CAP 5638: Project #2

Due on Wednesday, Dec 2, 2015

XiuWen Liu 10:10am

YongQing Zheng

Contents

Problem 1

[Background:]

Bayesian decision theory provides the optimal decision rule for classification when the true probabilities are known. However, for pattern classification applications, the final product we need is a classifier which can be represented by a set of discriminant function. Therefore, if we can learn discriminant functions directly, we can avoid the intermediate step of estimating probability models,which arguably is more difficult than learning discriminant functions with finite training data(note that by doing this we lose the first principle and many techniques are thus ad hoc). Linear discriminant functions are widely used because they are efficient and can often be analyzed analytically. Besides, through kernel methods and boosting algorithms, they can lead to accurate classifiers for complex, real-world applications.

[Purpose:]

Learn how to realize the two class/multi-class linear discriminant functions through perceptron-like algorithms and how to use boosting algorithms to build more accurate classifiers using linear discriminant functions.
Learn how to use the two class classification algorithm to deal with the multi-class classification problem.

[Methodology:]

Implement Algorithm 4(Fixed-increment Single-sample Perceptron Algorithm) and Algorithm 8 (Batch Relaxation with Margin) of chapter 5 as the basic classifiers. Use boost method Algorithm 8 (Adaboost) to create a strong classifier based on the weak classifier. The description of the three algorithm are as follows:

Algorithm 1: Fixed Increment Single Sample Perceptron

Fixed Increment Single Sample Perceptron(a, k, n);
$a, k \leftarrow 0$;
while *all pattern does not properly classified* **do**
    $k \leftarrow (k+1) \bmod n$;
    if y^k *is misclassified by* a **then**
        $a \leftarrow a + y^k$;
    end
end
Return a;

Problem 1 continued on next page. . .

Algorithm 2: Batch relaxation with Margin

Batch Relaxation with Margin($a, \eta(.), b, k$);
$a, \eta(.), b, k \leftarrow 0$;
while $y^k \neq \{\}$ **do**
  $k \leftarrow (k+1) \bmod n$;
  $y^k = \{\}$;
  $j = 0$;
  while $j \neq n$ **do**
    $j \leftarrow j+1$;
    if $a^t y^j \leq b$ **then**
      Append y^j to y^k;
    end
  end
  $a \leftarrow a + \eta(k) \sum_{y \in y} \frac{b-a^t y}{||y||^2} y$;
end
Return a

Algorithm 3: Adaboost

Adaboost;
$D = \{x_1, y^1, ..., x^n, y_n\}$,$k_{max}$,$W_1(i) = 1/n, i = 1, ..., n$;
$k \leftarrow 0$;
while $k \leq k_{max}$ **do**
  train weak learner C_k using D sampled according to $W_k(i)$;
  $E_k \leftarrow$ training error of C_k measured on D using $W_k(i)$;
  $\alpha_k \leftarrow \frac{1}{2} ln[(1-E_k)/E_k)]$;

$$W_{k+1}(i) \leftarrow \frac{W_k(i)}{Z_k} \prod \begin{cases} e^{-\alpha_k} & if\ h_k(x^i) = y_i\ (correctly\ classified) \\ e^{\alpha_k} & if\ h_k(x^i) \neq y_i\ (incorrectly\ classified) \end{cases}$$

end
Return C_k and α_k for $k = 1$ to k_{max} (ensemble of classifiers with weights);

To deal with the multi-class classification problem, we use the one against other and one against rest methods. For the one against rest method, we need to build p classifiers based on one class against the rest, and choose the final class which has the maximum linear discriminant function result. For the one against other method, we need to build $p(p-1)/2$ classifiers based on the each pairs of the class and use the voting scheme to decide the class of the test data.

[Dataset:]

We use two dataset to build and test our model.
The first one is UCI wine dataset. Training set of this data consists of 89 examples in three class(30 in class 1, 36 in class 2 and 24 in class 3). The test set consists also of 89 examples(29 in class 1, 36 in class 2, and 24 in class 3).
Another is USPS handwritten digit dataset. The training set consists of 2930 training samples (1194 in digit 0,1005 in digit 1, and 731 in digit 2). The test set consists of 821 samples(359 in digit 0, 264 in digit 1 and 198 in digit 2)

[Expriment results:]

We use Matlab and python to build our model to solve this project. From our calculation, we found that the USPS dataset was linearly separable compared with the UCI dataset which might not be properly separated.

Figure 1: Analysis of the UCI data set

UCI data set	Two Classes			Multi Classes	
Perceptron	class1-agianst-rest	class2-agianst-rest	class3-against-rest	one-against-rest	one-against-other
Accuracy	67.42%	59.55%	26.97%	40.45%	26.97%
Iteration	10	10	10	10	10
CPU time(s)				0.0076	0.0081
Perceptron	class1-agianst-rest	class2-agianst-rest	class3-against-rest	one-against-rest	one-against-other
Accuracy	67.42%	59.55%	52.81%	56.18%	26.97%
Iteration	100	100	100	100	100
CPU time(s)				0.038	0.038
Perceptron	class1-agianst-rest	class2-agianst-rest	class3-against-rest	one-against-rest	one-against-other
Accuracy	94.38%	59.55%	53.93%	40.45%	35.96%
Iteration	1000	1000	1000	1000	1000
CPU time(s)				0.337	0.227
Perceptron	class1-agianst-rest	class2-agianst-rest	class3-against-rest	one-against-rest	one-against-other
Accuracy	94.38%	66.29%	35.96%	40.45%	60.67%
Iteration	10000	10000	10000	10000	10000
CPU time(s)				3.36	2.21
Batch relaxation	class1-agianst-rest	class2-agianst-rest	class3-against-rest	one-against-rest	one-against-other
Accuracy	67.42%	59.55%	73.03%	32.58%	26.97%
Iteration	624	593	694	a1:624 a2:593 a3:694	a1:892 a2:1398 a3:1133
CPU time(s)				0.42	0.5

Figure 1 shows the result of both two class and multi-class problem. For each problem, we use two algorithm, Perceptron and Batch relaxation methods, to predict the class based on UCI data training and testing set. The Perceptron algorithm for this data set does not converge, since we can see that when the iteration numbers increases(x10), the accuracy rate does not change a lot, sometimes even decreases. For the perceptron method, it performances better for the class 1 against rest situation compared with other two cases. For the Batch relaxation method, we use η equal to 0.1 and b equal to 1. It converge for the training data, the iteration number is around 600. Class 3 against rest works better for this classifier.

From Figure 2, we can see that both perceptron and batch relaxation performance excellent on the USPS data set.The accuracy rate for all the two classes cases are above 95%, besides, for the multi-class problem, both the one against and one against other algorithm are efficient. Perceptron method is a little better than Batch relaxation method for the multi-class problem, around 97 % accuracy rate vs around 94 respectively%.
Figure 3 to Figure 6 show the result of ada boosting based on the above two weaker classifier. We also consider the UCI and USPS data together.

Figure 3 and figure 4 show the influence of the adaboosting method on the UCI data set, we use both perceptron and batch relaxation methods as weaker classifier. Since the UCI dataset is not linearly separated, we can see from the table, results for both algorithms with boosting performances better than the algorithm without boosting. Let's take Fixed single sample perceptron as example, under the Class 1 vs Class 3 cases, boosting method increase the accuracy rate from 86.54% to 94.23%.

Figure 5 and Figure 6 gives us the testing predicting results of USPS data based on two algorithms with and without ada-boosting.

Since this dataset is nearly linearly separable, so the original classifier itself can give a very good classification results. According to this reason, methods with boosting does not improve the result significantly. All the results for the one against other methods are above or around 95 %

[Program:]

Our team use both Matlab and python to realize the programming process for this project. The attachment is the code for reference.

List 1 shows a python and matlab script.

Listing 1: Python program for the project

```
#%% improt package
import os
import pandas as pd
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
#import matplotlib.pyplot as plt
import time

os.chdir("/home/jianwang/Documents/python/pattern_recognition")

#%% read data
uci_train=pd.read_csv("wine_uci_train.txt",header=-1,sep=" ")
uci_test=pd.read_csv("wine_uci_test.txt",header=-1,sep=" ")
# use the fisrt column as class and the remain columns as the features
uci_train_y=uci_train[uci_train.columns[0]]
uci_train_x=uci_train[uci_train.columns[1:14]]

zip_train_small=pd.read_csv("zip_train_0_2_small.txt",header=-1,sep=" ")
zip_test_small=pd.read_csv("zip_test_0_2_small.txt",header=-1,sep=" ")
zip_train=pd.read_csv("zip_train_0_2.txt",header=-1,sep=" ")
zip_test=pd.read_csv("zip_test_0_2.txt",header=-1,sep=" ")

#%% build model for the porject


"""
fist for the algorithm 4
"""
class pattern_recognition(object):
    def compute_error(self, data_target,data_predict):
        """
        input: class predict and class target
        return: error rate
        """
        sample_num=len(data_target)
        right_class=sum([1 for i in range(sample_num) if data_target[i]==data_predict[i]])
        return 1-right_class/sample_num
    def data_rebuild(self, data_x,data_y):
        """
        for class 1 remain same
```

```
        for class 2 change x to its opppsite, that is -1*x
        assume class1 as 1 and class as 2
        input: data_x, data_y
        return new data_x and data_y
        """
        sample_num = len(data_y)
        data_x_new=[]
        for i in range(sample_num):
            if data_y[i]!=1:
                data_x_new.append([item*-1 for item in data_x[i]])
            else: data_x_new.append(data_x[i])
        return np.array(data_x_new),np.array(data_y)


    def  fissp_fit(self, train_x, train_y, a,kmax):
        sample_num=len(train_y)
        a=a
        k =1
        #transfer to the np.array type
        train_x=np.array(train_x)
        train_y=np.array(train_y)
        a=np.array(a)
        # rebuild the data
        train_x,train_y=self.data_rebuild(train_x,train_y)
        while True:
            if k<=kmax:
                for i in range(sample_num):
                    if np.dot(a,train_x[i])<=0:
                        a=a+train_x[i]
            else: break
            y_predict=np.array([np.dot(a,train_x[i]) for i in range(sample_num)])
            k=k+1
            # define if all the y_predict bigger than zero
            if all([item>0 for item in y_predict]):
               break
        return np.array(a),k
    """
    algorthm 8
    """
    def  brwm_fit(self, train_x, train_y, a,kmax,eta=0.1,b=1):
        sample_num=len(train_y)
        a=a
        k=1
        #transfer to the np.array type
        train_x=np.array(train_x)
        train_y=np.array(train_y)
        a=np.array(a)
        # rebuild the data
        train_x,train_y=self.data_rebuild(train_x,train_y)
        while True:
            if k<=kmax:
                y=[]
                for i in range(sample_num):
```

```
                    if np.dot(a,train_x[i])<=b:
                        y.append(train_x[i])
                a= a+eta*sum([item*(b-np.dot(a,item))/np.linalg.norm(item)**2 for item in y])
                k=k+1
            else:
                break
            if not y:
                break
        return np.array(a),k

    def adaboost(self,train_x, train_y, kmax=100,algorithm=1):
        sample_num=len(train_y)
        w=[1/sample_num]*sample_num
        k =1
        #rebuild data for times -1 on class 2
        if algorithm==1:
            result=[]
            alpha_list=[]
            weight_list=[]
            while k<=kmax:
                k=k+1
            #sample data:use choice to sample data with known weights
            #note only can resample the one dimensional data
            #note that the sample size will be the 1/3 of the original
                train_data_index=np.random.choice(list(range(0,sample_num)),sample_num//3,replac
                train_data_x=train_x[train_data_index]
                train_data_y=train_y[train_data_index]
                a=[1]*train_data_x.shape[1]
                a=self.fissp_fit(train_data_x,train_data_y,a,kmax)[0]
                weight_list.append(a)
                #calculate the result for the weak classifier
                result_c=[]
                for i in range(sample_num):
                    if np.dot(a,train_x[i])>0:
                       result_c.append(1)
                    else: result_c.append(-1)
                # next find all the miss classification for the whole data
                error=0
                for i in range(sample_num):
                    if result_c[i]!=train_y[i]:
                       error=error+w[i]
                if error==0:
                    weight_list.pop()
                    break
                alpha=0.5*np.log((1-error)/error)
                alpha_list.append(alpha)
               # next update the weights
                for i in range(sample_num):
                    if result_c[i]!=train_y[i]:
                       w[i]=w[i]*np.exp(1)**(alpha)
                    else: w[i]=w[i]*np.exp(1)**(-alpha)
                z=sum(w)
                w=np.array(w)/z
```

```
                #add the result of this loop to the final result list
                result.append(result_c)

        if algorithm==2:
            result=[]
            alpha_list=[]
            weight_list=[]
            while k<=kmax:
                k=k+1
            #sample data:use choice to sample data with known weights
            #note only can resample the one dimensional data
            #note that the sample size will be the 1/3 of the original
                train_data_index=np.random.choice(list(range(0,sample_num)),sample_num//3,replac
                train_data_x=train_x[train_data_index]
                train_data_y=train_y[train_data_index]
                a=[1]*train_data_x.shape[1]
                a=self.brwm_fit(train_data_x,train_data_y,a,kmax)[0]
                weight_list.append(a)
                #calculate the result for the weak classifier
                result_c=[]
                for i in range(sample_num):
                    if np.dot(a,train_x[i])>0:
                       result_c.append(1)
                    else: result_c.append(-1)
                # next find all the miss classification for the whole data
                error=0
                for i in range(sample_num):
                    if result_c[i]!=train_y[i]:
                       error=error+w[i]
                if error==0:
                    weight_list.pop()
                    break
                alpha=0.5*np.log((1-error)/error)
                alpha_list.append(alpha)
               # next update the weights
                for i in range(sample_num):
                    if result_c[i]!=train_y[i]:
                       w[i]=w[i]*np.exp(1)**(alpha)
                    else: w[i]=w[i]*np.exp(1)**(-alpha)
                z=sum(w)
                w=np.array(w)/z
                #add the result of this loop to the final result list
                result.append(result_c)
        return np.array(alpha_list), np.array(weight_list),k
#%% model test for zip
## test the compute_error function
#a=[1,2,3]
#b=[1,3,2]
#pattern_recognition().compute_error(a,b)
#
#test the data_rebuild function
#a=[[1,2,3],[2,3,4]]
#b=[1,-1]
```

```
#pattern_recognition().data_rebuild(a,b)


kmax=200
b=1

zip_train_y=np.array(zip_train_small)[:,0]
zip_train_x=np.array(zip_train_small)[:,1:]

zip_test_y=np.array(zip_test_small)[:,0]
zip_test_x=np.array(zip_test_small)[:,1:]

retain_class1=0
retain_class2=2
del_class=1
zip_train_y=np.delete(zip_train_y,np.where([zip_train_y==del_class]),axis=0)
zip_train_y[zip_train_y==retain_class2]=-1
zip_train_y[zip_train_y==retain_class1]=1,
zip_test_y=np.delete(zip_test_y,np.where([zip_test_y==del_class]),axis=0)
zip_test_y[zip_test_y==retain_class2]=-1
zip_test_y[zip_test_y==retain_class1]=1


zip_train_x=np.delete(zip_train_x,np.where([np.array(zip_train_small)[:,0]==del_class]),axis=0)

zip_test_x=np.delete(zip_test_x,np.where([np.array(zip_test_small)[:,0]==del_class]),axis=0)



#"""
#test the fissp function
#"""
#a=[1]*zip_train_x.shape[1]
#t=time.time()
#w,k_fissp=pattern_recognition().fissp_fit(zip_train_x,zip_train_y,a,kmax)
#cpu_time_fissp=time.time()-t
#
##training error
#zip_train_x_predict=np.dot(zip_train_x,w)
#zip_train_x_predict[zip_train_x_predict>0]=1
#zip_train_x_predict[zip_train_x_predict<=0]=-1
#train_correct=sum(zip_train_x_predict==zip_train_y)/len(zip_train_y)
#
##testing error
#zip_test_x_predict=np.dot(zip_test_x,w)
#zip_test_x_predict[zip_test_x_predict>0]=1
#zip_test_x_predict[zip_test_x_predict<=0]=-1
#
#test_correct_fissp=sum(zip_test_x_predict==zip_test_y)/len(zip_test_y)
#
#result_fissp=[test_correct_fissp,k_fissp,cpu_time_fissp]
```

```
#"""
#test the brwm function
#"""
#a=[1]*zip_train_x.shape[1]
#t=time.time()
#w,k_fissp=pattern_recognition().brwm_fit(zip_train_x,zip_train_y,a,kmax,eta=0.1,b=b)
#cpu_time_fissp=time.time()-t
##training error
#zip_train_x_predict=np.dot(zip_train_x,w)
#zip_train_x_predict[zip_train_x_predict>0]=1
#zip_train_x_predict[zip_train_x_predict<=0]=-1
#train_correct=sum(zip_train_x_predict==zip_train_y)/len(zip_train_y)
#
##testing error
#zip_test_x_predict=np.dot(zip_test_x,w)
#zip_test_x_predict[zip_test_x_predict>0]=1
#zip_test_x_predict[zip_test_x_predict<=0]=-1
#
#test_correct_brwm=sum(zip_test_x_predict==zip_test_y)/len(zip_test_y)
#result_brwm=[test_correct_brwm,k_fissp,cpu_time_fissp]

#
#test the adaboosting
#

t=time.time()
alpha,weight,k_ada=pattern_recognition().adaboost(zip_train_x,zip_train_y,kmax=100,algorithm=2)
cpu_time=time.time()-t
# use alpha and result get discriminant function g(x)

h=np.dot(weight,zip_train_x.T)

for i in range(h.shape[0]):
    for j in range(h.shape[1]):
        if h[i,j]>0:
            h[i,j]=1
        else:
            h[i,j]=-1

g=np.dot(alpha,h)

train_predict=[1 if item>0 else -1 for item in g]

train_correct=sum(train_predict==zip_train_y)/len(zip_train_y)

#test error

h=np.dot(weight,zip_test_x.T)

for i in range(h.shape[0]):
    for j in range(h.shape[1]):
        if h[i,j]>0:
            h[i,j]=1
```

```
        else:
            h[i,j]=-1

g=np.dot(alpha,h)

test_predict=[1 if item>0 else -1 for item in g]

test_correct_ada=sum(test_predict==zip_test_y)/len(zip_test_y)

result_ada=[test_correct_ada,k_ada,cpu_time]

#%% model test for wine
## test the compute_error function
#a=[1,2,3]
#b=[1,3,2]
#pattern_recognition().compute_error(a,b)
#
#test the data_rebuild function
#a=[[1,2,3],[2,3,4]]
#b=[1,-1]
#pattern_recognition().data_rebuild(a,b)


kmax=150
b=1


uci_train_y=np.array(uci_train)[:,0]
uci_train_x=np.array(uci_train)[:,1:14]

uci_test_y=np.array(uci_test)[:,0]
uci_test_x=np.array(uci_test)[:,1:14]

#build the one against rest data, both training and testing
#build the one over other data
retain_class1=2
retain_class2=3
del_class=1
uci_train_y=np.delete(uci_train_y,np.where([uci_train_y==del_class]),axis=0)
uci_train_y[uci_train_y==retain_class2]=-1
uci_train_y[uci_train_y==retain_class1]=1

uci_test_y=np.delete(uci_test_y,np.where([uci_test_y==del_class]),axis=0)
uci_test_y[uci_test_y==retain_class2]=-1
uci_test_y[uci_test_y==retain_class1]=1


uci_train_x=np.delete(uci_train_x,np.where([np.array(uci_train)[:,0]==del_class]),axis=0)

uci_test_x=np.delete(uci_test_x,np.where([np.array(uci_test)[:,0]==del_class]),axis=0)


#
```

```
#"""
#test the fissp function
#"""
#a=[1]*uci_train_x.shape[1]
#t=time.time()
#w,k_fissp=pattern_recognition().fissp_fit(uci_train_x,uci_train_y,a,kmax)
#cpu_time_fissp=time.time()-t
#
##training error
#uci_train_x_predict=np.dot(uci_train_x,w)
#uci_train_x_predict[uci_train_x_predict>0]=1
#uci_train_x_predict[uci_train_x_predict<=0]=-1
#train_correct=sum(uci_train_x_predict==uci_train_y)/len(uci_train_y)
#
##testing error
#uci_test_x_predict=np.dot(uci_test_x,w)
#uci_test_x_predict[uci_test_x_predict>0]=1
#uci_test_x_predict[uci_test_x_predict<=0]=-1
#
#test_correct_fissp=sum(uci_test_x_predict==uci_test_y)/len(uci_test_y)
#
#result_fissp=[test_correct_fissp,k_fissp,cpu_time_fissp]
#
#"""
#test the brwm function
#"""
#a=[1]*uci_train_x.shape[1]
#t=time.time()
#w,k_brwm=pattern_recognition().brwm_fit(uci_train_x,uci_train_y,a,kmax,eta=0.1,b=b)
#cpu_time_brwm=time.time()-t
##training error
#uci_train_x_predict=np.dot(uci_train_x,w)
#uci_train_x_predict[uci_train_x_predict>0]=1
#uci_train_x_predict[uci_train_x_predict<=0]=-1
#train_correct=sum(uci_train_x_predict==uci_train_y)/len(uci_train_y)
#
##testing error
#uci_test_x_predict=np.dot(uci_test_x,w)
#uci_test_x_predict[uci_test_x_predict>0]=1
#uci_test_x_predict[uci_test_x_predict<=0]=-1
#
#test_correct_brwm=sum(uci_test_x_predict==uci_test_y)/len(uci_test_y)
#result_brwm=[test_correct_brwm,k_brwm,cpu_time_brwm]

"""
test the adaboosting
"""
t=time.time()
alpha,weight,k=pattern_recognition().adaboost(uci_train_x,uci_train_y,kmax=200,algorithm=2)
cpu_time=time.time()-t
# use alpha and result get discriminant function g(x)

h=np.dot(weight,uci_train_x.T)
```

```
for i in range(h.shape[0]):
    for j in range(h.shape[1]):
        if h[i,j]>0:
            h[i,j]=1
        else:
            h[i,j]=-1

g=np.dot(alpha,h)

train_predict=[1 if item>0 else -1 for item in g]

train_correct=sum(train_predict==uci_train_y)/len(uci_train_y)

#test error

h=np.dot(weight,uci_test_x.T)

for i in range(h.shape[0]):
    for j in range(h.shape[1]):
        if h[i,j]>0:
            h[i,j]=1
        else:
            h[i,j]=-1

g=np.dot(alpha,h)

test_predict=[1 if item>0 else -1 for item in g]

test_correct=sum(test_predict==uci_test_y)/len(uci_test_y)

result_ada=[test_correct,k,cpu_time]
#--------------------------------------
#%%class 1 vs rest fssip
##--------------------------------------
#
#zip_train_y=np.array(zip_train_small)[:,0]
#zip_train_x=np.array(zip_train_small)[:,1:]
#
#zip_test_y=np.array(zip_test_small)[:,0]
#zip_test_x=np.array(zip_test_small)[:,1:]
#zip_train_y[zip_train_y==1]=-1
#zip_train_y[zip_train_y==2]=-1
#zip_train_y[zip_train_y==0]=1
#
#train_num =len(zip_train_y)
#zip_test_y[zip_test_y==1]=-1
#zip_test_y[zip_test_y==2]=-1
#zip_test_y[zip_test_y==0]=1
#test_num =len(zip_test_y)
#
#
#a=[1]*zip_train_x.shape[1]
```

```
#w,k=pattern_recognition().fissp_fit(zip_train_x,zip_train_y,a,kmax)
## test the train error:
#zip_train_x_predict=np.dot(zip_train_x,w)
#zip_train_x_predict[zip_train_x_predict>0]=1
#zip_train_x_predict[zip_train_x_predict<=0]=-1
#sum(zip_train_x_predict==zip_train_y)/len(zip_train_y)
#
##test the testing error
#zip_test_x_predict=np.dot(zip_test_x,w)
#zip_test_x_predict[zip_test_x_predict>0]=1
#zip_test_x_predict[zip_test_x_predict<=0]=-1
#
#
#sum(zip_test_x_predict==zip_test_y)/len(zip_test_y)
#
#
#
#
##--------------------------------------
##%%class 2 vs rest fssip
##--------------------------------------
#zip_train_y=np.array(zip_train_small)[:,0]
#zip_train_x=np.array(zip_train_small)[:,1:]
#
#zip_test_y=np.array(zip_test_small)[:,0]
#zip_test_x=np.array(zip_test_small)[:,1:]
#zip_train_y[zip_train_y==0]=-1
#zip_train_y[zip_train_y==2]=-1
#zip_train_y[zip_train_y==1]=1
#
#train_num =len(zip_train_y)
#zip_test_y[zip_test_y==0]=-1
#zip_test_y[zip_test_y==2]=-1
#zip_test_y[zip_test_y==1]=1
#test_num =len(zip_test_y)
#
#a=[1]*zip_train_x.shape[1]
#w=pattern_recognition().fissp_fit(zip_train_x,zip_train_y,a,kmax)
## test the train error:
#zip_train_x_predict=np.dot(zip_train_x,w)
#zip_train_x_predict[zip_train_x_predict>0]=1
#zip_train_x_predict[zip_train_x_predict<=0]=-1
#sum(zip_train_x_predict==zip_train_y)/len(zip_train_y)
#
##test the testing error
#zip_test_x_predict=np.dot(zip_test_x,w)
#zip_test_x_predict[zip_test_x_predict>0]=1
#zip_test_x_predict[zip_test_x_predict<=0]=-1
#
#
#sum(zip_test_x_predict==zip_test_y)/len(zip_test_y)
#
##--------------------------------------
```

```
##%%class 3 vs rest fssip
##---------------------------------------
#zip_train_y=np.array(zip_train_small)[:,0]
#zip_train_x=np.array(zip_train_small)[:,1:]
#
#zip_test_y=np.array(zip_test_small)[:,0]
#zip_test_x=np.array(zip_test_small)[:,1:]
#zip_train_y[zip_train_y==0]=-1
#zip_train_y[zip_train_y==1]=-1
#zip_train_y[zip_train_y==2]=1
#
#train_num =len(zip_train_y)
#zip_test_y[zip_test_y==0]=-1
#zip_test_y[zip_test_y==1]=-1
#zip_test_y[zip_test_y==2]=1
#test_num =len(zip_test_y)
#
#a=[1]*zip_train_x.shape[1]
#w=pattern_recognition().fissp_fit(zip_train_x,zip_train_y,a,kmax)
## test the train error:
#zip_train_x_predict=np.dot(zip_train_x,w)
#zip_train_x_predict[zip_train_x_predict>0]=1
#zip_train_x_predict[zip_train_x_predict<=0]=-1
#sum(zip_train_x_predict==zip_train_y)/len(zip_train_y)
#
##test the testing error
#zip_test_x_predict=np.dot(zip_test_x,w)
#zip_test_x_predict[zip_test_x_predict>0]=1
#zip_test_x_predict[zip_test_x_predict<=0]=-1
#
#
#sum(zip_test_x_predict==zip_test_y)/len(zip_test_y)
#

#-----------------------------------
#Matlab Code
#-----------------------------------


%perceptron

function [w]=perceptron(sample)

sample_num=size(sample,1);

feature=size(sample,2);

w=ones(1,feature);

flag=1;

count=1;

while count<=10
```

```
    flag=0;

    for k=1:sample_num

        m=sample(k,:)*w';

        if  m<=0

            w=w+sample(k,:);

            flag=1;

        end

    end

    count=count+1;

end

disp(count);

%zip_train_0_2

D=importdata('wine_uci_train.txt');

num=size(D,1);

D1=D;

D2=D;

D3=D;

for i=1:num

    if D1(i,1)==1

       D1(i,1)=1;

    else

       D1(i,1)=1;

       D1(i,:)=-D1(i,:);

    end


    if D2(i,1)==2
```

```
        D2(i,1)=1;
    else
        D2(i,1)=1;
        D2(i,:)=-D2(i,:);
    end

    if D3(i,1)==3
        D3(i,1)=1;
    else
        D3(i,1)=1;
        D3(i,:)=-D3(i,:);
    end
end
T=importdata('wine_uci_test.txt');
numt=size(T,1);
T1=T;
T2=T;
T3=T;
numc1=0;
numc2=0;
numc3=0;
for i=1:numt
    if T1(i,1)==1
        T1(i,1)=1;
    else
        T1(i,1)=1;
        T1(i,:)=-T1(i,:);
```

```
    end

    if T2(i,1)==2
       T2(i,1)=1;
    else
       T2(i,1)=1;
       T2(i,:)=-T2(i,:);
    end

    if T3(i,1)==3
       T3(i,1)=1;
    else
       T3(i,1)=1;
       T3(i,:)=-T3(i,:);
    end
end

% batch_relaxation
function [w]=batch_relaxation(sample)
b0=1;
eta=0.1;
sample_num=size(sample,1);
feature=size(sample,2);
w=ones(1,feature);
flag=1;
count=1;
```

```
while count<=10000
    flag=0;
    sk=[];
    for k=1:sample_num
        m=sample(k,:)*w';
        if m<=b0
            flag=1;
            yk=((b0-m)/(norm(sample(k,:))^2)).*sample(k,:);
            sk=[sk;yk];
        end
    end
    if (isempty(sk))
        break;
    end
    w=w+eta*sum(sk);
    count=count+1;
end
disp(count);
%perceptron two-class
%training
w1=perceptron(D1);
w2=perceptron(D2);
w3=perceptron(D3);
%testing
w=w2;
Tk=T;
count=0;
```

```
for i=1:numt
    Tk(i,1)=1;
    m=Tk(i,:)*w';

    if m>=0
       Tk(i,1)=1;
    else
       Tk(i,1)=-1;
    end

    if T2(i,1)==Tk(i,1)
       count=count+1;
    end
end
accuracy=count/numt;
% batch_relaxation two-class
D=importdata('wine_uci_train.txt');
num=size(D,1);
D1=D;
D2=D;
D3=D;
for i=1:num
    if D1(i,1)==1
       D1(i,1)=1;
    else
       D1(i,1)=1;
```

```
        D1(i,:)=-D1(i,:);

    end


    if D2(i,1)==2

        D2(i,1)=1;

    else

        D2(i,1)=1;

        D2(i,:)=-D2(i,:);

    end


    if D3(i,1)==3

        D3(i,1)=1;

    else

        D3(i,1)=1;

        D3(i,:)=-D3(i,:);

    end

end

T=importdata('wine_uci_test.txt');

numt=size(T,1);

T1=T;

T2=T;

T3=T;

numc1=0;

numc2=0;

numc3=0;

for i=1:numt

    if T1(i,1)==1
```

```
        T1(i,1)=1;
    else
        T1(i,1)=1;
        T1(i,:)=-T1(i,:);
    end

    if T2(i,1)==2
        T2(i,1)=1;
    else
        T2(i,1)=1;
        T2(i,:)=-T2(i,:);
    end

    if T3(i,1)==3
        T3(i,1)=1;
    else
        T3(i,1)=1;
        T3(i,:)=-T3(i,:);
    end
end

%training
w1=batch_relaxation(D1);
w2=batch_relaxation(D2);
w3=batch_relaxation(D3);
%testing
```

```
w=w1;
Tk=T;
count=0;
for i=1:numt
    Tk(i,1)=1;
    m=Tk(i,:)*w';

    if m>=0
       Tk(i,1)=1;
    else
       Tk(i,1)=-1;
    end

    if T1(i,1)==Tk(i,1)
       count=count+1;
    end
end
accuracy=count/numt;
D=importdata('wine_uci_train.txt');
%one against rest
num=size(D,1);
D1=D;
D2=D;
D3=D;
for i=1:num
    if D1(i,1)==1
       D1(i,1)=1;
```

```
    else
       D1(i,1)=1;
       D1(i,:)=-D1(i,:);
    end

    if D2(i,1)==2
       D2(i,1)=1;
    else
       D2(i,1)=1;
       D2(i,:)=-D2(i,:);
    end

    if D3(i,1)==3
       D3(i,1)=1;
    else
       D3(i,1)=1;
       D3(i,:)=-D3(i,:);
    end
end
T=importdata('wine_uci_test.txt');
numt=size(T,1);
T1=T;
T2=T;
T3=T;
for i=1:numt
    if T1(i,1)==1
```

```
        T1(i,1)=1;
    else
        T1(i,1)=1;
        T1(i,:)=-T1(i,:);
    end

    if T2(i,1)==2
        T2(i,1)=1;
    else
        T2(i,1)=1;
        T2(i,:)=-T2(i,:);
    end

    if T3(i,1)==3
        T3(i,1)=1;
    else
        T3(i,1)=1;
        T3(i,:)=-T3(i,:);
    end
end

tic
%training
w1=batch_relaxation(D1);
w2=batch_relaxation(D2);
w3=batch_relaxation(D3);
Tk=T;
```

```
count=0;
for i=1:numt
    Tk(i,1)=1;
    g1=Tk(i,:)*w1';
    g2=Tk(i,:)*w2';
    g3=Tk(i,:)*w3';
    g=max([g1,g2,g3]);
    if g==g1
       Tk(i,1)=0;
    elseif g==g2
       Tk(i,1)=1;
    elseif g==g3
       Tk(i,1)=2;
    end
    if Tk(i,1)==T(i,1)
        count=count+1;
    end
end
accuracy=count/numt;
toc
one against other
D=importdata('wine_uci_train.txt');
num=size(D,1);
D1=D;
D2=D;
D3=D;
```

```
Dk1=[];

Dk2=[];

Dk3=[];

for i=1:num

    if D1(i,1)==1

       D1(i,1)=1;

       Dk1=[Dk1;D1(i,:)];

    elseif D1(i,1)==2

       D1(i,1)=1;

       D1(i,:)=-D1(i,:);

       Dk1=[Dk1;D1(i,:)];

    end



    if D2(i,1)==1

       D2(i,1)=1;

       Dk2=[Dk2;D2(i,:)];

    elseif D2(i,1)==3

       D2(i,1)=1;

       D2(i,:)=-D2(i,:);

       Dk2=[Dk2;D2(i,:)];

    end


    if D3(i,1)==2

       D3(i,1)=1;

       Dk3=[Dk3;D3(i,:)];

    elseif D3(i,1)==3
```

```
        D3(i,1)=1;

        D3(i,:)=-D3(i,:);

        Dk3=[Dk3;D3(i,:)];

    end



end

%T=importdata('zip_test_0_2_small.txt');

T=importdata('wine_uci_test.txt');

tic

w1=batch_relaxation(Dk1);

w2=batch_relaxation(Dk2);

w3=batch_relaxation(Dk3);

numt=size(T,1);

Tk=T;

count=0;

for i=1:numt

    m1=T(i,:)*w1';

    if m1>=0

        c1=1;

    else

        c1=2;

    end



    m2=T(i,:)*w2';

    if m2>=0

        c2=1;
```

```
    else
        c2=3;
    end


    m3=T(i,:)*w3';
    if m3>=0
        c3=2;
    else
        c3=3;
    end


    if (c1~=c2)&&(c1~=c3)&&(c2~=c3)
        c=4;
    else
        A=[c1 c2 c3];
        c=mode(A);
    end
    Tk(i,1)=c;
    if c==T(i,1)
        count=count+1;
    end
end
accuracy=count/numt;
toc
```

Figure 2: Analysis of the USPS data set

USPS dataset	Two Class			Multi Class	
Perceptron	class1-agianst-rest	class2-agianst-rest	class3-against-rest	one-against-rest	one-against-other
Accuracy	0.9833	0.98	0.9667	0.9733	0.9667
Iteration	6	3	11	a1:6 a2:2 a3:11	a1:3 a2:6 a3:5
CPU time(s)				0.0071	0.069
Batch	class1-agianst-rest	class2-agianst-rest	class3-against-rest	one-against-rest	one-against-other
Accuracy	0.9867	0.9733	0.9533	0.94	0.9467
Iteration	100000	7	100000	a1:100000 a2:7 a3:100000	a1:100000 a2:7 a3:100000
CPU time(s)				16.15	14.67

Figure 3: Results for UCI data without ada boosting

	Fixed Increment Single Sample Perceptron			Batch relaxation with Margin		
	Class 1 vs Class 2	Class 1vs Class 3	Class 2 vs Class 3	Class 1 vs Class 2	Class 1vs Class 3	Class 2 vs Class 3
Accuracy Rate	90.63%	86.54%	60.00%	56.25%	53.85%	60.00%
Iteration number	151	151	151	151	151	151
CPU time(s)	0.042	0.029	0.039	0.072	0.059	0.063

Figure 4: Results for UCI data with ada boosting

	Fixed Increment Single Sample Perceptron			Batch relaxation with Margin		
	Class 1 vs Class 2	Class 1vs Class 3	Class 2 vs Class 3	Class 1 vs Class 2	Class 1vs Class 3	Class 2 vs Class 3
Accuracy Rate	95.31%	94.23%	95.00%	93.75%	82.69%	65.00%
Iteration number	201	201	201	201	201	201
CPU time	2.717	2.213	2.467	8.113	6.564	5.375

Figure 5: Results for USPS data without ada boosting

	Fixed Increment Single Sample Perceptron			Batch relaxation with Margin		
	Class 1 vs Class 2	Class 1vs Class 3	Class 2 vs Class 3	Class 1 vs Class 2	Class 1vs Class 3	Class 2 vs Class 3
Accuracy Rate	97.78%	97.80%	96.34%	99.11%	97.80%	95.29%
Iteration number	2	6	3	201	201	6
CPU time	0.021	0.022	0.021	0.069	0.073	0.026

Figure 6: Results for USPS data with ada boosting

	Fixed Increment Single Sample Perceptron			Batch relaxation with Margin		
	Class 1 vs Class 2	Class 1vs Class 3	Class 2 vs Class 3	Class 1 vs Class 2	Class 1vs Class 3	Class 2 vs Class 3
Accuracy Rate	99.11%	97.80%	97.38%	98.22%	95.05%	93.72%
Iteration number	3	8	6	4	5	4
CPU time	0.034	0.051	0.035	0.052	0.087	0.050