PA-3 Project

MNIST classification

ZHAI Guanxun 54345382 4/12/2015

This project is for hand written digits classification, I used three kinds of methods to do the
classification:
SVM (one-class, C-SVM), Logistic Regression, Naïve Bayesian Classifier.
In this project, I focus on the program designing to achieve some useful classifiers. And I used the library 'libsvm' and 'liblinear' to implement SVM and Logistic Regression. As for the Naïve Bayesian classifier, I write a customized program in matlab.

PA-3 Project

MNIST classification

Part I

Pre-process on data

This part is about how to change the digits picture data to make it become easier to implement the algorithm.

Step 1

Change Digits Pictures Into Pixel Grey Scale Value Matrix.

This part is introduced in the assignment document. What I need to do is to read the grey scale value of the pictures. Since each digit picture has 28 * 28 = 784 pixels, this means the feature vector without operated will be a 784-dim feature vector. And this is not easy to operate. So we need to decrease the dimensions or use some other methods to simplify the data.

Step 2

Normalization.

I tried to directly use the original data, which is a value range in 0~255, to implement the classifier. However, the result is not very good. Maybe that is because the value will have larger variance for classification algorithm to deal with.

So after getting the pixel values, I dived the value by 255 to get a matrix whose value range in 0~1.

Step 3

Mean-subtraction by the pixels and digits.

Mean-subtraction is a good method to deal with MNIST digits pictures. That is because they are not colored pictures. So the data will not be distorted too much.

Mean-subtraction can lower the entropy of the data and eliminate some redundancy, which is useful to make the calculation less and increase the accuracy.

Firstly, I just get the mean value of all the pixel data, and then subtract them in each feature. However, the result is not very good. The proper way is to calculate the mean of the pictures of the same digits, and subtract it in these corresponding picture features. Then the results are much better.

Step 4

PCA

Since Matlab has a function names pca (), which is used for reducing the matrix dimensions, it is very easy to implement this pre-process on the data. In Matlab, this function returns an n * n matrix, which represents the eigenvectors of the original data. And the columns are arranged in descending order according to the eigenvalue. So only the dimensions with larger eigenvalues have a decisive effect on the classification.

The range of the eigenvalues of the matrix ranges from about 5 to 1.5-e34, so it seems only a slight part of the dimension. However, this is not totally the fact. I will try to explain in the Naïve Bayesian Classifier.

After getting the PCA matrix, how to decide the number of the dimensions is the next problem. My method is to use the mean value of all the eigenvalues to find the decisive part of the data. It comes out that approximately 86 or 87 dimension is OK.

The PCA method is useful for optimization the calculation, In some methods, it is very helpful to increase the efficiency of the algorithm.

Part II

Implement Algorithm

Method I

SVM

I used the library 'libsvm' to set the SVM algorithm. In this problem, the class number should be 10, representing the digits 0~9, And I have 2 methods to get a 10-class classifier. One is to use the default C-SVM and directly implement it on the training data, which will automatically have a 10-class classifier. The other method is to set 10 one-class classifiers, each corresponding to a digit. And use the 10 classifiers to get the integrated result of the predicting.

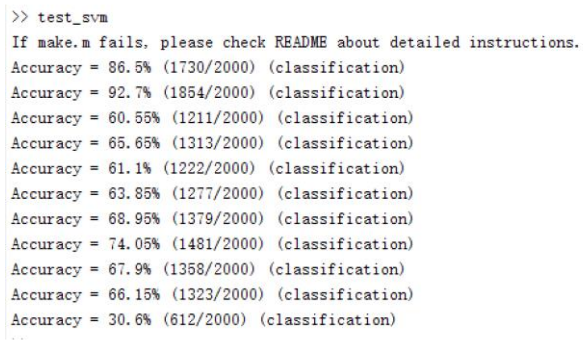
How PCA affects SVM results.

PCA methods can effectively reduce the redundancy in the calculation, and it can help return very good results.

There should be a best number of dimensions for the classification. Actually, when the dimension is around 10, we can already get a proper result. However, if the dimensions are too few, the result will be inaccurate. And if the dimensions are too many, the result will not be the best.

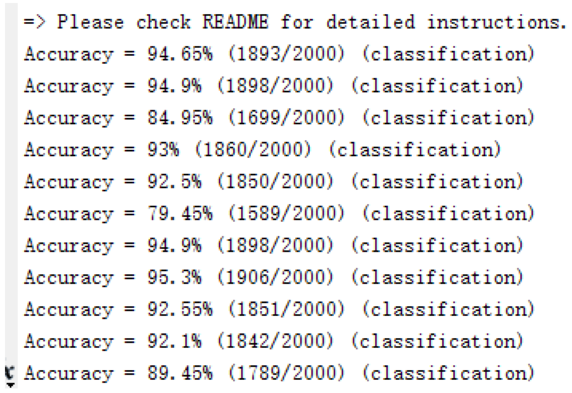
(The last result is the 10-class C-SVM accuracy, not the integrated accuracy of the 10 1-class SVM)

This is the result when d = 1



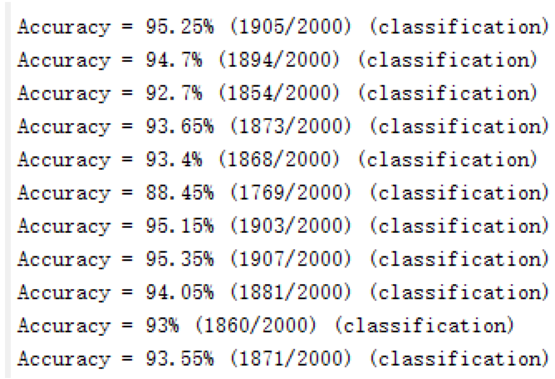
```
>> test_svm
If make.m fails, please check README about detailed instructions.
Accuracy = 86.5% (1730/2000) (classification)
Accuracy = 92.7% (1854/2000) (classification)
Accuracy = 60.55% (1211/2000) (classification)
Accuracy = 65.65% (1313/2000) (classification)
Accuracy = 61.1% (1222/2000) (classification)
Accuracy = 63.85% (1277/2000) (classification)
Accuracy = 68.95% (1379/2000) (classification)
Accuracy = 74.05% (1481/2000) (classification)
Accuracy = 67.9% (1358/2000) (classification)
Accuracy = 66.15% (1323/2000) (classification)
Accuracy = 30.6% (612/2000) (classification)
```

This is the result when d = 700



```
=> Please check README for detailed instructions.
Accuracy = 94.65% (1893/2000) (classification)
Accuracy = 94.9% (1898/2000) (classification)
Accuracy = 84.95% (1699/2000) (classification)
Accuracy = 93% (1860/2000) (classification)
Accuracy = 92.5% (1850/2000) (classification)
Accuracy = 79.45% (1589/2000) (classification)
Accuracy = 94.9% (1898/2000) (classification)
Accuracy = 95.3% (1906/2000) (classification)
Accuracy = 92.55% (1851/2000) (classification)
Accuracy = 92.1% (1842/2000) (classification)
Accuracy = 89.45% (1789/2000) (classification)
```

This is the result when d = 87



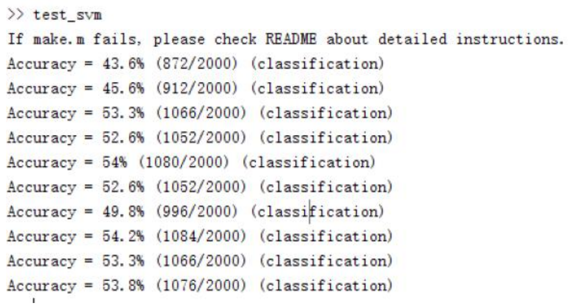
```
Accuracy = 95.25% (1905/2000) (classification)
Accuracy = 94.7% (1894/2000) (classification)
Accuracy = 92.7% (1854/2000) (classification)
Accuracy = 93.65% (1873/2000) (classification)
Accuracy = 93.4% (1868/2000) (classification)
Accuracy = 88.45% (1769/2000) (classification)
Accuracy = 95.15% (1903/2000) (classification)
Accuracy = 95.35% (1907/2000) (classification)
Accuracy = 94.05% (1881/2000) (classification)
Accuracy = 93% (1860/2000) (classification)
Accuracy = 93.55% (1871/2000) (classification)
```

Number of pictures to train the 1-class SVM

In the training step, because there are 10 1-class SVM, so we can choose to use all the data to train one SVM also, we can only use the corresponding digit pictures to train one SVM.

After comparing the result. I find that using the exact same digits pictures to train corresponding SVM is better than using all the training pictures.

The result when using all the data for a single 1-class SVM



```
>> test_svm
If make.m fails, please check README about detailed instructions.
Accuracy = 43.6% (872/2000) (classification)
Accuracy = 45.6% (912/2000) (classification)
Accuracy = 53.3% (1066/2000) (classification)
Accuracy = 52.6% (1052/2000) (classification)
Accuracy = 54% (1080/2000) (classification)
Accuracy = 52.6% (1052/2000) (classification)
Accuracy = 49.8% (996/2000) (classification)
Accuracy = 54.2% (1084/2000) (classification)
Accuracy = 53.3% (1066/2000) (classification)
Accuracy = 53.8% (1076/2000) (classification)
```

Note that when dealing with these 1-class SVMs, the label of the data need to be set accurately to the range in {-1, 1}, not [0, 1… 9], Otherwise, the accuracy will decrease as the digits number increases.

The Cross Validation of SVM is:

10-class C-SVM: 93.3%

10 1-class SVM: 40% ~ 50%

Method II

Logistic Regression

I used 'liblinear' to get the LR algorithm. In 'liblinear', there are 3 kinds of LR, L1 Logistic Regression, L2 Primal Logistic Regression and L2 Dual Logistic Regression.

Since I use the library to implement the classifiers, the program can be similar between SVM and LR

According to a reading material about Logistic Regression, this method is better for large quantity of data. In MNIST, the data size is not very big. So the LR results are a little worse than SVM. Here are the results of the 3 kinds of LR:

L2 Logistic Regression (Primal): 82.8%

Cross Validation (10 trails): 83.4%

L1 Logistic Regression: 82.9%

Cross Validation (10 trails): 81.85%

L2 Logistic Regression (dual): 82.85%

Cross Validation (10 trails): 81.6%

We can see the results are similar, while Primal L2 LR has the best classification results.

Note that for Logistic Regression in the 'liblinear', the instances are required to be sparse matrix.

Method III

Naïve Bayesian Classifier

Naïve Bayesian Classifier is a 1-class classifier based on the Bayesian Statistics, it has discrete and continuous methods. For data with normal distribution, we can use continuous method, which is by Gaussian distribution. Here for the MNIST digits this method might not be the best algorithm. However, implementing this algorithm is interesting and it also has some advantage.

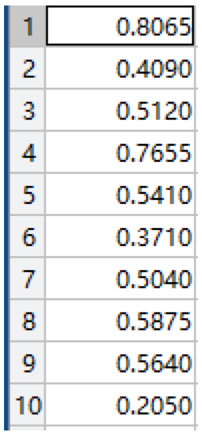
The theory of this method is calculate the mean and variance of each dimension in 2 classes. One is for the correct prediction, the other is for the wrong prediction. With a proper {-1, 1} label vector assigned to the data sets, the training data is separated into 2 parts, one is the 'correct', the other is 'wrong'. Then calculate the mean and standard deviation of the 2 parts data in all the dimension. Set the 2 2-d vector, which contains mean and std. as the classifier. Also, it is better to integrate the label or just the ratio of the 2 parts of the data along with the classifier. In this case, since there are 10 1-class classifier, it's better to have the class sign 0 ~ 9 along with the classifier.

During the programming, another important thing is the value scale. Because the maximum likelihood is a product of many-dimension values. And Matlab only support numerical limit of 2 ^ 52. So if the value is too small, it will be floored to 0, and the posterior probability will be 0. To deal with this problem, I did a log function on the Gaussian distribution value, and take the mean value of the logarithm of all the fractions, then take the exponent of inverse of the fractions as a scalar. Assign this scalar to all the probabilities. And finally, I can get a product value as the criteria. After comparing the right probability and wrong probability, I can get the predicted label of the point.

Another important thing about this method is the PCA effect. Because the original pixel values has many zeroes, so the sigma of the Gaussian distribution in some dimensions will be 0 too. And the function gaussmf () will not be able to operate. Thus a PCA step is necessary.

However, due to the distribution based decision method, this algorithm will very much based on the number of dimension. i.e. The more dimensions there are, the more accurate the results will be. After testing, I find that the accuracy is constantly increasing as the number of dimensions goes higher.

Unfortunately, my algorithm still have some problems that I didn't managed to fix, the final result of each 1-class classifier is



1	0.8065
2	0.4090
3	0.5120
4	0.7655
5	0.5410
6	0.3710
7	0.5040
8	0.5875
9	0.5640
10	0.2050

This result is very inaccurate. And I think maybe there are some bugs with the labels.