HW6 report　0410024 余東儒

Code explanation:

Part 1---Function1: Readfile

X_name: X_test or X_train path

T_name:T_test or T_train path

Size: number of data

Mode: output file name will be train.txt or test.txt,it's for LIBSVM format file

label: store all labels from T file(2500,5000)

data: store all data from X file(784*2500,784*5000)

x_mean: store each pixel's average ,used for caculating covariance in PCA

outer loop:

1. Read file line by line
2. **line** for X_file;**tm** for T_file
3. **write_string** for writing a line in LIBSVM format file
4. **x**: matrix for storing one data
5. innerloop: Pick every pixel's value out and dividing **size** storing into **x_mean**. Also, storing into write_string
6. write into outputfile; storing x into data

```
def read_file(X_name,T_name,mode,size):
    X=open(X_name,"r")
    T=open(T_name,"r")
    f=open(mode,"w")
    label=[]
    data=[]
    x_mean=np.zeros((784,1), dtype=np.float)
    for i in range(size):
        line=X.readline()
        tm=int(T.readline())
        label.append(tm)
        write_string=""
        write_string+=(str(tm)+' ')
        x=np.zeros((784,1), dtype=np.float)
        e=0
        for j in range(783):
            end=line.find(',',e)
            x[j]=float(line[e:end])
            x_mean[j]+=(x[j]/size)
            write_string+=(str(j)+":"+line[e:end]+" ")
            e=end+1
        end=line.find('\n')
        write_string+=("783:"+line[e:end]+"\n")
        x[783]=float(line[e:end])
        f.write(write_string)
        data.append(x)
    f.close()
    return label,data,x_mean
```

Part2---Function2:PCA

Main purpose: calculating W martrix and projecting traing data to 2D space

Arguments: results from Readfile function,using train_data and x_mean

First loop and second loop: Calculating covariance matrix

Covariance to W matrix:

1. Dealing with eigen problem by eig library.
2. Eigvector needs to transpose because original form will be col base,trasposing to row base will be more easy to get
3. Sorting eigenvalue by argsort()
4. Picking first two largest eigenvalue and corresponding eigenvector and put them to **w**

Last loop:

Using **w** matrix to project **train_data** to **z**(2d sapce)

```
def pca(train_data,x_mean):
    covariance=np.zeros((784,784), dtype=np.float)

    for i in range(size):
        tmp=train_data[i]-x_mean
        covariance+=np.dot(tmp,tmp.transpose())

    for i in range(784):
        for j in range(784):
            covariance[i][j]/=size
    w=np.zeros((2,784), dtype=np.float)
    eigvalue,eigvector=eig(covariance)
    eigvec=eigvector.transpose()


    idx = eigvalue.argsort()

    #eigvector=eigvector.transpose()
    eigvalue = eigvalue[idx]
    for i in range(2):
        w[i]=eigvec[idx[783-i]]


    z=[]
    for i in range(size):
        z.append((np.dot(w,train_data[i])).transpose())
    return z,w
```

Main part---training and testing:

1. Read test and train file first
2. Using svm_read_problem to get label and data from LIBSVM form file
3. Using training data and label to train model used by svm_train.Here set option to C=50
4. Using testing data and label to test model used by svm_predict
5. p_acc will be the Accuracy rate

```
size=int(input("size:"))
train_label,train_data,x_mean=read_file("X_train.csv","T_train.csv","train.txt",size)


test_label,test_data,xx_mean=read_file("X_test.csv","T_test.csv","test.txt",int(size/2))
y, x = svm_read_problem("train.txt")
yy,xx=svm_read_problem("test.txt")
m = svm_train(y,x,'-c 50')
p_label, p_acc, p_val = svm_predict(yy,xx, m)
```

Ploting part--- PCA and plot it!!!

Do the PCA first and get z and w

1.Plot training data

First loop: plot all the training_data from z depend on which label it is

a: get the indices where the support vectors are by get_sv_indices

Second loop: Plot all the support vectors with special symbols

```
z,w=pca(train_data,x_mean)

for i in range(size):
    if train_label[i]==1:
        plt.plot(z[i][0][0],z[i][0][1],'ro')
    if train_label[i]==2:
        plt.plot(z[i][0][0],z[i][0][1],'bo')
    if train_label[i]==3:
        plt.plot(z[i][0][0],z[i][0][1],'go')
    if train_label[i]==4:
        plt.plot(z[i][0][0],z[i][0][1],'co')
    if train_label[i]==5:
        plt.plot(z[i][0][0],z[i][0][1],'yo')
a=m.get_sv_indices()
for n in a:
    i=n-1
    if train_label[i]==1:
        plt.plot(z[i][0][0],z[i][0][1],'r*')
    if train_label[i]==2:
        plt.plot(z[i][0][0],z[i][0][1],'bs')
    if train_label[i]==3:
        plt.plot(z[i][0][0],z[i][0][1],'gp')
    if train_label[i]==4:
        plt.plot(z[i][0][0],z[i][0][1],'c+')
    if train_label[i]==5:
        plt.plot(z[i][0][0],z[i][0][1],'yx')
plt.savefig("SVM.png")
plt.show()
```

2.Plot testing data

First loop:

zz: storing testing data’s projecting data

use w to get projecting data

Second loop:

Plot all testing data

Third loop:

Plot all support vectors

```
zz=[]
for i in range(int(size/2)):
    zz.append((np.dot(w,test_data[i])).transpose())

for i in range(int(size/2)):
    if p_label[i]==1:
        plt.plot(zz[i][0][0],zz[i][0][1],'ro')
    if p_label[i]==2:
        plt.plot(zz[i][0][0],zz[i][0][1],'bo')
    if p_label[i]==3:
        plt.plot(zz[i][0][0],zz[i][0][1],'go')
    if p_label[i]==4:
        plt.plot(zz[i][0][0],zz[i][0][1],'co')
    if p_label[i]==5:
        plt.plot(zz[i][0][0],zz[i][0][1],'yo')
for n in a:
    i=n-1
    if train_label[i]==1:
        plt.plot(z[i][0][0],z[i][0][1],'r*')
    if train_label[i]==2:
        plt.plot(z[i][0][0],z[i][0][1],'bs')
    if train_label[i]==3:
        plt.plot(z[i][0][0],z[i][0][1],'gp')
    if train_label[i]==4:
        plt.plot(z[i][0][0],z[i][0][1],'c+')
    if train_label[i]==5:
        plt.plot(z[i][0][0],z[i][0][1],'yx')

plt.savefig("test_original.jpg")
plt.show()
```

Performance:

In previous code, I use C=50 and default RBF. The accuracy rate is 96.96% (2423/2500).

Some discussion on performance (C,gamma) (For bonus):

Chart1:C=1~491 1/gamma=1~4901

C,1/gamma	1	701	1401	2101	2801	3501	4201	4901
1	30.04	95.32	94.88	94.64	94.44	94.16	93.96	93.84
71	31.32	97.12	96.48	96.24	96.24	96.2	96.16	96.12
141	31.32	97.04	96.56	96.44	96.24	96.08	96.08	96.12
211	31.32	97	96.44	96.36	96.32	96.24	96.08	96.08
281	31.32	96.96	96.4	96.32	96.2	96.12	96.12	96.04
351	31.32	97	96.32	96.2	96.2	96.2	96	96.08
421	31.32	97	96.32	96.08	96.12	96.2	96.24	96
491	31.32	97	96.28	96.08	96.16	96.12	96.2	96.16

So according to Chart1, we can limit the 1/gamma range below 700. C doesn't influence much at results.

Chart2:

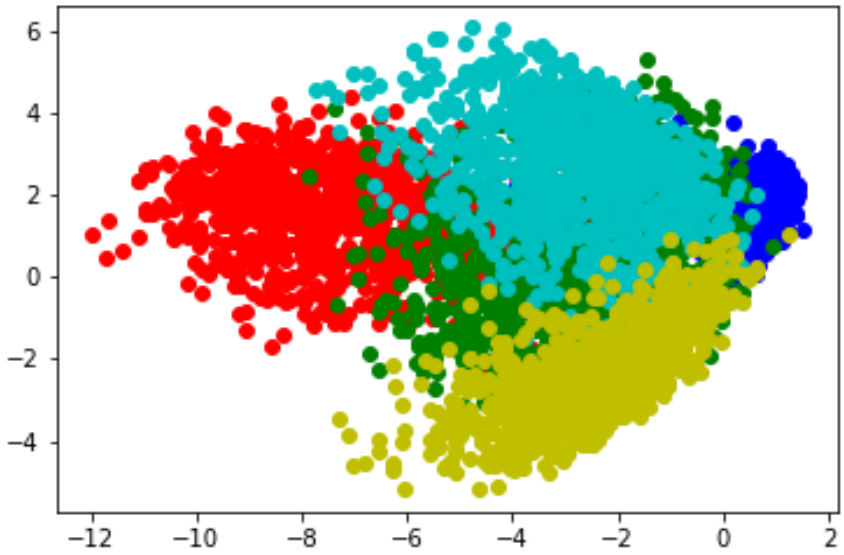
C,1/gamma	5	15	25	35	45	55	65	75
5	69.36	96.96	98.44	98.6	98.52	98.48	98.4	98.28
200	69.36	96.96	98.44	98.6	98.52	98.48	98.4	98.28
400	69.36	96.96	98.44	98.6	98.52	98.48	98.4	98.28
600	69.36	96.96	98.44	98.6	98.52	98.48	98.4	98.28
800	69.36	96.96	98.44	98.6	98.52	98.48	98.4	98.28
1000	69.36	96.96	98.44	98.6	98.52	98.48	98.4	98.28

According to Chart2, results has a peak at 1/gamma=35.

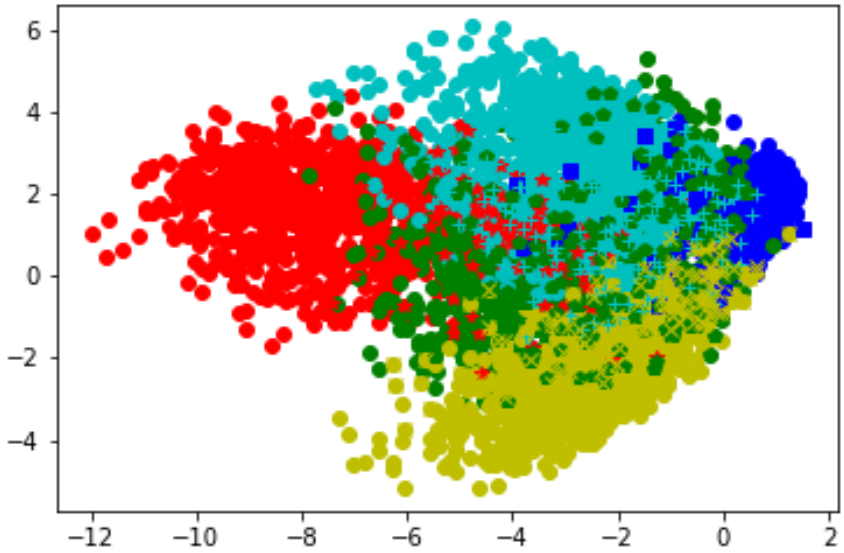
It's how I get previous charts.

```
y, x = svm_read_problem("train.txt")
yy,xx=svm_read_problem("test.txt")
#tmp="-c "+str(45)+" -g "+str(1/10)
#print("C="+str(50)+"    gamma="+str(1/700))
#m = svm_train(y,x,tmp)
#p_label, p_acc, p_val = svm_predict(yy,xx, m)
for i in range(200,1001,200):
    for j in range(5,80,10):
        tmp="-c "+str(i)+" -g "+str(1/j)
        print("C="+str(i)+"    gamma="+str(1/j))
        m = svm_train(y,x,tmp)
        p_label, p_acc, p_val = svm_predict(yy,xx, m)
```

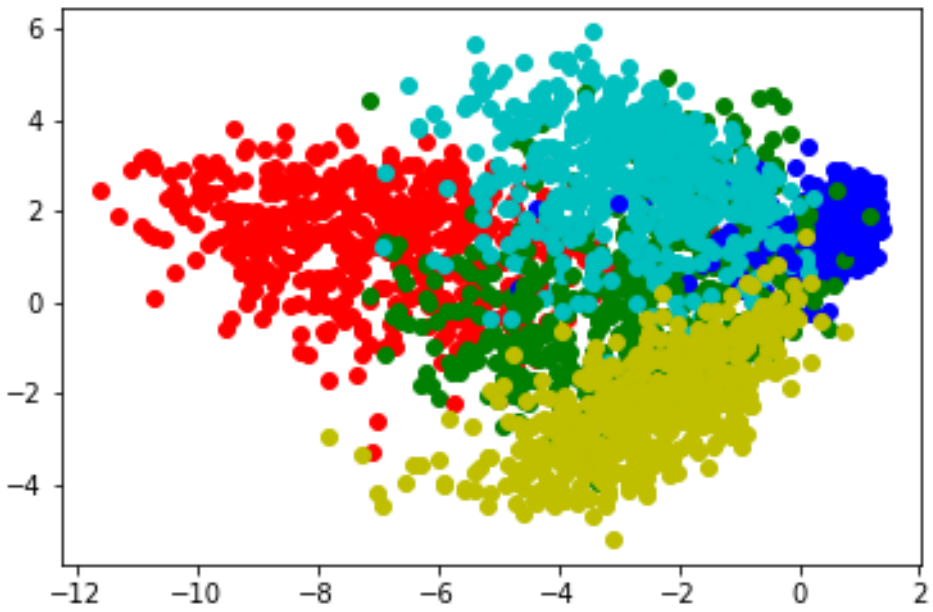
PLOT1:Training data on 2D space



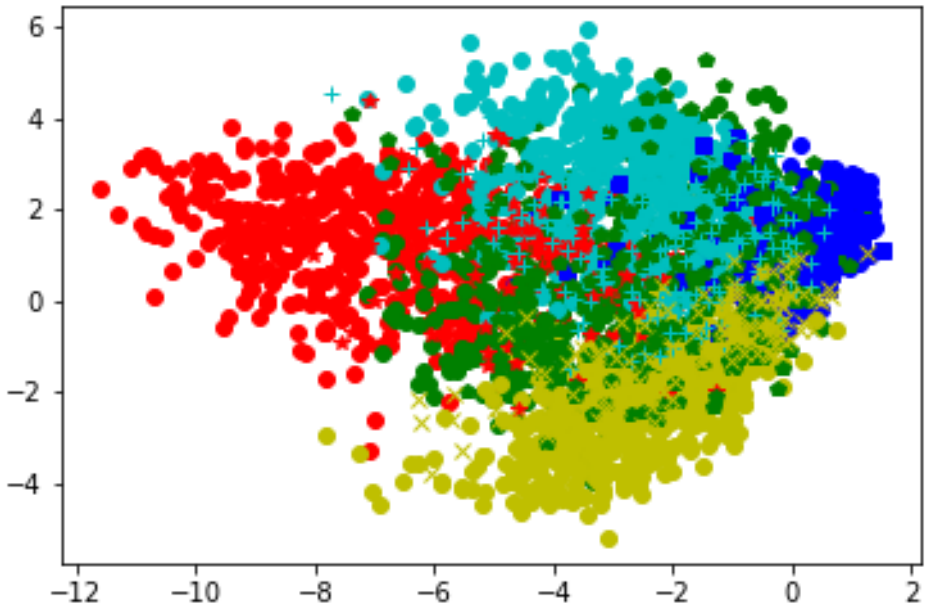
PLOT2:Training data on 2D space with support vectors



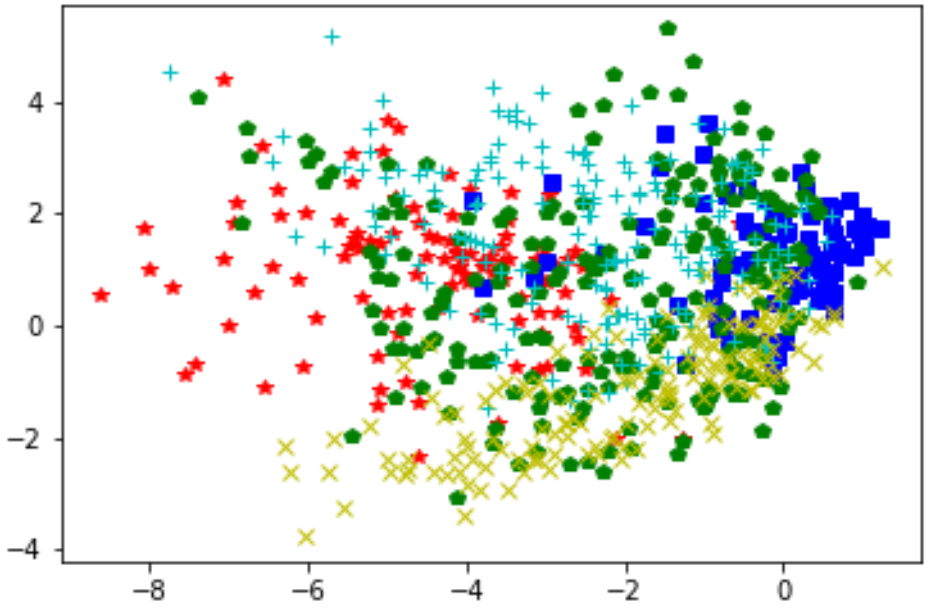
PLOT3:Testing data on 2D space



PLO4:TTesting data on 2D space with support vectors



PLOT5:Support vetors



心得:

之前學的都是上課內容交的公式觀念等等，很難想像這些理論真的能夠做到這些分類，這次把 PCA 跟 SVM 實做出來也讓我對這些數學 model 更加熟悉。

從 PCA 投影出的圖看的出來，果然同一個 class 的 data 會聚集在同一區，不過 SVM 因為有五個 class，並且從 784d-2d，比較難看得出 gap。