ECEN 765 Machine Learning with Networks

programming assignment 1

Jicheng Gong

October 14, 2017

1 Binary Images

Solution:
In order to manipulate images in Python, we need to have package PILLOW. I created a new image and then set its pixel value to 1 if the corresponding position in the digit file is 1. Figure 1 shows the original txt file and the binary image created by Python.

```
00000000000000000111100000000000
00000000000000001111111100000000
00000000000000011111111100000000
00000000000000011111111110000000
00000000000000011111111110000000
00000000000000111111111100000000
00000000000000111111111100000000
00000000000001111111111100000000
00000000000001111111111100000000
00000000000001111111111100000000
00000000000001111111111000000000
00000000000011111111111000000000
00000000000011111111111000000000
00000000001111111111111000000000
00000001111111111111111000000000
00000011111111111111110000000000
00000111111111111111110000000000
00000111111111111111110000000000
00000011111111111111110000000000
00000011111110111111110000000000
00000001111100111111110000000000
00000000000000011111110000000000
00000000000000011111100000000000
00000000000000111111110000000000
00000000000000111111110000000000
00000000000000111111110000000000
00000000000000111111110000000000
00000000000000011111110000000000
00000000000000011111110000000000
00000000000000001111111000000000
00000000000000001111110000000000
```

Figure 1: The original txt file

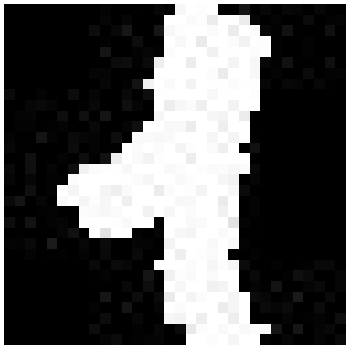

Figure 2: The binary image

2 Implement Naive Bayes Classifiers

2.1 Model selection

Here I used four different algorithms for Naive Bayes Classifiers.
I first wrote a multinomial Naive Bayes Classifier program by some reference from https://github.com/wepe/MachineLearning/blob/master/NaiveBayes/NaiveBayes.py.
The program basically has two parts, one for training and the other for prediction. In training part, the probability of prior P(y) and conditional probability $P(x_i|y_k)$ first need to be calculated. In order to handle some 0 occurrence situation, we add a Laplacian smoothing here.

$$P(y_k) = \frac{N_{y_k} + \alpha}{N + k\alpha}$$

Where N is the total number of samples, k is the number of the number of categories, N_{y_k} is the number of samples whose category is y_k, and α is the smoothing value.

$$P(x_i|y_k) = \frac{N_{y_k,x_i} + \alpha}{N_{y_k} + n\alpha}$$

Where N_{y_k} is the number of samples whose category is y_k, n is the number of dimension, N_{y_k,x_i} is the number of samples whose i dimension is equal to x_i.

As to the classification part, the maximum product of prior and conditional probability is the classification result.

$$f(x) = argmax P(y_k) \prod_{i=1}^{n} P(x_i|y_k)$$

On the other hand, from the sklearn package we can directly use the Naive Bayes Classifiers. However, notice that there are three basic models can be chosen, MultinomialNB, GaussianNB and BernoulliNB. Considering that the given data is already binary data, there won't be difference between using MultinomialNB and BernoulliNB and it is obviously inappropriate to use GaussianNB. But still, tests have been done for all the three models to verify whether this is the case.

2.2 Tests

Table 1: test errors

Method	NaiveBayes	MultinomialNB	GaussianNB	BernoulliNB
Training Error rate	0.0517	0.0724	0.208	0.0631
Test Error rate	0.124	0.0762	0.266	0.0688

From the table we can see that MultinomialNB and BernoulliNB has close results in both training and test error rate. And BernoulliNB perform slightly better than MultinomialNB. GaussianNB is definitely not suitable for this problem. Whereas the self written NBC has a good accuracy in training error while poor accuracy in test error. Right now I can't find the reason for that.

3 Implement K-Nearest-Neighbors

3.1 KNN

The principle of KNN algorithm is relatively easy. It calculate Euclidean distance between the input vector and each other vectors in the training set. If k is 1, then the testing vector has the same category as the closest one, else the testing vector will choose the category from majority.

3.2 Test for training and testing data

Table 2: KNN test

k	1	2	3	4	5	6	7	8	9	10
Training Error rate	0	0	0.0119	0.0103	0.016	0.0129	0.0191	0.017	0.0232	0.0212
Test Error rate	0.0137	0.0137	0.0105	0.0116	0.0179	0.0179	0.0221	0.019	0.0221	0.0201

While using the testing set, we can see that when k equals to 3 it has the lowest error rate. And with the growing number of K, the error rate also goes up. As for the training set, the error rate for 1 and 2 is 0, this is because it will always have one candidate that has 0 distance. When k is larger than 2, there might be two other vectors from a wrong category, then the prediction will be wrong. When the K grows larger, it will be slightly more accurate than testing set. However, when K is very large, about 9 or 10, it will basically be the same as testing error rate.
Generally,The best choice of k depends upon the data; generally, larger values of k reduce the effect of noise on the classification, but make boundaries between classes less distinct.

3.3 Comparison between KNN and NBC

Firstly, the time consumption for two algorithms are different. For NBC (not counting its training time) it takes 35s to do the classification, and takes 30s to do the training. For KNN, it takes 1min 28s to do the classification. Thus we may say that NBC has higher classification efficiency.
Reason why KNN is slow is because The kNN is a type of lazy learning where the function is only approximated locally and all computation is deferred until classification.

However, the test error rate for NBC is 0.0688 while KNN's average test error rate is 0.0169. NBC's error rate is 3 times higher than KNN's.
To improve Naive Bayes Classification, one possible solution will be in next section, which is using PCA to remove the redundant dimensions first. Since these correlated dimensions are against the Naive Bayes assumption that each attribute should be independent. The other solution may be using Adaboost to iteratively improve the weak learner.

4 Principle Component Analysis

4.1 NBC and KNN after PCA

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables. It can be done by eigenvalue decomposition of a data covariance (or correlation) matrix or singular value decomposition of a data matrix.

Here I combine the testing and training digits together to computer principle components. After computation, it will contain 200 dimensions that have totally 95% of the variation and 500 dimensions contain 99% of the variation. Then apply the PCA transform to both training and testing data.Since after PCA transformation, the data will have a Gaussian distribution, it is more appropriate to use the GaussianNB to do NBC than BernoulliNB. And table 3 also shows that GaussianNB has a better error rate. Also in the table, it shows that with more number of components included in PCA, the training error rate will decrease. However it is not the case for test error rate.

As to KNN, I just choose the special case k=3,5,10 to see how the number of components included will affect the result. From table 4 we can see that When there are 50

components included in the principle components, it will produce a lower error rate than the original KNN algorithm. Others are basically a little bit more accurate than the original KNN. But the trend remains the same, which is error rate grows when k grows larger.

Table 3: PCA-NBC

Method	GaussianNB	BernoulliNB	number of components	time
Training Error rate	0.0888	0.284	50	10s
Test Error rate	0.0496	0.133	50	10s
Training Error rate	0.0623	0.242	200	13s
Test Error rate	0.0528	0.146	200	13s
Training Error rate	0.0359	0.154	500	17s
Test Error rate	0.0676	0.23	500	17s

Table 4: PCA-KNN

Error type	k=3	k=5	k=10	number of components	time
Training Error rate	0.0088	0.0128	0.0165	50	3s
Test Error rate	0.0095	0.0126	0.0179	50	3s
Training Error rate	0.0134	0.0144	0.0181	200	20s
Test Error rate	0.0148	0.0158	0.0169	200	20s
Training Error rate	0.0115	0.0154	0.0194	500	35s
Test Error rate	0.0106	0.0169	0.0201	500	35s

4.2 Analysis

PCA helps reduce redundant dimensions. In the experiment we can find that even using only 50 dimensions out of 1024, we can still get a very accurate result. With a good choice of parameter model, we can even lower the error rate.
On the other hand, with the decreasing number of dimensions, it also helps improve the efficiency. It used to be very time consuming for KNN to compute, however, it improved 20 time with the help of PCA.
What's needed to pay attention is that we need to do data normalization first before perform a PCA on the testing and training data. In this case, the digits file has already been normalized, otherwise more efforts are needed to pay.

Also in this assignment, due to time limitation, I also have some questions remain unknown.

1. What's the connection between training error rate and testing error rate.
2. Why after PCA transformation, the new data has a Gaussian distribution.
3. I also tried to do PCA on testing and training data respectively, the prediction result was very wrong. I wonder why that would happen.