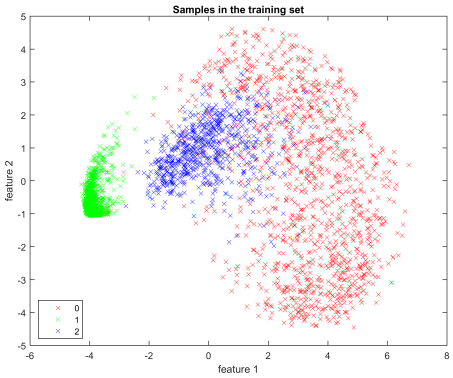

Figure 3: The data points as projection of samples onto the top two principal components.

As the fist algorithm, we grow the full decision tree by recursive binary splitting[2]. The resulting tree and corresponding region splitting is presented in Figure 4. The error rate

$$\frac{\text{number of misclassified samples}}{\text{total number of samples}}$$

in the training set is not terrible, however, the region splitting could be overfitting.

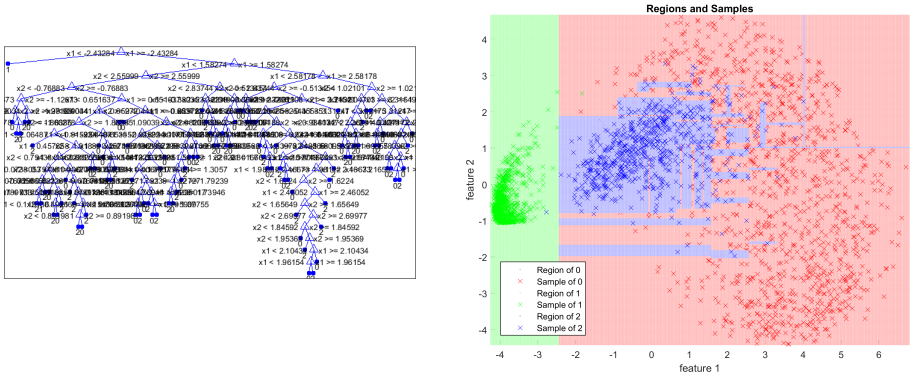

Figure 4: The full tree with 197 nodes (left) and the split regions (right) with error rate 2.22% and 8.54% in the training and test set, respectively.

Next, we a 5-fold cross validation, we prune the full decision tree and obtain an optimal subtree as shown in Figure 5[3]. Note that the classification based on the subtree lead to less regions and more errors in the training set, but enjoys a higher accuracy in the test set, in a comparison with the full tree. It shows an improvement for the overfitting.

Furthermore, we perform the bagging method in which 100 trees are grown over different bags sampled from the training set with replacement. Then we have them 'vote' over every sample for classification[4]. The result is exhibited in Figure 6. It is worth mentioning that the OOB error is kept in low after growing about 10 trees, which make us confident that the number of total trees, 100, are sufficient.

[2]Here we employ `fitctree` and `loss` in `Matlab` (2015b)

[3]`Matlab` (2015b) function: `fitctree`, `cvloss`, `prune`, `loss`.

[4]`Matlab` (2015b) function: `TreeBagger`, `predict`.

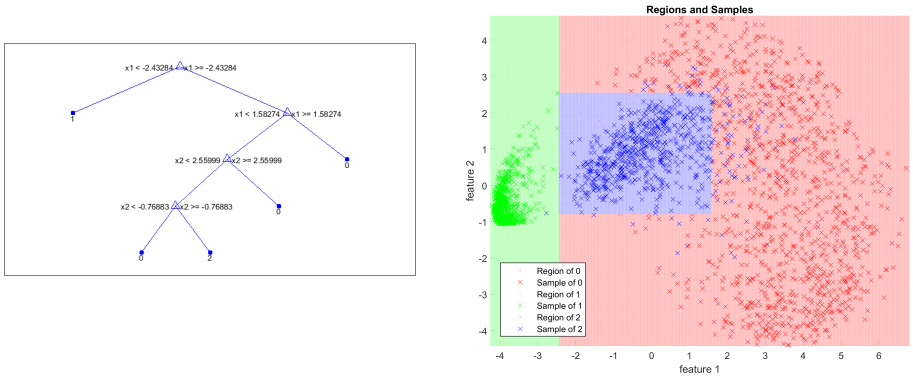

Figure 5: The sub tree with 9 nodes (left) and the split regions (right) with error rate 4.81% and 7.64% in the training and test set, respectively.

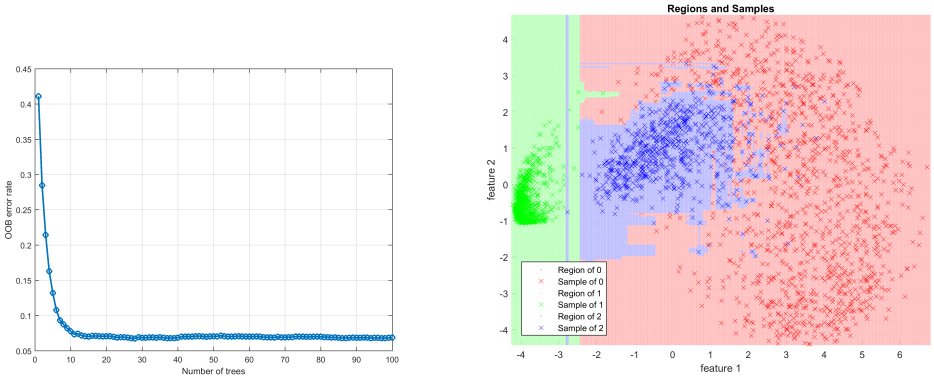

Figure 6: The OOB error (left) and split regions (right) resulted from bagging method with error rate 0.00% and 2.32% in the training and test set, respectively.

Finally, with the typical setting $m = \sqrt{p}$, we conduct random forest algorithm[5]. It is not surprising the result is similar to the bagging method. See Figure 7.

1.4 Classification of the handwritten dataset

With a better understanding to tree-based algorithms, we go back and carry out the same procedures over the target problem, the classification of all digits $0, 1, \ldots, 9$. Unfortunately, as one can see in Table 2, two features are no longer enough for such a challenging task. However, with the increasing in the number of features, we still obtain satisfying accuracy 97.67% (i.e. error rate 2.33%). Moreover, we also collect in the last experiment some misclassified samples in the test set and their recovery with 16 principal components, which may give us a visual clue for why those samples end up errors in the classification. See Figure 8.

[5] Matlab (2015b) function: TreeBagger, predict.

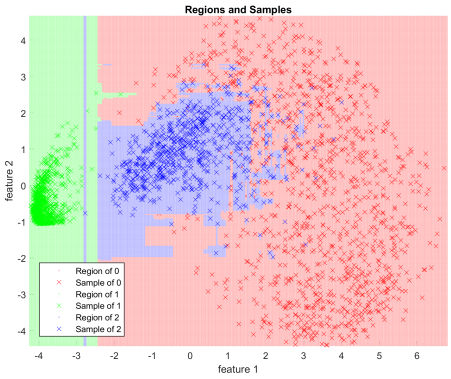

Figure 7: The split regions resulted from random forest with error rate 0.00% and 2.32% in the training and test set, respectively.

# features in use		2	4	8	16
Full tree	Training	22.34%	10.60%	5.71%	4.66%
	Test	50.36%	30.96%	21.50%	19.71%
Pruned tree	Training	42.37%	22.49%	11.73%	9.81%
	Test	46.87%	27.59%	21.38%	20.19%
Bagging	Training	0.00%	0.00%	0.0%	0.00%
	Test	13.48%	6.79%	4.28%	3.06%
Random forest	Training	0.00%	0.00%	0.0%	0.00%
	Test	13.48%	6.69%	3.98%	2.33%

Table 2: Error rates in the training and test set with the use of different number of features and methods.

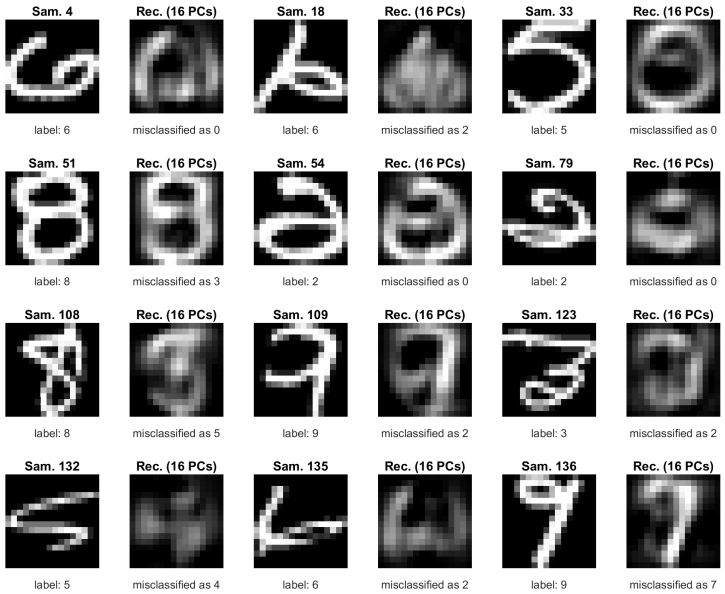

Figure 8: Some misclassified samples (in the left of every pair) and their recoveries (right) in the test set. The bad handwriting or unclearness of the recovery may make them difficult to be recognized.

2 Some attempts on drug sensitivity

2.1 About datasets and strategy for classification

In this section, we work with two (sample) datasets on drug sensitivity of cancer cells:

- 1-drug responses of 542 cancer cell lines with 60 binary features (genes);
- 20-drugs responses of 100 cancer cell lines with 20 discrete features (drug dosages).

Based on learning from given samples, we target to predicts the responses (IC50 outputs) of the test samples.

The datasets are poor, e.g. having few (hundreds) samples. On the other hand, their inputs is likely binary (discrete: 0, 1, 2, 3) while their outputs are continuous. Each sample of inputs has a distinguished (real-valued) output from the rest. How can one classify the outputs of a dataset (of 100 samples, for instance)?

We observe that the output range of a dataset is normally a bounded interval on the real line. Distribution of sample-outputs (density) is very abnormal. It is very concentrated on a certain subset of the range, but is also sparse on other intervals. Therefore, if we divide the range into a uniform partition, then some parts will possess a lot of observations while others do few ones.

The difficulties when dealing with datasets of drug sensitivity are:

- Inputs is discrete while outputs are continuous;
- It is really hard to predict continuous outputs (fitting);
- Output distribution is abnormal;
- Classifying hundreds of distinct outputs.

Our idea to overcome these difficulties is replacing fitting problem by classification. Of course, classifiers do not give us real-values as wanted, but they are useful in the sense that they can tell us where the outputs lie in. This will become clearly in our experiment. Our strategy is dividing the range of the responses into subintervals such that output distribution is as much equal as possible. Note that, in general, lengths of the subintervals after partition are very different. All sample-outputs belonging to a subinterval form a class.

In experiment, we must withdraw a testing set (15%) from the datasets randomly. Then we apply decision (bagged and RUSBoosted) trees to train classifiers. Since the both testing and training sets are small, each time of withdrawing will have a remarkable impact on error rates. It is interesting that in spite of terribly bad training result (50%), the testing accuracy is rather good (70%). This strange behavior is our main question with no clue to answer.

2.2 On 1-drug samples

The dataset of response for one drug has 542 known-value samples, dividing into 461-training and 81-testing subsets. We apply **Bagged Trees** for this samples to train the classifier with 5-folds cross validation. The training fails with accuracy 55% (no PCA) and 49% (with PCA). Since the testing set is taken out randomly, it will affect the result. Therefore, we run the program 5 times to see variance.

The output range is divided into 5 classes as followed.

Class	[-4.1, -2)	[-2, -0.3)	[-0.3, -0.3]	(0.3, 2.5]	(2.5, 5.1]
Counts	22	38	19	162	301
mean	-2.9	-1.1	1e-3	1.6	3.4

The following table shows the accuracy change.

Run	1st	2nd	3rd	4th	5th	6th	7th
no PCA	0.53	0.63	0.64	0.68	0.68	0.70	0.52
PCA	0.56	0.68	0.70	0.74	0.75	0.79	0.69

2.3 On 20-drugs samples

The dataset of response for 20-drugs has 100 known-value samples, dividing into 85-training and 15-testing subsets. We apply **Bagged and RUSBoosted Trees** for this samples to train the classifiers with 5-folds cross validation. The trainings fails with accuracy 48% for bagging and 34% for RUUBoost. We run the program many times to see variance.

The output range is divided into 6 classes as followed.

Class	[-0.4, -0.1]	(-0.1, -0.02]	(-0.02, 0.02]	(0.02, 0.11)	[0.11, 0.2)	[0.2, 0.4]
counts	12	6	12	27	22	21
mean	-0.2	-0.06	3e-3	0.07	0.15	0.27

The following table shows the accuracy change.

Run (times)	7	4	6	3	4
Bagged	0.93	0.73	0.87	0.8	1
RUSBoosted	0.93	0.8	0.73	0.67	0.65

Now, the 20-drugs samples are divided into 80-training and 20-testing subsets. We still apply **Bagged and RUSBoosted Trees** for this samples to train the classifiers with 5-folds cross validation. The trainings fails with accuracy 34% for bagging and 33% for RUUBoost. We run the program 10 times to see variance.

Bagged	0.95(3)	0.9(2)	0.85(3)	0.8(2)
RUSBoosted	0.7(1)	0.75(1)	0.85(5)	0.8(3)

2.4 Conclusion and question

Although the accuracy rates in the experiments are fluctuating, they are much higher than training process. This is very strange and extraordinary. We do not have any idea to explain that behavior, so put into open question.