Machine Learning Summer School Project

Yongxin Zhang

2014-09-01

Contents

1 Introduction

This project is for Phoneme Classification, from the instruction of the project, we need to implement 3 algorithms we have learnt in the course to do the classification: K Nearest Neighbor; Naive Bayes classifier and Perceptron. Followed from part 3, the Model Parameters Selection and Performance Evaluation part, I used the 10-fold cross validation method for the K-NN part to choose the optimal K for the Naive Bayes part and in the Perceptron part for the optimal iteration times.

2 K-NN Classifier

2.1 Parameters Selection and algorithm creation

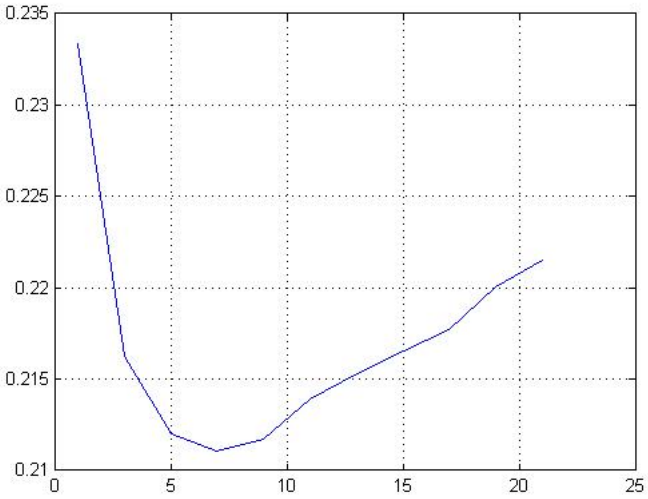
For the first case, we choose the parameter to be K. We sweep K from the default 1 to 21, just use odd number. The reason for the maximum of K is that when we increase the value of K, the average error will not decrease and the variance will also increase.

This is the code for the metric to be Euclidean distance:

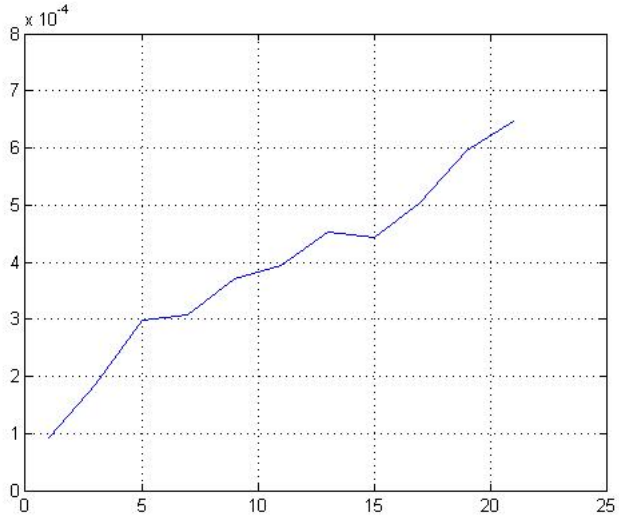
```
for k = 1:2:21
for j = 1:10
Training = (trainSet.X(:,1+1000*(j-1):1000+1000*(j-1)))';
Sample   = ([trainSet.X(:,1:1000*(j-1)),trainSet.X(:,1001+1000*(j-1):10000)])';
Group  = (trainSet.Y(1+1000*(j-1):1000+1000*(j-1)))';
Class = knnclassify(Sample, Training, Group, k, 'euclidean','nearest');
Class2 = (Class)';
Class3 = abs(Class2 - [trainSet.Y(:,1:1000*(j-1)),trainSet.Y(:,1001+1000*(j-1):10000)]);
N0 = length(Class3);
error1(j) = 0;
        error1(j) = sum(Class3);
error2(j) = error1(j)/N0;
end
error2;
M((k+1)/2) = mean(error2);
V((k+1)/2) = var(error2);

end
plot(k,M);
plot(k,V);
grid;
```

We can get the following result:

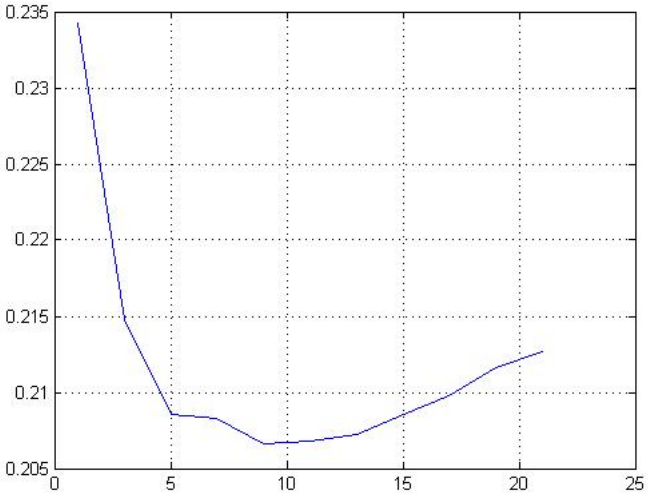


This is the average error for the training data at the Euclidean distance and we can see that at the point that K=7, we can get the minimum average error 0.2111.

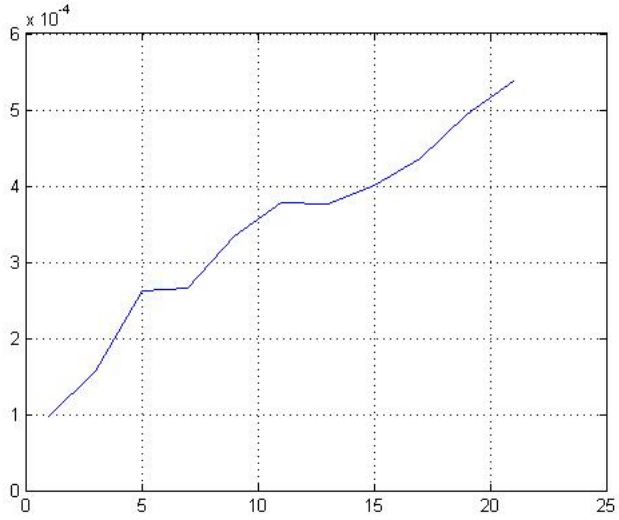


This is the variance of the error based on K, we can see that the variance is almost always increasing. But since out criterion to choose the parameter is by the minimum average error. So we will not pay much attention on the variances.

This is the code for the algorithm when we choose the distance to be absolute sum (cityblock) instead of Euclidean distance.



We can see that when K=9, we have the minimum average error of 0.2066, which is smaller than the first case. But for the 2 cases, we can see that the optimal parameter K is not the same. For the minimum average error consideration, we will use the absolute sum distance for the test set. However, since we have already had the test set data. I would like to implement the 2 cases and compare their results.



This is the variance for the absolute sum distance and we can see that the tendency is the same, the variance will increase with K.

2.2 Performance Evaluation

For this part, first we will use the K=7 and the *Euclidean* distance and we have the following code.

```
    Training = (trainSet.X(:,1:10000))';
Sample   = (testSet.X(:,1:5000))';
Group  = (trainSet.Y(1:10000))';
Class = knnclassify(Sample, Training, Group);
k = 7;
Class = knnclassify(Sample, Training, Group, k, 'euclidean','nearest');
Class2 = (Class)';
Class3 = abs(Class2 - testSet.Y(1:5000));
error1 = 0;
    error1 = sum(Class3);
error2 = error1/N0
```

We can see from the result that the error in this case is $error2 = 0.1748$. Compared with the train data, we can say that the error is fair, not bad performance.

Since for another parameter, we have another K=9. Then, for the absolute sum distance, we have the following code.

```
    Training = (trainSet.X(:,1:10000))';
Sample   = (testSet.X(:,1:5000))';
Group  = (trainSet.Y(1:10000))';
Class = knnclassify(Sample, Training, Group);
k = 9;
Class = knnclassify(Sample, Training, Group, k, 'cityblock','nearest');
Class2 = (Class)';
Class3 = abs(Class2 - testSet.Y(1:5000));
error1 = 0;
    error1 = sum(Class3);
error2 = error1/N0
```

We can see from the result that the error in this case is $error2 = 0.1676$, which is less than the error we got for the previous case. Thus, we can claim that when combining the 2 parameters. The better choice for the train data is also a better choice for test data.

2.3 Improve the performance

In the data we got above, I have one list about the minimum error for a certain K and metric and we can see that when $K = 9$ and metric is absolute sum, we have the minimum error. Of course, we need to find the data we used for this minimum error.

By running the algorithm again, we can find the data we used for the minimum error is the data $5001 : 6000$.

Thus we can modify the final algorithm for the test data, with K=9 and absolute sum distance. In this way, we can find the final error to be: 0.1860.

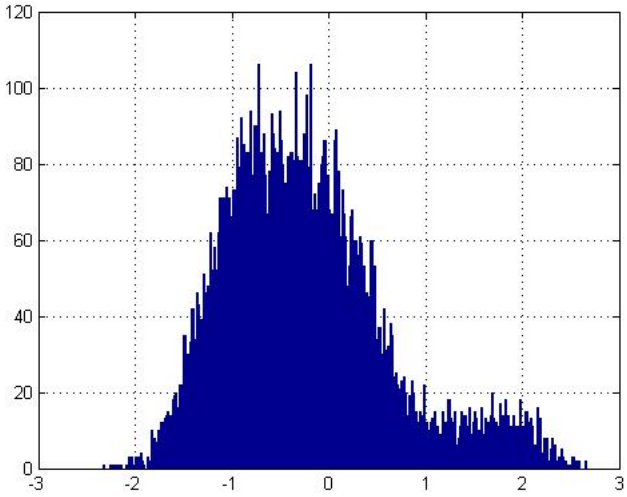
But for the metric selection, we have correlation choice, if we use correlation, even without optimization, just the same K and same data as in the previous case. We have error: 0.1634. While is better than the result for the final error for our test set. But I don't know well about this parameter. Just possible answer for the improvement.

For the parameter of the MATLAB function Class = knnclassify(Sample, Training, Group, k, 'cityblock','nearest');. The first parameter *Sample* is the data in the test set or it is the data that we need to make the classification. The second parameter $Training$ is the input parameter of the training data, this will be used as the reference coordinate for the *Sample* data. Thirdly, the parameter *Group* stands for the class of the training Set. While the $4th$ parameter K is the number of neighbors we used for the algorithm. The $5th$ parameter is the distance choice, for example, the Euclidean choice or the absolute sum and finally, it is the rule we choose to select the classification class, it can be the *nearest* or *randomly* choice.

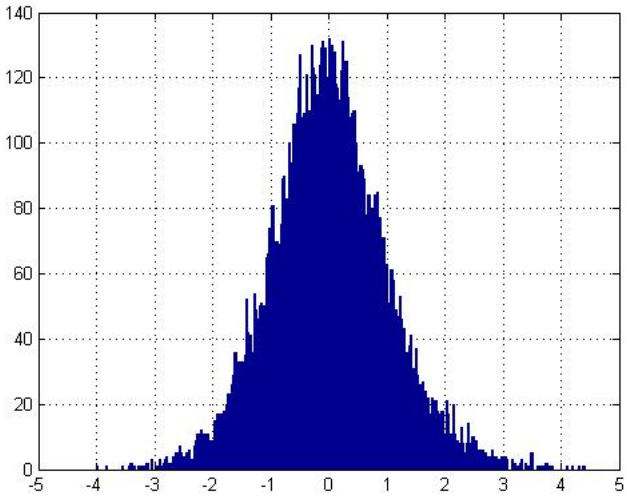
3 Naive Bayes Classifier

3.1 Parameters\Model Selection and algorithm creation

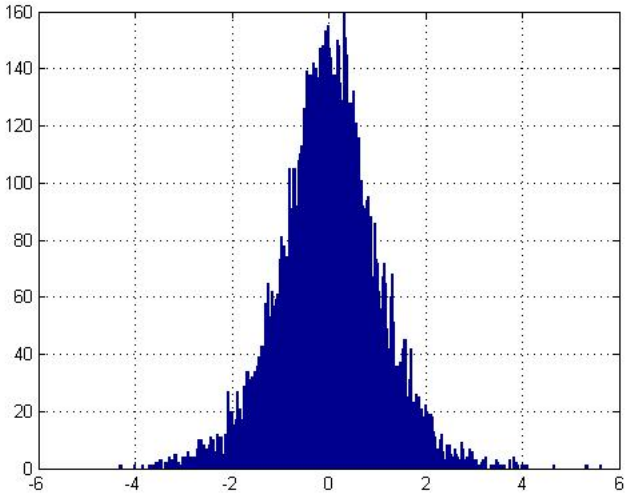
Firstly we need to estimate the distribution of the input data, or it will be hard for us to find an approximate distribution for the question. For the data we have, it is 41 dimensional input data. By using the *hist* function in MATLAB, we can have some graphs regarding to the input of the data.



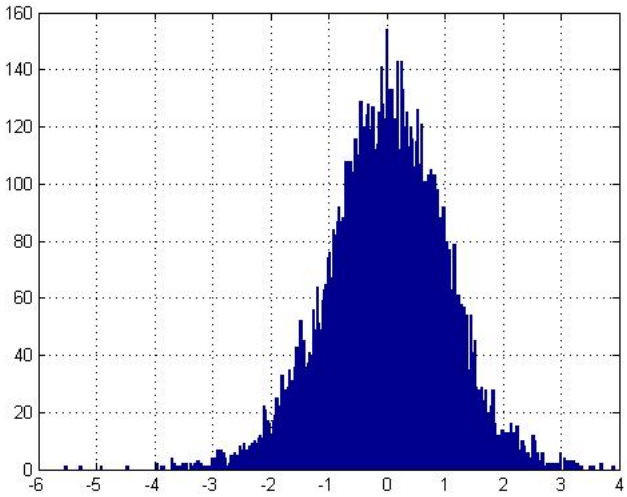
This is one input data regarding to the $1st$ feature.



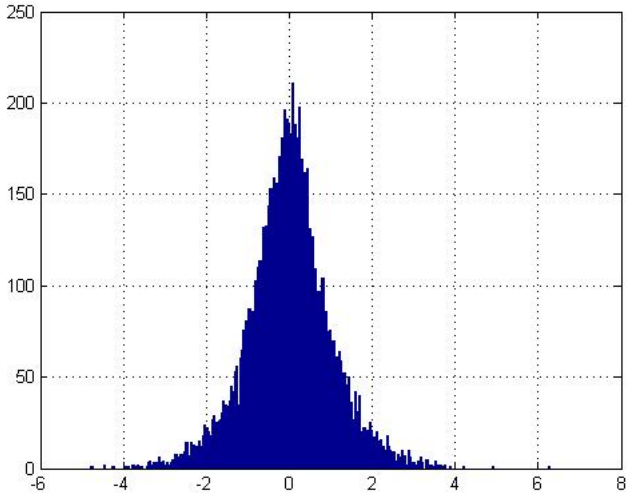
This is the graph for the $11th$ feature.



This is the graph for the $21st$ feature.



This is the graph for the $31st$ feature.



This is the graph for the $41st$ feature.

The criterion for choosing the example graph is using the statistical method, choosing $10 \times (k-1)+1$ feature. We can see that all the features are like Gaussian distribution. Thus, perhaps the parametric method is a good choice for us. And for simplicity, let's assume each feature is independent of other features.

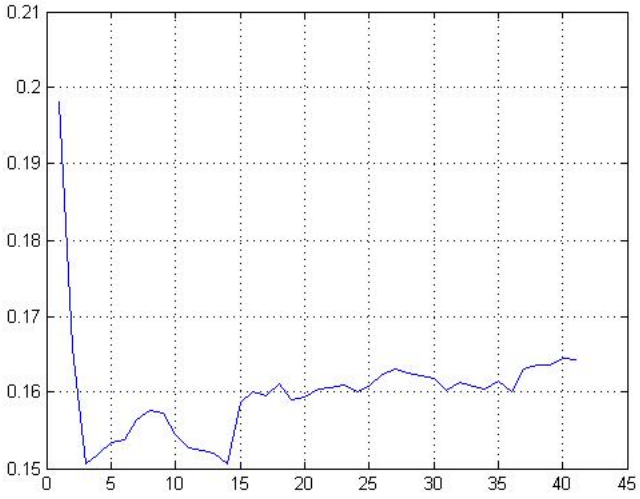
The advantage for this method is simple, and if the true distribution is this one, it works pretty well. While the disadvantage is that the assumption that the features are independent cannot be guaranteed and perhaps the distribution we choose cannot represent the true case.

Because we don't have the parameter for the chosen distribution, thus, we need to estimate the mean and covariance for it. The problem is that if the parameters we estimated are not correct, it would be a big problem. In general, if the data is generated randomly and if we have big sample data, the error would be negligible.

For the Naive Bayes method, there is not explicit parameter that we can use as the N-fold method. However, actually we still can have some parameters to choose. For example, I choose the dimensions of the train set data as parameter and sweep it from 1 to 41 (but it is in order, for example, if I choose the parameter to be 2, then the features I choose is the 1st and the 2nd. Because there is too many combinations). The code for this part is here.

```
N0 = 9000;
for j = 1:41
for k = 1:1:10
```

```
training = (trainSet.X(1:j,1000*(k-1)+1:1000*k))';
class  = (trainSet.Y(1,1000*(k-1)+1:1000*k))';
  test0  = (trainSet.X(1:j,[1:1000*(k-1),1000*k+1:10000]))';
nb = NaiveBayes.fit(training, class, 'Distribution', 'normal', 'Prior',
'empirical', 'KSWidth', 10000, 'KSSupport', 'unbounded', 'KSType', 'normal');
cpre = predict(nb,test0);
cpre0 = cpre';
temp0 = trainSet.Y(1,[1:1000*(k-1),1000*k+1:10000]);
cpre1 = abs(cpre0-temp0);
error0(k) = 0;
error0(k) = sum(cpre1);
end
error1(j) = mean((error0(k)))/N0;
end
a = 1:1:41;
plot(a,error1);
grid;
```

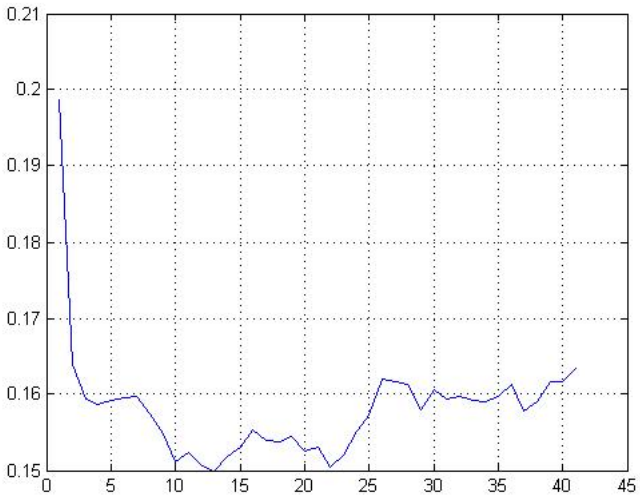


From the error result, we can see that when $k = 14$, we have the minimum average error 0.1506. Then we will use the 1st to the 14th features for the test set.

3.2 Performance Evaluation and possible improvement

Using the features we got above, we can get the error for the test set is 0.1518, which is better than the K-NN method.

For the possible improvement, I don't have clear idea now. We can choose to sweep the feature parameter for the test set to get better result. But this doesn't make sense since we shouldn't know the test data before choosing K. But never mind, let's try it first.



We can see the minimum error is at the number of features to be $k = 13$ where we can get 0.1500 error value. It is slightly better than the parameter we choose for $k = 14$.

The code for the problem is:

```
for j = 1:41

training = (trainSet.X(1:j,:))';
class  = (trainSet.Y)';
    test0  = (testSet.X(1:j,:))';
nb = NaiveBayes.fit(training, class, 'Distribution', 'normal', 'Prior',
    'empirical', 'KSWidth', 10000, 'KSSupport', 'unbounded', 'KSType', 'normal');
cpre = predict(nb,test0);
cpre0 = cpre';
cpre1 = abs(cpre0-testSet.Y);
error0 = 0;
N0 = length(cpre1);
for i = 1:N0
error0 = cpre1(i) + error0;
end
error1(j) = error0/N0;
end
```

```
error1;
a = 1:1:41;
plot(a,error1);
grid;
```

For the parameters of the NaiveBayes.fit and predict function. From the MATLAB help command, we can see that nb = NaiveBayes.fit(training, class, 'Distribution', 'normal', 'Prior', 'empirical', 'KSWidth', 10000, 'KSSupport', 'unbounded', 'KSType', 'normal').This is the NaiveBayes.fit fucntion and we can see the $1st$ parameter is X coordinate of the training data and the $2nd$ parameter is the Y/class the training data. The $3rd$ combination is the chosen of the Distribution, with the default to be *normal* and we also choose normal for the first glance of hist. The $4th$ parameter is about the $Prior$ and we learnt in this course, the $NaiveBayes$ method use the $'empirical'$ choice. While for the $KSWidth$, for some distribution, it will use this parameter to set the number of training data we use. While for the KSSupport, it is the range of the density function, since in our case, we assume normal distribution, we set this parameter to be *unbounded*. Finally we come to the last parameter KSType, from *help*, we know it is type of kernel smoother to use (actually I don't know the meaning of it).

4 The Perceptron Algorithm

4.1 The algorithm creation

We know from the question that this is the classification problem with the label $y \in \{0, 1\}$. By letting $y^{'} = 2y - 1$, we can map the label to be $\{-1, +1\}$. Thus, we can use the on-line learning algorithm.

• Initialization: The parameters vector is initialized to be ω_0, for instance, we set it to be 0

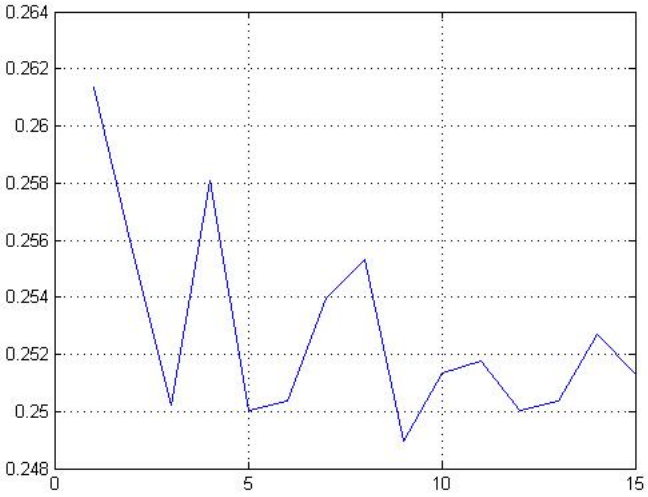
• At each iteration $t = 1, 2, 3...$

⊖ Pick one sample from the training set $\{x_k\}_{k=1}^{n}$

⊖ Compute the perceptron's output for x_t, using the current weight vector ω_t and then we will have $Y_t = f(\omega_t^T x_t)$.

⊖ Update the weight vector $\omega_{t+1} = \omega_t + \eta(y_t - Y_t)x_t$

This is the algorithm of the Perceptron and here I use the samples in the training set more than one time and will find the relationship between the number of times and the average error for the test set. To get the iteration times, I use the 10-fold method again and we can have the following code and result.



We can see that for the training data, when the iteration number is 9, we have the minimum average error to be 0.2490.

The code for this part is:

```
ita = 0.1;
N0 = 9000;
N1 = 15;
Y0 = 2*trainSet.Y - 1;
Y1 = 2*testSet.Y - 1;
wt0 = zeros(41,N1);
yt = 0;
yt1 = 0;
t22 = zeros(1,N0);
error0 = zeros(1,N1);
t11 = zeros(1,N0);

for j = 1:1:N1
error00 = zeros(1,10);
    error0(j) = 0;

for k = 1:1:10
wt = zeros(41,1);
for m = 1:1:j
for i = 1000*(k-1)+1:1000*k
yt = sign(wt'*trainSet.X(:,i));
wt = wt + ita*(Y0(i) - yt)*trainSet.X(:,i);
end
```

```
end
wt0(:,k) = wt;
trainset0 = trainSet.X(:, [1:1000*(k-1),1000*k:10000]);
        y00 = Y0(1, [1:1000*(k-1),1000*k:10000]);
for ii = 1:1:N0
yt1 = sign(wt0(:,k)'*trainset0(:,ii));
t11(ii) = (y00(ii) - yt1);
t22(ii) = sign (abs(t11(ii)));
error00(k) = error00(k) + t22(ii);
end
    end
error00 = error00/N0;
error0(j) = mean(error00);
end
m0 = 1:1:N1;
plot(m0,error0);
grid;
```

4.2 Performance Evaluation

While for the test data, we fix the iteration parameter to be 9 and the code for this part is:

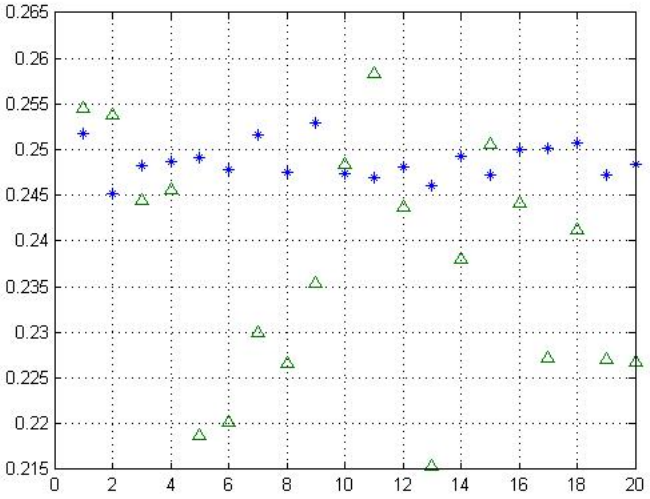
```
ita = 0.1;
N0 = 5000;
Y0 = 2*trainSet.Y - 1;
Y1 = 2*testSet.Y - 1;
yt = 0;
yt1 = 0;
t22 = zeros(1,N0);
error00 = 0;
t11 = zeros(1,N0);

j = 9
wt = zeros(41,1);
for m = 1:1:j
for i = 1:10000
yt = sign(wt'*trainSet.X(:,i));
wt = wt + ita*(Y0(i) - yt)*trainSet.X(:,i);
end
end

for ii = 1:1:N0
yt1 = sign(wt0(:,k)'*testSet.X(:,ii));
t11(ii) = (Y1(ii) - yt1);
t22(ii) = sign (abs(t11(ii)));
```

```
error00 = error00 + t22(ii);
end
error00 = error00/N0
```

We can get the final error of it is 0.2354 when choosing the iteration number to be 9.



From the requirement of the document, we need to show the training error and the test error on the same axe. Here the triangle is the test error while the blue star is the averaged train set error. Here we can see that when the iteration number equals to 13, we have the minimum error of 0.2154. We can see that the actual optimal iteraton number is not identically the same as the optimal iteration number we got from the training set.

The code for this part is as follows:

```
ita = 0.1;
N0 = 10000;
N1 = 20;
Y0 = 2*trainSet.Y - 1;
Y1 = 2*testSet.Y - 1;
wt = zeros(41,1);
wt0 = zeros(41,N1);
yt  = 0;
yt1 = 0;
t22 = zeros(1,5000);
t1 = zeros(1,10000);
t2 = zeros(1,10000);
```

```
error0 = zeros(1,N1);
error00 = zeros(1,N1);

t11 = zeros(1,5000);

for j = 1:1:N1
    error0(j) = 0;
      error00(j) = 0;
    for i = 1:10000
        yt = sign(wt'*trainSet.X(:,i));
        wt = wt + ita*(Y0(i) - yt)*trainSet.X(:,i);
        t1(i) = (Y0(i) - yt);
        t2(i) = sign (abs(t1(i)));
        error0(j) = error0(j) + t2(i);
    end
    wt0(:,j) = wt;


    for ii = 1:1:5000
        yt1 = sign(wt0(:,j)'*testSet.X(:,ii));

        t11(ii) = (Y1(ii) - yt1);
        t22(ii) = sign (abs(t11(ii)));
        error00(j) = error00(j) + t22(ii);
    end

end

error1 = error0/10000
error11 = error00/5000
k = 1:1:N1;
plot(k,error1,'*',k,error11,'^');
grid;
```

Here is a little bit tricky, because the stopping condition has nothing to do with the value of error, I set the number of iterations as parameter and sweep it. For the maximum of the parameter, I just set it to be pretty big ($N_1 = 15$) and intuitively, I think it is large enough to cover the optimal iteration value.

5 Conclusion

In this project, we implemented 3 algorithms and for each algorithm, we also had many considerations. We can see that all of the 3 algorithms can give us some results. For the K-NN algorithm, following the instruction of the project, we can achieve an error of 0.1676 with $K = 9$ and the metric is the absolute sum distance. For the Naive Bayes algorithm, by using the features from 1 to 13, we can achieve the classification error for the test set to be 0.1518, which is better than the K-NN method. Then for the Perceptron algorithm, by using the iteration number to be 9, we can achieve the classification error to be 0.2354, which is the largest among the 3 algorithms.

From the course, we know that only the Naive Bayes is the parametric method and in this project, it gives us the best result. Thus, from this project, I want to say if the input data really has some distribution and if we can find the correct distribution, it will give us very good result. Although the K-NN and the Perceptron algorithm can deal with almost all kinds of data we have, it cannot guarantee better result compared with the parametric method.