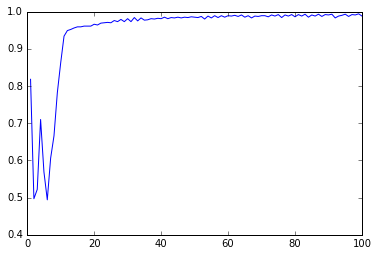
**Homework 3**

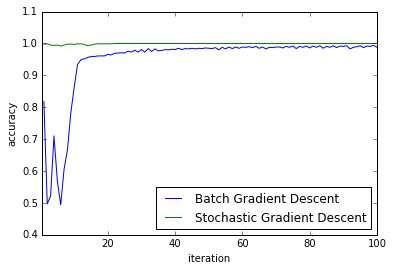
**1(d)**

The accuracy plot is the following:



**1(f)**

The accuracy plot is the following:

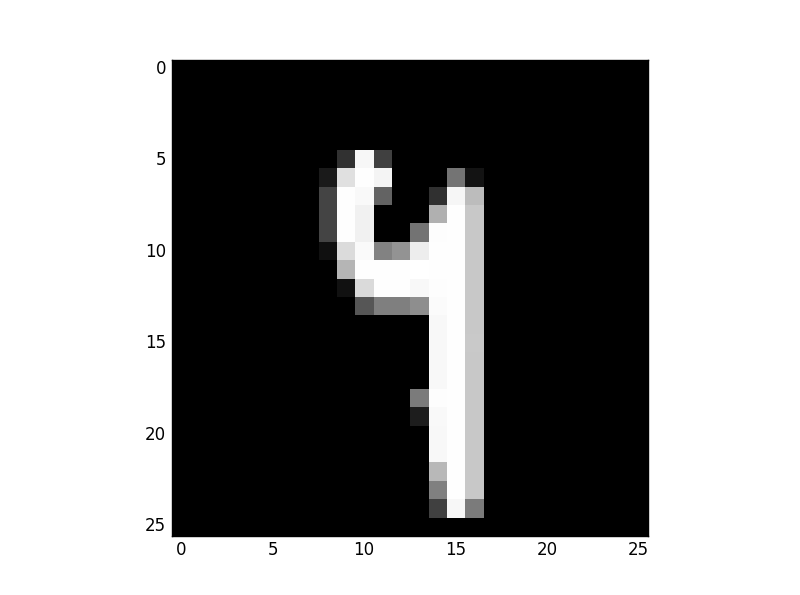


**1(g)**

The stochastic gradient descent converges faster than batch gradient decent. From the plot in 1(f), we can see the accuracy of stochastic gradient descent converges to 1 after approximate 30 iterations, while the batch gradient decent still has a little fluctuation. It is mainly because in the stochastic gradient descent, in each iteration, w & b are updated N times in 1 iteration. In batch gradient decent, w & b remain the same in 1 iteration.

**1(i)**

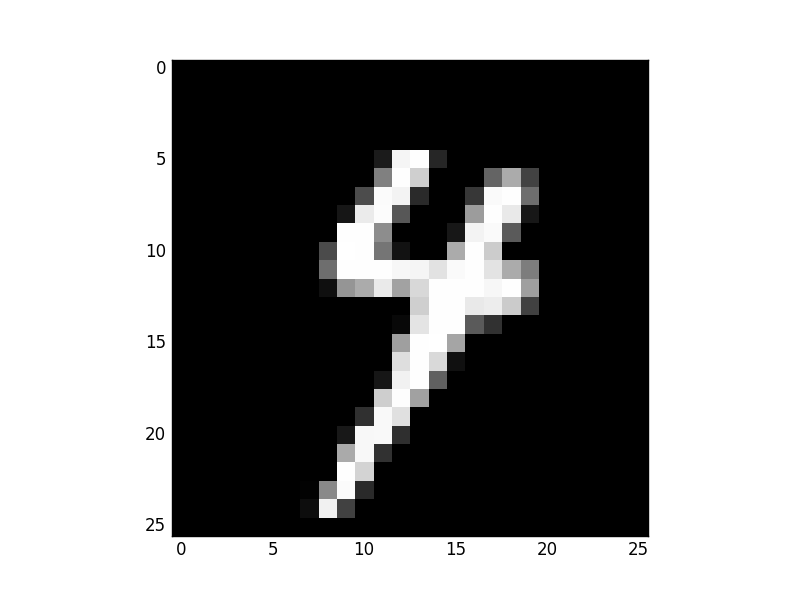
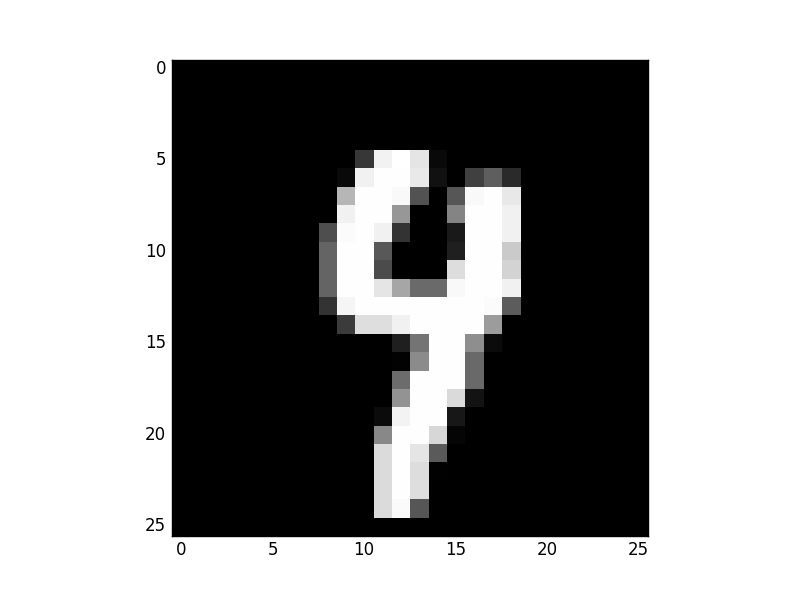
I choose Gamma=10^(-7), C=10 in this problem. The accuracy of training data and test data is 99.9% and 99.8%. Therefore, there is only 1 misclassified test image. It is the 164th of the test labels, it is “9” initially, but the SVM labels it “4”. The picture is shown as follows:

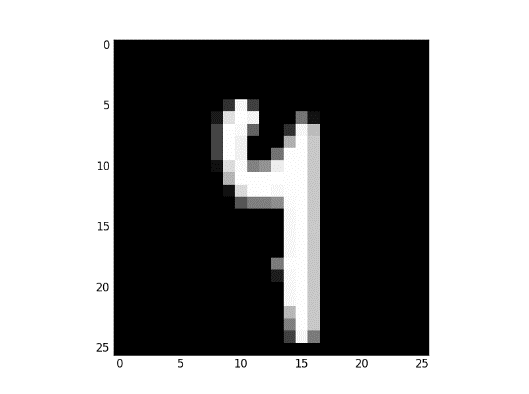
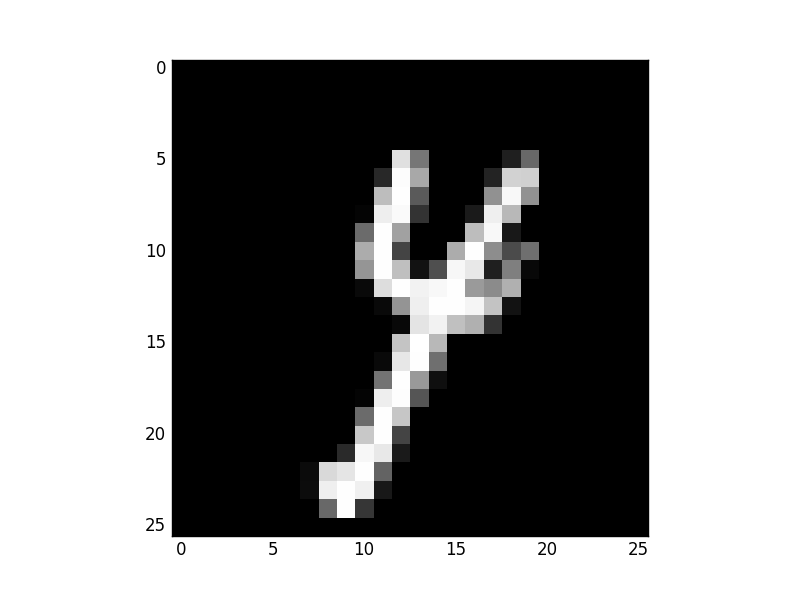
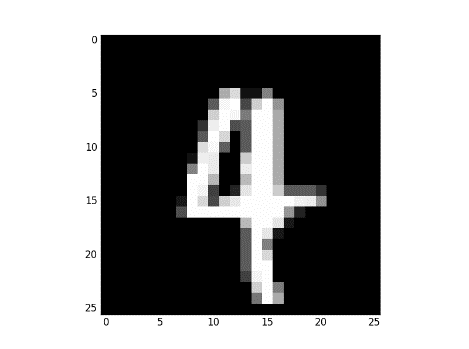


**1(j)**

Based on the separate final accuracy, there is no significant difference between LDA and SVM.

The The accuracy of training data and test data is 99.7% and 90.4%. 5 of the misclassified test images are :



The first and the third are misclassified as “4” and the others are misclassified “9”.

**4b**

The RMSE of the models are as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| RMSE | 7.412 | 4.831 | 2.962 | 13.199 |

Codes:

(1d)

import numpy as np

trainx = np.loadtxt("digits\_training\_data.csv",delimiter=",")

trainy=np.loadtxt("digits\_training\_labels.csv",delimiter=",")

w=np.zeros(676);b=0;

iterN=np.linspace(1,100,100);

for i in range(1000):

if trainy[i]==4:

trainy[i]=1;

else:

trainy[i]=-1;

from \_\_future\_\_ import division

def Ewb\_grad(w,b):

Ew=np.zeros(676)

Eb=np.zeros((1000,1))

for i in range(1000):

if trainy[i]\*(trainx[i,:].dot(w)+b)<1:

Ew = Ew - trainy[i]\*(trainx[i,:].T)

Eb[i] = -3\*trainy[i]

else:

Ew=Ew

Eb[i]=0

Ew\_grad=Ew+w

Eb\_grad=np.sum(Eb)

return Ew\_grad, Eb\_grad;

w0=w;b0=b;goal1=np.zeros(100)

for j in range(100):

yita=0.001

wgrad,bgrad=Ewb\_grad(w0,b0)

alpha=yita/(1+(j+1)\*yita)

w0=w0-alpha\*wgrad

b0=b0-alpha\*bgrad

yfit=trainy\*(trainx.dot(w0)+b0)

cali=np.zeros(1000)

goal1[j]=np.sum(yfit>cali)/1000

from matplotlib import pyplot as plt

plt.plot(iterN,goal1)

(1f)

import numpy as np

trainx = np.loadtxt("digits\_training\_data.csv",delimiter=",")

trainy=np.loadtxt("digits\_training\_labels.csv",delimiter=",")

w=np.zeros(676);b=0;

iterN=np.linspace(1,100,100);

for i in range(1000):

if trainy[i]==4:

trainy[i]=1;

else:

trainy[i]=-1;

from \_\_future\_\_ import division

def wb\_grad(w,b,j):

Ew=np.zeros(676);

w1=w;b1=b; yita=0.001;

alpha=yita/(1+(j+1)\*yita);

r=np.random.permutation(1000)

for i in r:

if trainy[i]\*(trainx[i,:].dot(w)+b)<1:

Ew = 0.001\*w1 - 3\*trainy[i]\*(trainx[i,:].T)

Eb = -3\*trainy[i]

w1=w1-alpha\*Ew

b1=b1-alpha\*Eb

else:

Ew=0.001\*w1

Eb=0

w1=w1-alpha\*Ew

b1=b1-alpha\*Eb

return w1, b1;

w0=w;b0=b;goal=np.zeros(100)

for j in range(100):

w0,b0=wb\_grad(w0,b0,j)

yfit=trainy\*(trainx.dot(w0)+b0)

cali=np.zeros(1000)

goal[j]=np.sum(yfit>cali)/1000

from matplotlib import pyplot as plt

plt.plot(iterN,goal)

SVM1, =plt.plot(iterN,goal1,label='Batch Gradient Descent')

SVM2, =plt.plot(iterN,goal,label='Stochastic Gradient Descent')

plt.legend(handles=[SVM1,SVM2],loc='lower right')

plt.xlabel('iteration')

plt.ylabel('accuracy')

plt.axis([0.5,100,0.5,1.1])

(1i)

import numpy as np

from sklearn import svm

trainx = np.loadtxt("digits\_training\_data.csv",delimiter=",")

trainy=np.loadtxt("digits\_training\_labels.csv",delimiter=",")

testx= np.loadtxt("digits\_test\_data.csv",delimiter=",")

testy=np.loadtxt("digits\_test\_labels.csv",delimiter=",")

for i in range(500):

if testy[i]==4:

testy[i]=1;

else:

testy[i]=-1;

for i in range(1000):

if trainy[i]==4:

trainy[i]=1;

else:

trainy[i]=-1;

from \_\_future\_\_ import division

clf = svm.SVC(kernel='rbf',gamma=10\*\*(-7), C=10)

clf.fit(trainx, trainy)

z=clf.predict(trainx)

goal=np.sum(z==trainy)/1000

clf = svm.SVC(kernel='rbf',gamma=10\*\*(-7), C=10)

clf.fit(testx, testy)

z=clf.predict(testx)

goal1=np.sum(z==testy)/500

for i in range(500):

if z[i]\*testy[i]<0:

k=i

import matplotlib.cm as cm

from matplotlib import pyplot as plt

plt.imshow(testx[k,:].reshape((26,26)), interpolation="nearest", cmap=cm.Greys\_r)

(1j)

import numpy as np

trainx = np.loadtxt("digits\_training\_data.csv",delimiter=",")

trainy=np.loadtxt("digits\_training\_labels.csv",delimiter=",")

testx = np.loadtxt("digits\_test\_data.csv",delimiter=",")

testy=np.loadtxt("digits\_test\_labels.csv",delimiter=",")

for i in range(500):

if testy[i]==4:

testy[i]=1;

else:

testy[i]=-1;

for i in range(1000):

if trainy[i]==4:

trainy[i]=1;

else:

trainy[i]=-1;

from \_\_future\_\_ import division

trainx1=trainx[(trainy==-1),]

trainx2=trainx[(trainy==1),]

mean1=np.reshape(np.mean(trainx1,0),(1,len(trainx1[0])))

mean2=np.reshape(np.mean(trainx2,0),(1,len(trainx2[0])))

trainx1norm=trainx1-mean1;

trainx2norm=trainx2-mean2;

trainxnorm=np.concatenate((trainx1norm,trainx2norm))

sigma=(trainxnorm.T.dot(trainxnorm))/1000

gamma1 = -0.5\*mean1.dot(np.linalg.pinv(sigma)).dot(mean1.T)+np.log(1.0\*len(trainx1)/len(trainx))

beta1 = np.linalg.pinv(sigma).dot(mean1.T)

gamma2 = -0.5\*mean2.dot(np.linalg.pinv(sigma)).dot(mean2.T)+np.log(1.0\*len(trainx2)/len(trainx))

beta2 = np.linalg.pinv(sigma).dot(mean2.T)

trainp1 = np.exp((trainx.dot(beta1)+gamma1))/(np.exp((trainx.dot(beta1)+gamma1))+np.exp((trainx.dot(beta2)+gamma2)))

trainp2 = np.exp((trainx.dot(beta2)+gamma2))/(np.exp((trainx.dot(beta1)+gamma1))+np.exp((trainx.dot(beta2)+gamma2)))

trainyfit=np.zeros((1000))

for i in range(1000):

if trainp1[i]>=trainp2[i]:

trainyfit[i]=-1

else:

trainyfit[i]=1

goal=np.sum(trainyfit==trainy)/1000

testp1 = np.exp((testx.dot(beta1)+gamma1))/(np.exp((testx.dot(beta1)+gamma1))+np.exp((testx.dot(beta2)+gamma2)))

testp2 = np.exp((testx.dot(beta2)+gamma2))/(np.exp((testx.dot(beta1)+gamma1))+np.exp((testx.dot(beta2)+gamma2)))

testyfit=np.zeros((500))

for i in range(500):

if testp1[i]>=testp2[i]:

testyfit[i]=-1

else:

testyfit[i]=1

goal1=np.sum(testyfit==testy)/500

m=np.zeros((500))

for i in range(500):

if testyfit[i]\*testy[i]<0:

m[i]=i

k=m[m>0]

import matplotlib.cm as cm

from matplotlib import pyplot as plt

for i in range(5):

plt.figure()

plt.imshow(testx[k[i],:].reshape((26,26)), interpolation="nearest", cmap=cm.Greys\_r)

(2)

import numpy as np

from sklearn import svm

trainy = np.loadtxt('trainingLabels.gz', dtype=np.uint8, delimiter=',')

trainx = np.loadtxt('trainingData.gz', dtype=np.uint8, delimiter=',')

testx = np.loadtxt('testData.gz', dtype=np.uint8, delimiter=',')

from \_\_future\_\_ import division

clf = svm.SVC(kernel='rbf',gamma=10\*\*(-8), C=10)

clf.fit(trainx, trainy)

z=clf.predict(testx)

(4b)

import numpy as np

train = np.loadtxt("steel\_composition\_train.csv", delimiter=",", skiprows=1, usecols=(1,2,3,4,5,6,7,8,9))

x1=train[:,0:8]

y1=train[:,8]

x1 = (x1-np.mean(x1,0))/np.std(x1,0)

y1 = np.reshape(y1,((len(y1),1)))

K1 = (x1.dot(x1.T)+1)\*\*2

a1 = np.linalg.inv(np.identity(len(x1))+K1).dot(y1)

Err1 = a1.T.dot(K1).dot(K1).dot(a1)-2\*y1.T.dot(K1).dot(a1)+y1.T.dot(y1)+a1.T.dot(K1).dot(a1)

rmse1 = np.sqrt(Err1/len(x1))

K2 = (x1.dot(x1.T)+1)\*\*3

a2 = np.linalg.inv(np.identity(len(x1))+K2).dot(y1)

Err2= a2.T.dot(K2).dot(K2).dot(a2)-2\*y1.T.dot(K2).dot(a2)+y1.T.dot(y1)+a2.T.dot(K2).dot(a2)

rmse2 = np.sqrt(Err2/len(x1))

K3 = (x1.dot(x1.T)+1)\*\*4

a3 = np.linalg.inv(np.identity(len(x1))+K3).dot(y1)

Err3= a3.T.dot(K3).dot(K3).dot(a3)-2\*y1.T.dot(K3).dot(a3)+y1.T.dot(y1)+a3.T.dot(K3).dot(a3)

rmse3 = np.sqrt(Err3/len(x1))

from numpy.linalg import norm

K4 = []

for i in range(0,len(x1)):

K4.append(np.exp(-0.5\*norm(x1[i]-x1,axis=1)\*\*2))

K4 = np.array(K4)

a4 = np.linalg.inv(np.identity(len(x1))+K4).dot(y1)

Err4 = a4.T.dot(K4).dot(K4).dot(a4)-2\*y1.T.dot(K4).dot(a4)+y1.T.dot(y1)+a4.T.dot(K4).dot(a4)

rmse4 = np.sqrt(Err4/len(x1))