Worked with: Steve Novakov

1 Problem 1

We are given the soft margin SVM, which is equivalent to the following minimization problem:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{s.t} \quad & t^{(i)}(w^Tx^{(i)}+b) \geq 1-\xi_i \\ & \xi_i \geq 0 \quad (i=1,\dots,\text{N}) \end{aligned} \tag{1}$$

Which is equivalent to

$$\min_{w,b} \; \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\max\Big(0, 1-t^{(-1)}(w^Tx^{(-1)}+b)\Big) \tag{2}$$

1.1 Part a

For this section we are asked to prove that (1) and (2) are equivalent. To prove this, the first thing we note, by the KKT complementary slackness conditions, is that, when the dual is taken, either

$$t^{(i)}(w^Tx^{(i)}+b) = 1-\xi_i \quad \text{or} \quad 1-t^{(i)}(w^Tx^{(i)}+b) \leq 0 \tag{3}$$

With this in hand, we consider the optimization problem in (1) and the following two situations that could occur on $1-t^{(i)}(w^Tx^{(i)}+b)$

- $1-t^{(i)}(w^Tx^{(i)}+b) \leq 0 \Rightarrow \xi_i = 0$
- $1-t^{(i)}(w^Tx^{(i)}+b) > 0 \Rightarrow \xi_i = 1-t^{(i)}(w^Tx^{(i)}+b) > 0$

Therefore, with these two scenarios, it is easy to see that the ξ_i in the sum of optimization problem (1) will either be zero or $1-t^{(i)}(w^Tx^{(i)}+b) > 0$. Another way to formulate this is that

$$\xi_i = \max\Big(0, 1-t^{(-1)}(w^Tx^{(-1)}+b)\Big) \tag{4}$$

Since if $1-t^{(-1)}(w^Tx^{(-1)}+b) < 0$ ξ_i will just go to zero. Therefore, we can reformulate (1) and (2) and the proof is complete.

1.2 Part b

From class, we found that the distance from the hyperplane can be defined as

$$r_i = \frac{(w^Tx_i+b)}{\|w\|} \tag{5}$$

At the solution of the minimization problem (1) (when $\xi_i^* > 0$), we know that

$$\xi_i^* = 1-t^{(i)}(w^{*T}x^{(i)}+b^*) \tag{6}$$

Which can be reformulated as

$$t^{(i)}(w^{*T}x^{(i)} + b^*) = 1 - \xi_i^* \tag{7}$$

$$t^{(i)}\frac{(w^{*T}x^{(i)} + b^*)}{\|w\|} = \frac{1 - \xi_i^*}{\|w\|} \tag{8}$$

$$r_i = \frac{1 - \xi_i^*}{t^{(i)}\|w\|} \tag{9}$$

and we get that our distance r_i is proportional to ξ_i^*.

1.3 Part c

We are asked to find the derivatives $\bigtriangledown_w E(w,b)$ and $\frac{\partial}{\partial b}E(w,b)$ of

$$E(w,b) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \max\left(0, 1 - t^{(i)}(w^T x^{(i)} + b)\right) \tag{10}$$

To find this partial derivative, the main thing to consider is the max $\left(0, 1 - t^{(i)}(w^T x^{(i)} + b)\right)$ term, which is the only term that poses some difficulties. To do so, I used a subderivative, which was conditioned on the value of this term. Naturally, if $1 - t^{(i)}(w^T x^{(i)} + b) \leq 0$, this value will be zero and therefore the derivative will be zero. However, if $1 - t^{(i)}(w^T x^{(i)} + b) > 0$, this term will actually have a partial derivative. This logic can be explained via the expression

$$E_i(w) = \begin{cases} 0 & \text{if } 1 - t^{(i)}(w^T x^{(-1)} + b) \leq 0 \\ -t^i x^{(i)} & \text{if } 1 - t^{(i)}(w^T x^{(i)} + b) > 0 \end{cases} \tag{11}$$

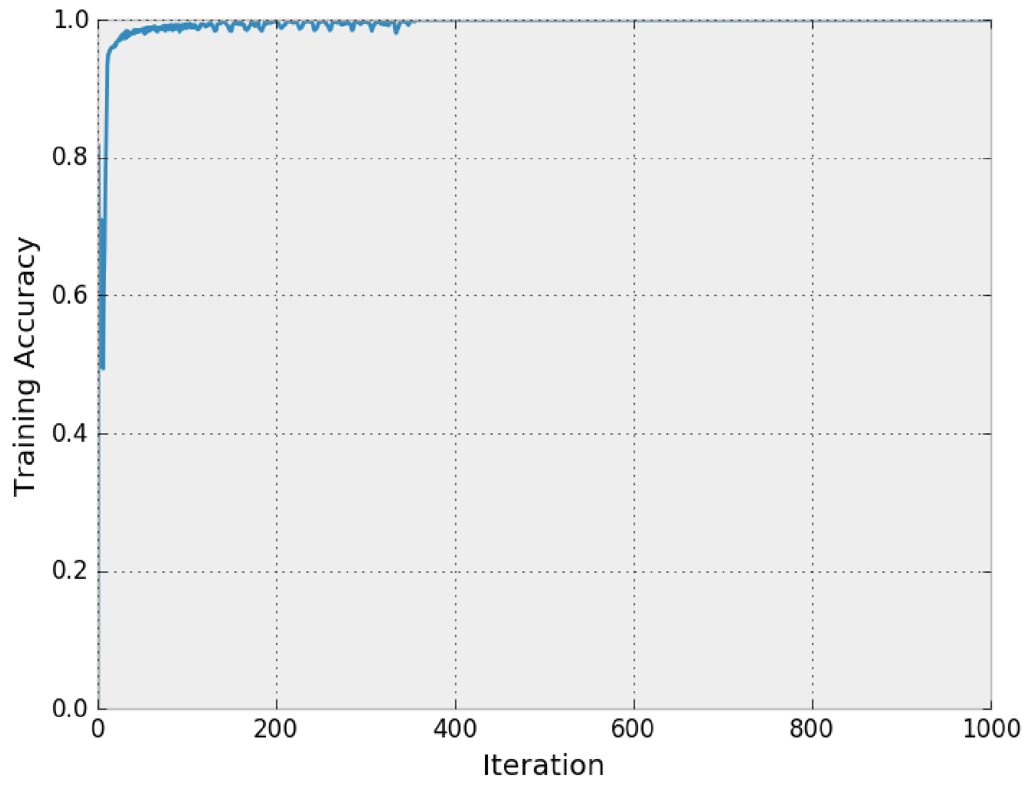
and $\bigtriangledown E(w,b) = w + C\sum\limits_{i=1}^{N} E_i(w)$. By similar logic, I can get

$$E_i(b) = \begin{cases} 0 & \text{if } 1 - t^{(i)}(w^T x^{(-1)} + b) \leq 0 \\ -t^i & \text{if } 1 - t^{(i)}(w^T x^{(i)} + b) > 0 \end{cases} \tag{12}$$

and $\frac{\partial}{\partial b}E(w,b) = w + C\sum\limits_{i=1}^{N} E_i(b)$.

1.4 Part d

For parts d and f, I used the same Python code, which output the singular plot shown in Part f. So I will redirect this question to part f, which will show the total code and resulting plot. However, the plot of the training accuracy is shown below: As you can see, there is some noise for about 350 iterations, then the accuracy heads towards 100%.



1.5 Part e

Part e is very similar to $E_i(w)$ and $E_i(b)$ of part c. Using the logic explained there, I will get

$$\nabla_w E^{(i)}(w,b) = \begin{cases} \frac{1}{N}w & \text{if } 1 - t^{(i)}(w^T x^{(-1)} + b) \leq 0 \\ \frac{1}{N}w - Ct^i x^{(i)} & \text{if } 1 - t^{(i)}(w^T x^{(i)} + b) > 0 \end{cases} \tag{13}$$

and

$$\frac{\partial}{\partial b} E^{(i)}(w,b) = \begin{cases} 0 & \text{if } 1 - t^{(i)}(w^T x^{(-1)} + b) \leq 0 \\ -Ct^i & \text{if } 1 - t^{(i)}(w^T x^{(i)} + b) > 0 \end{cases} \tag{14}$$

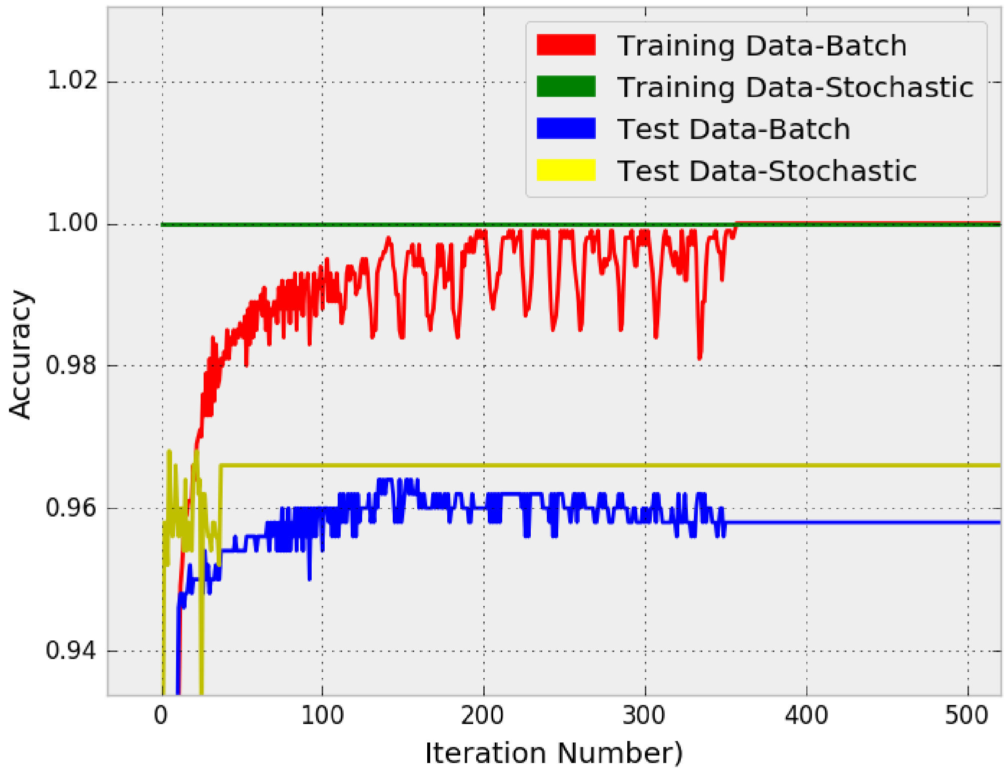
1.6 Part f

```
import numpy as np
import pandas
import math
from operator import add
import matplotlib.patches as mpatches
from matplotlib import pyplot as plt;
if "bmh" in plt.style.available: plt.style.use("bmh");

#Import Data
training = pandas.read_csv("digits_training_data.csv", sep=",",header=None)
```

```
trainingL = pandas.read_csv("digits_training_labels.csv", sep=",",header=None)
test = pandas.read_csv("digits_test_data.csv", sep=",",header=None)
testL = pandas.read_csv("digits_test_labels.csv", sep=",",header=None)


(tr_row,tr_col) = np.shape(training)
(te_row,te_col) = np.shape(test)

len_tr          = len(trainingL)
len_te          = len(testL)


trainingL[trainingL==4] = -1
trainingL[trainingL==9] = 1
testL[testL==4] = -1
testL[testL==9] = 1

wGr = np.zeros((tr_col,1))
wst = np.zeros((tr_col,1))
bGr = 0
bst = 0
eta = 0.001
C   = 3
#l;
num_it=100000
ErrorGr=[]
ErrorSt=[]
Tr_Ac_Ba = []
Te_Ac_Ba = []
Tr_Ac_St = []
Te_Ac_St = []

for i in range(num_it):
    alpha = eta/(1+float(i)*eta)
    E=0
    wgrad=0
    bgrad=0
    Grad_ac=tr_row
    for j in range(tr_row):
        t = np.asarray(trainingL[0][j])
        X = np.asarray(training.iloc[[j]])
        term = 1-t*(np.transpose(wGr).dot(np.transpose(X))+bGr)
        wg = -C*t*np.transpose(X)
        bg = -C*t
        if term<0:
            term  = 0
            wg    = np.zeros((tr_col,1))
            bg    = 0
        elif term>=1:
            Grad_ac = Grad_ac-1
        wgrad = wgrad+wg
        bgrad = bgrad+bg
    Accuracy = float(Grad_ac)/float(tr_row)
    Tr_Ac_Ba.append(Accuracy)
    wgrad = map(add,wGr,wgrad)
    wGr     = wGr-np.dot(alpha,wgrad)
    bGr     = bGr-alpha*bgrad


    E=0
    Stoch_ac=tr_row
    for j in np.random.permutation(tr_row):
        t = np.asarray(trainingL[0][j])
        X = np.asarray(training.iloc[[j]])
        term = 1-t*(np.transpose(wst).dot(np.transpose(X))+bst)
        wg = -alpha*C*t*np.transpose(X)
        bg = -alpha*C*t
        wv = alpha*wst
        if term<0:
            term  = 0
            wg    = 0
            bg    = 0
        elif term>=1:
            Stoc_ac = Stoch_ac-1
        wg2 = -(wv/tr_row+wg)
        wst = map(add,wst,wg2)
        wst = np.asarray(wst)
        bst = bst-bg
    Accuracy = float(Stoch_ac)/float(tr_row)
    Tr_Ac_St.append(Accuracy)

    Ba_te_ac = te_row
    St_te_ac = te_row
    for j in range(te_row):
        t = np.asarray(testL[0][j])
        X = np.asarray(test.iloc[[j]])
        term_ba = t*(np.transpose(wGr).dot(np.transpose(X))+bGr)
        term_st = t*(np.transpose(wst).dot(np.transpose(X))+bst)
        if term_ba<0:
            Ba_te_ac = Ba_te_ac-1
        elif term_st<0:
            St_te_ac = St_te_ac-1
    Accuracy = float(Ba_te_ac)/float(te_row)
    Te_Ac_Ba.append(Accuracy)
    Accuracy = float(St_te_ac)/float(te_row)
    Te_Ac_St.append(Accuracy)
```

```
plt.plot(range(num_it),Tr_Ac_Ba,"r",range(num_it),Tr_Ac_St,"g",range(num_it),Te_Ac_Ba,"b",range
plt.ylabel('Error')
plt.xlabel('Iteration Number)')
red_patch = mpatches.Patch(color='red', label='Training Data-Batch')
green_patch = mpatches.Patch(color='green', label='Training Data-Stochastic')
blue_patch = mpatches.Patch(color='blue', label='Test Data-Batch')
yellow_patch = mpatches.Patch(color='yellow', label='Test Data-Stochastic')
plt.legend(handles=[red_patch,green_patch, blue_patch,yellow_patch])
plt.show()
```

Which produces the following plot output



1.7 Part g

I can conclude that the batch gradient descent takes much longer to converge compared to the stochastic gradient descent. This makes sense, because through one iteration of the training data, the batch method is updating the weighting parameter once. However, the stochastic descent method is updating the weighting parameter at each training point, which comes out to around 1000 (in this example) times more updating. Therefore, it's easy to see how the stochastic method would converge quicker. I made this conclusion by looking at my data and the results.

1.8 Part h

We are given the primal function

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{s.t} \quad & t^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \quad (i = 1, \ldots, \text{N}) \end{aligned} \tag{15}$$

and asked to show the Lagrangian function and to derive the dual formulation of this problem. To show the dual, we utilize two dual variables, α and β. With the above constraints, we can generate the Lagrangian

$$L(w,b,\xi,\alpha,\beta) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\left(t^{(i)}(w^T x^{(i)} + b) - 1 + \xi_i\right) - \beta^T\xi_i \tag{16}$$

where $\alpha_i, \beta_i \geq 0 \forall i = 1, \ldots, N$. With this Lagrangian expression, we can now solve for the optimal primal parameters w^*, b^*, ξ^* by setting the partial derivatives equal to zero. We see

$$\begin{aligned} \nabla_w L(w,b,\xi,\alpha,\beta) = w - \sum_{i=1}^{N}\alpha_i t^{(i)}x^{(i)} = 0 \\ \Rightarrow w^* = \sum_{i=1}^{N}\alpha_i t^{(i)}x^{(i)} \end{aligned} \tag{17}$$

$$\frac{\partial L(w,b,\xi,\alpha,\beta)}{\partial b} \Rightarrow \sum_{i=1}^{N}\alpha_i t^{(i)} = 0 \tag{18}$$

$$\frac{\partial L(w,b,\xi,\alpha,\beta)}{\partial \xi_i} \Rightarrow C - \alpha_i - \beta_i = 0 \tag{19}$$

So we get an optimal parameter for w and constraints for b and ξ. Substituting these parameters back into the Lagrangian, we obtain

$$L = \frac{1}{2}\left(\sum_{i=1}^{N}\alpha_i t^{(i)}x^{(i)}\right)^T\left(\sum_{i=1}^{N}\alpha_i t^{(i)}x^{(i)}\right) + \sum_{i=1}^{N}\xi_i\left(C - \alpha_i - \beta_i\right) - \sum_{i=1}^{N}\alpha_i t^{(i)}b - \sum_{i=1}^{N}\alpha_i t^{(i)}(\alpha_i t^{(i)}x^{(i)})^T x^{(i)} + \sum_{i=1}^{N}\alpha_i \tag{20}$$

$$= -\frac{1}{2}\left(\sum_{i=1}^{N}\alpha_i t^{(i)}x^{(i)}\right)^T\left(\sum_{i=1}^{N}\alpha_i t^{(i)}x^{(i)}\right) + \sum_{i=1}^{N}\alpha_i \tag{21}$$

and since α_i and $t^{(i)}$ are scalars, this can be simplified to

$$L = -\frac{1}{2}\left(\sum_{i,j=1}^{N}\alpha_i\alpha_j t^{(i)}t^{(j)}x_i^T x_j\right) + \sum_{i=1}^{N}\alpha_i \tag{22}$$

With all of this said, the overall dual optimization problem can be reformulated as

$$\begin{aligned}
&\underset{\alpha,\beta}{\text{maximize}} -\frac{1}{2}\left(\sum_{i,j=1}^{N}\alpha_i\alpha_j t^{(i)}t^{(j)}x_i^T x_j\right)+\sum_{i=1}^{N}\alpha_i \\
&\text{s.t.} \sum_{i=1}^{N}\alpha_i t^{(i)}=0 \\
&C-\alpha_i-\beta_i=0 \\
&\forall i,\quad \alpha_i\geq 0,\beta_i\geq 0
\end{aligned} \tag{23}$$

From this point, we can simplify the optimization problem in two areas. First, we can define a kernel operation $k(x_i, x_j) = x_i^T x_j$. Next, we can see that the only time β shows up is in one of the constraints. knowing $\beta_i \geq 0$, we can redefine the optimization problem as

$$\begin{aligned}
&\underset{\alpha,\beta}{\text{maximize}} -\frac{1}{2}\left(\sum_{i,j=1}^{N}\alpha_i\alpha_j t^{(i)}t^{(j)}k(x_i,x_j)\right)+\sum_{i=1}^{N}\alpha_i \\
&\text{s.t.} \sum_{i=1}^{N}\alpha_i t^{(i)}=0 \\
&\forall i\ \ 0\leq\alpha_i\leq C
\end{aligned} \tag{24}$$

Now our optimization is only reliant on dual variables and kernel functions and the proof is complete.

1.9 Part i

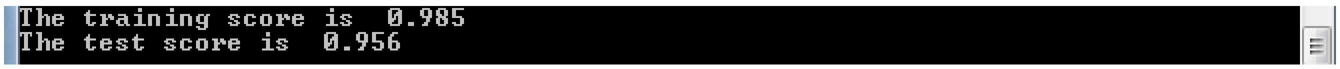
For this section, I used the SVM.svc package in sklearn. As was suggesting in piazza, I used a grid search to find the optimal γ and C parameters for the optimization. The code below was used, with the results shown after

```
from matplotlib import pyplot
import matplotlib as mpl
import numpy as np
from time import time
import pandas




#Import Data
data_train = pandas.read_csv("digits_training_data.csv", sep=",",header=None)
targets_train = pandas.read_csv("digits_training_labels.csv", sep=",",header=None)
data_test = pandas.read_csv("digits_test_data.csv", sep=",",header=None)
targets_test = pandas.read_csv("digits_test_labels.csv", sep=",",header=None)


targets_train=targets_train.rename(columns={0:'digit'})
targets_test=targets_test.rename(columns={0:'digit'})

train = [data_train,targets_train]
```

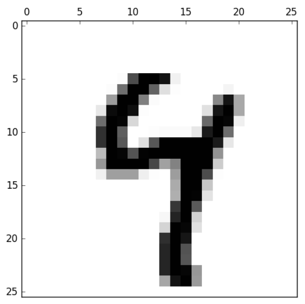
```
test = [data_test,targets_test]

train = pandas.concat(train,axis=1)
test  = pandas.concat(test,axis=1)



train4 = train[train['digit']==4]
train9 = train[train['digit']==9]

P4    = float(len(train4))/float(len(train))
P9    = 1.0-P4
test4 = test[test['digit']==4]
test9 = test[test['digit']==9]

Mean_Train4 = train4.mean()
Mean_Train9 = train9.mean()
Mean_Test4 = test4.mean()
Mean_Test9 = test9.mean()

matrix = data_train
mean =  np.mean(matrix,axis=0)
# make a mean matrix the same shape as data for subtraction
mean_mat = np.outer(np.ones((1000,1)),mean)

cov = matrix - mean_mat
Cov = np.dot(cov.T,cov)/(1000 -1)

MTr4 = np.asarray(Mean_Train4[0:676])
MTr9 = np.asarray(Mean_Train9[0:676])

print MTr4
print Mtr9



Gamma4 = -0.5*np.dot(np.dot(np.transpose(MTr4),np.linalg.pinv(Cov)),MTr4)
Gamma9 = -0.5*np.dot(np.dot(np.transpose(MTr9),np.linalg.pinv(Cov)),MTr9)



Beta4  = np.dot(np.linalg.pinv(Cov),MTr4)
Beta9  = np.dot(np.linalg.pinv(Cov),MTr9)

Tr4 = train4[0:676]
Tr9 = train9[0:676]
Te4 = test4[0:676]
Te9 = test9[0:676]
```

```
ProbTr4 = data_train.apply(lambda x: P4*np.exp(np.dot(np.transpose(Beta4),x)+Gamma4)/(np.exp(np
    np.exp(np.dot(np.transpose(Beta9),x)+Gamma9)), axis=1)
ProbTr9 = data_train.apply(lambda x: P9*np.exp(np.dot(np.transpose(Beta9),x)+Gamma9)/(np.exp(np
    np.exp(np.dot(np.transpose(Beta9),x)+Gamma9)), axis=1)

ProbTe4 = data_test.apply(lambda x: P4*np.exp(np.dot(np.transpose(Beta4),x)+Gamma4)/(np.exp(np
    np.exp(np.dot(np.transpose(Beta9),x)+Gamma9)), axis=1)
ProbTe9 = data_test.apply(lambda x: P9*np.exp(np.dot(np.transpose(Beta9),x)+Gamma9)/(np.exp(np
    np.exp(np.dot(np.transpose(Beta9),x)+Gamma9)), axis=1)

ProbTr=ProbTr4
ProbTe=ProbTe4
ProbTr[ProbTr4>ProbTr9]=4
ProbTr[ProbTr4<ProbTr9]=9
ProbTe[ProbTe4>ProbTe9]=4
ProbTe[ProbTe4<ProbTe9]=9

Train_Error = 1.0-sum(targets_train['digit']==ProbTr)/float(len(targets_train))
Test_Error = 1.0-sum(targets_test['digit']==ProbTe)/float(len(targets_test))

print "The training accuracy is ", 1-Train_Error
print "The test accuracy is ", 1-Test_Error
```

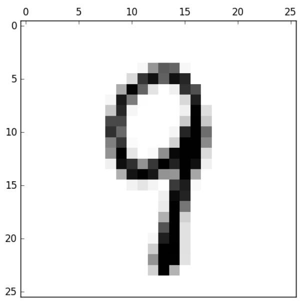
With the following output and 5 miscalculated images



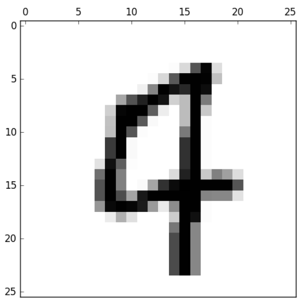
```
The training score is  0.985
The test score is  0.956
```



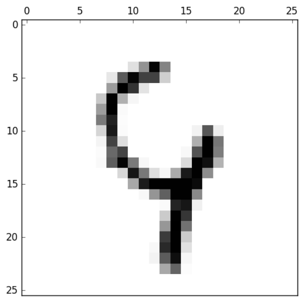
(a) Miscalculated as:9



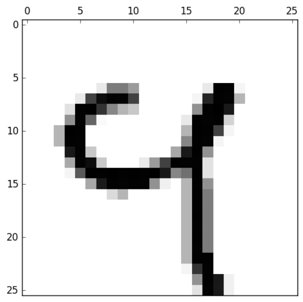
(b) Miscalculated as:4



(c) Miscalculated as:9



(a) Miscalculated as:4



(b) Miscalculated as:4

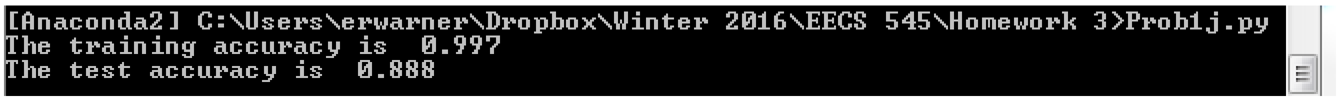
1.10 Part j

The code to find the LDA is shown below, with the resulting test and training accuracy.

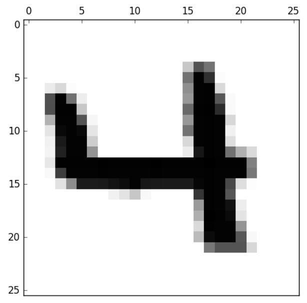
```
import matplotlib.pyplot as plt
import numpy as np
from time import time
import pandas



#Import Data
data_train = pandas.read_csv("digits_training_data.csv", sep=",",header=None)
targets_train = pandas.read_csv("digits_training_labels.csv", sep=",",header=None)
data_test = pandas.read_csv("digits_test_data.csv", sep=",",header=None)
targets_test = pandas.read_csv("digits_test_labels.csv", sep=",",header=None)


targets_train=targets_train.rename(columns={0:'digit'})
targets_test=targets_test.rename(columns={0:'digit'})

train = [data_train,targets_train]
test = [data_test,targets_test]

train = pandas.concat(train,axis=1)
test  = pandas.concat(test,axis=1)

train4 = train[train['digit']==4]
train9 = train[train['digit']==9]
test4 = test[test['digit']==4]
test9 = test[test['digit']==9]

Mean_Train4 = train4.mean()
Mean_Train9 = train9.mean()
Mean_Test4 = test4.mean()
Mean_Test9 = test9.mean()
```

```
matrix = data_train
mean =  np.mean(matrix,axis=0)
# make a mean matrix the same shape as data for subtraction
mean_mat = np.outer(np.ones((1000,1)),mean)

cov = matrix - mean_mat
Cov = np.dot(cov.T,cov)/(1000 -1)

MTr4 = np.asarray(Mean_Train4[0:676])
MTr9 = np.asarray(Mean_Train9[0:676])

Gamma4 = -0.5*np.dot(np.dot(np.transpose(MTr4),np.linalg.pinv(Cov)),MTr4)
Gamma9 = -0.5*np.dot(np.dot(np.transpose(MTr9),np.linalg.pinv(Cov)),MTr9)

Beta4  = np.dot(np.linalg.pinv(Cov),MTr4)
Beta9  = np.dot(np.linalg.pinv(Cov),MTr9)

Tr4 = train4[0:676]
Tr9 = train9[0:676]
Te4 = test4[0:676]
Te9 = test9[0:676]

ProbTr = data_train.apply(lambda x: np.exp(np.dot(np.transpose(Beta4),x)+Gamma4)/(np.exp(np.do
    np.exp(np.dot(np.transpose(Beta9),x)+Gamma9)), axis=1)

ProbTe = data_test.apply(lambda x: np.exp(np.dot(np.transpose(Beta4),x)+Gamma4)/(np.exp(np.dot
    np.exp(np.dot(np.transpose(Beta9),x)+Gamma9)), axis=1)

ProbTr[ProbTr>0.5]=4
ProbTr[ProbTr<=0.5]=9
ProbTe[ProbTe>0.5]=4
ProbTe[ProbTe<=0.5]=9

Train_Error = 1.0-sum(targets_train['digit']==ProbTr)/float(len(targets_train))
Test_Error = 1.0-sum(targets_test['digit']==ProbTe)/float(len(targets_test))

print Train_Error
print Test_Error
```

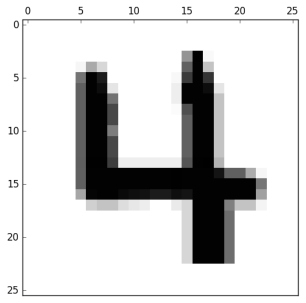
With the following output and first 5 miscalculated images.



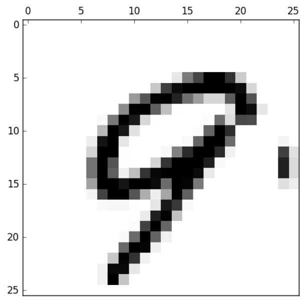
```
[Anaconda2] C:\Users\erwarner\Dropbox\Winter 2016\EECS 545\Homework 3>Prob1j.py
The training accuracy is  0.997
The test accuracy is  0.888
```



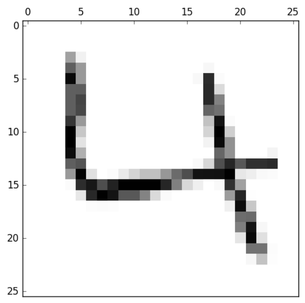
(a) Miscalculated as:9



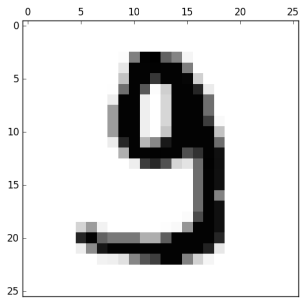
(b) Miscalculated as:9



(c) Miscalculated as:4



(a) Miscalculated as:9



(b) Miscalculated as:4

2 Problem 2

For this problem, I will just show my code, which is shown below. Basically I was changing sklearn solvers and check the performance of each.

```
import numpy as np
from matplotlib import pyplot
import matplotlib as mpl
import pandas
import time

# Import datasets, classifiers and performance metrics
from sklearn import datasets, svm, pipeline
from sklearn.kernel_approximation import (RBFSampler,
                                          Nystroem)
from sklearn.decomposition import PCA
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier, GradientBoostingClassifier
# load training data
trainLabels = np.loadtxt('trainingLabels.gz', dtype=np.uint8, delimiter=',')
trainData = np.loadtxt('trainingData.gz', dtype=np.uint8, delimiter=',')

# load test data
testData = np.loadtxt('testData.gz', dtype=np.uint8, delimiter=',')

data_train = trainData
targets_train = trainLabels
data_test = (testData)

(tr_row,tr_col) = np.shape(data_train)
(te_row,te_col) = np.shape(data_test)

print te_row
print te_col
print tr_row
print tr_col

len_tr         = len(targets_train)

targets_train = np.reshape(targets_train,(len_tr,))

X, y = data_train, targets_train
X = np.repeat(X, 10, axis=0)
y = np.repeat(y, 10, axis=0)

n_estimators = 10
```

```
start = time.time()
clf = OneVsRestClassifier(BaggingClassifier(SVC(kernel='rbf', probability=True, class_weight='a
clf.fit(X, y)
end = time.time()
print "Bagging SVC", end - start, clf.score(X,y)
proba = clf.predict_proba(X)
tagets_test = clf.predict(data_test)

#Write to a Dataframe
df = pandas.DataFrame(targets_test, range(1,te_row+1),columns=['category'])

df.to_csv('erwaner_images_pred_bagging.csv')
```

3 Problem 3

3.1 Part a

For this problem, we are asked to find the feature map ϕ for the kernal:

$$k(u,v) = (< u,v > +1)^4 \tag{25}$$

We are asked to show the expression for u, v of dimension 3. To do so, it is relatively easy to show

$$(< u,v > +1)^4 =< u,v >^4 +4 < u,v >^3 +6 < u,v >^2 +4 < u,v > +1 \tag{26}$$

$$= (u_1v_1 + u_2v_2 + u_3v_3)^4 + 4(u_1v_1 + u_2v_2 + u_3v_3)^3 + 6(u_1v_1 + u_2v_2 + u_3v_3)^2 + 4(u_1v_1 + u_2v_2 + u_3v_3) + 1 \tag{27}$$

From here, I will decompose one of the above expressions, i.e.

$$(u_1v_1 + u_2v_2 + u_3v_3)^2 \tag{28}$$

$$= u_1v_1u_1v_1 + u_1v_1u_2v_2 + \cdots + u_3v_3u_3v_3 \tag{29}$$

So, as you can see above, this is the combination of every $u_iv_iu_jv_j$ for $i, j = 1, \ldots, d$, or

$$\sum_{j,i=1}^{d} u_iu_jv_iv_j \tag{30}$$

Looking at the cubed version of this we will get a very similar expression, i.e.,

$$\sum_{j,i,k=1}^{d} u_iu_ju_kv_iv_jv_k \tag{31}$$

with analogous results for the expression to the fourth. With this knowledge, we can easily see

$$k(u,v) = (< u,v > +1)^4 \tag{32}$$

$$= 1 + 4u^Tv + 6\sum_{j,i=1}^{d} u_iu_jv_iv_j + 4\sum_{j,i,k=1}^{d} u_iu_ju_kv_iv_jv_k + \sum_{j,i,k,m=1}^{d} u_iu_ju_ku_mv_iv_jv_kv_m \tag{33}$$

Which can then be decomposed into the feature map

$$\phi(x) = \left(1, 2x, \sqrt{6}\underset{i,j=1,\ldots,d}{x_ix_j}, 2\underset{i,j,k=1,\ldots,d}{x_ix_jx_k}, \underset{i,j,k,m=1,\ldots,d}{x_ix_jx_kx_m}\right) \tag{34}$$

where $\underset{i,j=1,\ldots,d}{x_ix_j}$ is the sequence of all terms of the vector being multiplied with one another. It is important to note that if two feature vectors $\phi(x)$ and $\phi(y)$ were to be multiplied $\phi(x)^T\phi(y)$, the sequence iterative scalars i, j, k, m must be matched up for x and y as in the sum expressions above.

3.2 Part b

For this problem, it is helpful to know that any matrix K can be decomposed into $K = Q\Lambda Q^{-1}$, where Q is a matrix of the eigenvectors and Λ is a diagonal matrix of the eigenvalues. It is also important, from Homework 1, that we found that if $\Lambda \geq 0$, then K is positive definite. With this being said, what we are trying to prove, in order for K to be a positive-definite kernel, is that it is positive semi-definite and symmetric. These are the only conditions that matter.

3.2.1 Part i: $k(x,z) = k_1(x,z) + k_2(x,z)$

The first thing to note is that the resulting expression will be $K = K_1 + K_2$, where K_1 and K_2 are the overall kernel matrices. It is trivial to see that the resulting matrix will be symmetric since

$$k(x,z) = k_1(x,z) + k_2(x,z) = k_1(z,x) + k_2(z,x) = k(z,x) \tag{35}$$

since k_1 and k_2 are valid kernels, we know$k_1(x,z) = k_1(z,x)$ and $k_2(x,z) = k_2(z,x)$ and therefore the above holds. Additionally, the resulting matrix will be positive semidefinite, since adding two positive semidefinite matrices together cannot create a negative semidefinite matrix, i.e.,

$$x^T(K_1 + K_2)x = x^T K_1 x + x^T K_2 x \geq 0 \tag{36}$$

since both K_1 and K_2 are p.s.d. And therefore the kernel is valid

3.2.2 Part ii: $k_1(x,z) - k_2(x,z)$

This kernel is easy to disprove. Consider $K_1 = I$, $K_2 = 2I$, both of which are valid kernels since they are both symmetric and positive semidefinite. However, $K_1 - K_2 = -I$ is negative definite and therefore a violation. Therefore, a counter-example is shown and it is not necessarily a positive definite kernel.

3.2.3 Part iii: $k(x,z) = ak_1(x,z)$

This is a very easy kernel to show, given that a is a positive number. It is easy to see, via the properties of matrices and scalars

$$(aK_1)^T = aK_1^T = aK_1 \qquad \text{(because } K_1 \text{ is symmetric)} \tag{37}$$

$$x^T(aK_1)x = ax^T K_1 x \geq 0 \qquad \text{(since } a > 0 \text{ and } K_1 \text{ is p.s.d)} \tag{38}$$

3.2.4 Parti v: $k_1(x,z)k_2(x,z)$

It is easy to see the resulting kernel matrix will be symmetric, since

$$k(x,z) = k_1(x,z)k_2(x,z) = k_1(z,x)k_2(z,x) = k(z,x) \tag{39}$$

since k_1 and k_2 are valid kernels, we know$k_1(x,z) = k_1(z,x)$ and $k_2(x,z) = k_2(z,x)$ and therefore the above holds. Next, we must show the resulting kernel matrix is positive definite, which is a significantly more difficult thing to prove. Consider the 2×2 matrix case, with two matrices

$$A = \begin{bmatrix} a & b \\ b & c \end{bmatrix}, B = \begin{bmatrix} d & e \\ e & f \end{bmatrix} \tag{40}$$

From the properties of a positive semi-definite matrix, we know that all of the determinants of leading minors are non-negative, i.e, $ac \geq b^2$ and $df \geq e^2$. The element wise combination of the two matrices will now be

$$C = \begin{bmatrix} ad & be \\ be & cf \end{bmatrix} \tag{41}$$

with the determinant $adcf - b^2e^2$ and since $ac \geq b^2$ and $df \geq e^2$, we can safely say $adcf \geq b^2e^2$. Therefore, this can be extrapolated to any size of K, and the kernel is valid and positive-definite.

Also, since the kernal is an inner product of a transformation $\phi_1(x) = k_1(x,z)$ and $\phi_2(x) = k_2(x,z)$ the kernel is valid.

3.2.5 Part v: $k(x,z) = f(x)f(z)$

Once again, the symmetric property is trivial, using the same logic as above. Since f is a scalar valued function,

$$k(x,z) = f(x)f(z) = f(z)f(x) = k(z,x) \tag{42}$$

Additionally, we can say that the matrix is positive semi-definite. Consider the 2-D case, i.e., when

$$K = \begin{bmatrix} f(x)^2 & f(x)f(z) \\ f(x)f(z) & f(z)^2 \end{bmatrix} \tag{43}$$

And therefore the determinant will be $f(x)^2 f(z)^2 - f(x)f(z)f(z)f(z) = 0$ and is not negative. Additionally, the eigenvalue equation will come out to be

$$(\lambda - f(x)^2)(\lambda - f(z)^2) - f(x)^2 f(z)^2 \tag{44}$$
$$= \lambda^2 - \lambda(f(x)^2 + f(z)^2) \tag{45}$$
$$\Rightarrow \lambda = \{0, \left(f(x)^2 + f(z)^2\right)\} \tag{46}$$

clearly, $(f(x)^2 + f(z)^2)$ will be positive and therefore the eigenvalues are positive. This can therefore be extrapolated to higher dimensions and the kernel is valid.

Also, since this kernel is an inner product of a transformation $\phi(x) = f(x)$, we can say that the kernel is positive semidefinite and therefore valid.

3.2.6 Part vi: $k(x,z) = p(k_1(x,z))$

Once again, symmetry is easy to prove, since we already know $k_1(x,z) = k_1(z,x)$ and therefore

$$k(x,z) = p(k_1(x,z)) = p(k_1(z,x)) = k(z,x) \tag{47}$$

As far as positive definiteness, we can see, from parts *i* and *iii* that any addition and positive scalar multiplication of valid kernels still produces another valid kernel. Therefore, we must merely prove that taking the polynomial of kernel terms will maintain the properties of a kernel. To see this, we look to *iv*, which says that multiplication of any two valid kernels will produce a valid kernel. Using this logic, we can set $k_2 = k_1$ and see $k(x,z) = k_1(x,z)^2$ is a valid kernel. With this in hand, we can set $k_2 = k_1^2$ and see $k(x,z) = k_1(x,z)^3$ is a valid kernel. This process can be repeated to obtain any polynomial value and the kernel is therefore valid.

3.2.7 Part vii: Prove $k(x,z) = exp\left(\frac{-\|x-z\|^2}{2\sigma^2}\right)$ can be expressed as $\phi(x)^T\phi(z)$

From the exponential expression, we can see

$$exp\left(\frac{-\|x-z\|^2}{2\sigma^2}\right) = exp\left(\frac{-x^Tx - z^Tz + 2z^Tx}{2\sigma^2}\right) \tag{48}$$
$$= exp\left(\frac{-x^Tx}{2\sigma^2}\right) exp\left(\frac{-z^Tz}{2\sigma^2}\right) exp\left(\frac{z^Tx}{\sigma^2}\right) \tag{49}$$

From the definition of an exponential, it can be expressed as the power series

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \tag{50}$$

and therefore, $k(x,z) = exp\left(\frac{x^Tz}{\sigma^2}\right)$ can be expressed as

$$k(x,z) = 1 - \frac{(x^Tz)^T(x^Tz)}{\sigma^2} + \frac{\left((x^Tz)^T(x^Tz)\right)^2}{2!(\sigma^4)} - \frac{\left((x^Tz)^T(x^Tz)\right)^3}{3!(\sigma^6)} + ... \tag{51}$$

Which looks very similar to part a of this problem. The exception is that, for this problem, the series is infinite and there is no constant term. Additionally, there will be another exponential term $exp\left(\frac{-x^Tx}{2\sigma^2}\right)$ or $exp\left(\frac{-z^Tz}{2\sigma^2}\right)$. Therefore, the overall expression will be

$$\phi(x) = exp\left(\frac{-x^Tx}{2\sigma^2}\right)\left(1, -\underset{i,j=1,\ldots,d}{x_ix_j} * \frac{1}{\sigma^2}, \quad \underset{i,j,k,m=1,\ldots,d}{x_ix_jx_kx_m} * \frac{1}{2!\sigma^2}, \ldots\right) \tag{52}$$

With each successive term increasing the number of x terms by 2 and the denominator by an additional number in the factorial. Additionally the sign will change on each term and the series will be infinite.

4 Problem 4

4.1 Part a

For part a we are given the closed-form solution for the error function for ridge regression, i.e.,

$$\hat{w} = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T t \ \Rightarrow \ \hat{f}(x) = \hat{w}^T\phi(x) = t^T\Phi(\Phi^T\Phi + \lambda I)^{-1}\phi(x) \tag{53}$$

and the following matrix inverse lemma

$$(P + QRS)^{-1} = P^{-1} - P^{-1}Q(R^{-1} + SP^{-1}Q)^{-1}SP^{-1} \tag{54}$$

To start, we utilize the equation

$$\hat{w} = t^T\Phi(\Phi^T\Phi + \lambda I)^{-1}\phi(x) \tag{55}$$

More specifically, we concern ourselves with $\Phi(\Phi^T\Phi + \lambda I)^{-1}$. Defining

- $P = \lambda I$, $Q = \Phi^T$, $R = I$, $S = \Phi$

and utilizing the matrix inverse lemma and properties of the inverse, we get

$$\Phi\left\{\frac{1}{\lambda}I - \frac{1}{\lambda}\Phi^T\left[I + \frac{1}{\lambda}\Phi\Phi^T\right]^{-1}\Phi\frac{1}{\lambda}\right\} \tag{56}$$

$$=\Phi\left\{\frac{1}{\lambda}I - \frac{1}{\lambda}\Phi^T\left[(\Phi^{-T}\Phi^{-1} + \frac{1}{\lambda}I)(\Phi\Phi^T)\right]^{-1}\Phi\frac{1}{\lambda}\right\} \tag{57}$$

and since $(AB)^{-1} = B^{-1}A^{-1}$,

$$=\Phi\left\{\frac{1}{\lambda}I - \frac{1}{\lambda}\Phi^T\left(\Phi^{-T}\Phi^{-1}\right)\left[(\Phi^{-T}\Phi^{-1} + \frac{1}{\lambda}I)\right]^{-1}\Phi\frac{1}{\lambda}\right\} \tag{58}$$

$$=\frac{\Phi}{\lambda} - \frac{1}{\lambda}\left[(\Phi^{-T}\Phi^{-1} + \frac{1}{\lambda}I)\right]^{-1}\Phi\frac{1}{\lambda} \tag{59}$$

We once again use the matrix inversion lemma, this time defining

- $P = \frac{1}{\lambda}I$, $Q = \Phi^{-T}$, $R = I$. $S = \Phi^{-1}$

We get

$$\frac{\Phi}{\lambda} - \frac{1}{\lambda}\left\{\lambda I - \lambda\Phi^{-T}\left[I + \lambda\Phi^{-1}\Phi^{-T}\right]^{-1}\Phi^{-1}\lambda\right\}\Phi\frac{1}{\lambda} \tag{60}$$

$$=\left\{\Phi^{-T}\left[I + \lambda\Phi^{-1}\Phi^{-T}\right]^{-1}\Phi^{-1}\right\}\Phi \tag{61}$$

$$=\left\{\Phi^{-T}\left[(\Phi^{-1})\Phi(\Phi^T + \lambda I)(\Phi^{-T})\right]^{-1}\Phi^{-1}\right\}\Phi \tag{62}$$

$$=\left\{\Phi^{-T}(\Phi^{-T})^{-1}\left[\Phi\Phi^T + \lambda I\right]^{-1}(\Phi^{-1})^{-1}\Phi^{-1}\right\}\Phi \tag{63}$$

$$=\left[\Phi\Phi^T + \lambda I\right]^{-1}\Phi = [K + \lambda I]^{-1}\Phi \tag{64}$$

And therefore we get

$$\hat{f}(x) = t^T\Phi(\Phi^T\Phi + \lambda I)^{-1}\phi(x) \tag{65}$$

$$= t^T[K + \lambda I]^{-1}\Phi\phi(x) \tag{66}$$

$$= t^T[K + \lambda I]^{-1}k(x) = k(x)^T[K + \lambda I]^{-1}t \tag{67}$$

Which is the closed-from solution and model for kernelized ridge regression and the proof is complete.

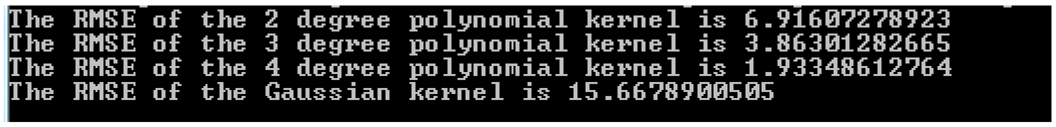
4.2 Part b

For all of the parts of this problem, I used the same code, which is shown below:

```
import numpy as np
import pandas
import math
from operator import add
from matplotlib import pyplot as plt;
import sklearn
from sklearn.kernel_ridge import KernelRidge

#Import Data
train = pandas.read_csv("steel_composition_train.csv", sep=",")
test = pandas.read_csv("steel_composition_test.csv", sep=",")

names = ["id","Carbon","Nickel","Manganese","Sulfur","Chromium","Iron","Phosphorus","Silicon"]


data_train = train[names]
targets_train = train["Strength"]

tr_len = len(targets_train)

krr2 = KernelRidge(alpha = 1,kernel="polynomial",degree=2,coef0=1)
krr2.fit(data_train, targets_train)
krr2_score = krr2.score(data_train, targets_train)

K2TR = krr2.predict(data_train)
E= K2TR - targets_train
E = np.asarray(E)
RMSE2 = np.sqrt(np.dot(np.transpose(E),E)/tr_len)


krr3 = KernelRidge(alpha = 1,kernel="polynomial",degree=3,coef0=1)
krr3.fit(data_train, targets_train)
krr3_score = krr3.score(data_train, targets_train)

K3TR = krr3.predict(data_train)
E= K3TR - targets_train
E = np.asarray(E)
RMSE3 = np.sqrt(np.dot(np.transpose(E),E)/tr_len)

krr4 = KernelRidge(alpha = 1,kernel="polynomial",degree=4,coef0=1)
krr4.fit(data_train, targets_train)
krr4_score = krr4.score(data_train, targets_train)

K4TR = krr4.predict(data_train)
E= K4TR - targets_train
```

```
E = np.asarray(E)
RMSE4 = np.sqrt(np.dot(np.transpose(E),E)/tr_len)

krrG = KernelRidge(alpha = 1,kernel="rbf")
krrG.fit(data_train, targets_train)
krrG_score = krrG.score(data_train, targets_train)

KGTR = krrG.predict(data_train)
E= KGTR - targets_train
E = np.asarray(E)
RMSEG = np.sqrt(np.dot(np.transpose(E),E)/tr_len)

print "The RMSE of the 2 degree polynomial kernel is", RMSE2
print "The RMSE of the 3 degree polynomial kernel is", RMSE3
print "The RMSE of the 4 degree polynomial kernel is", RMSE4
print "The RMSE of the Gaussian kernel is", RMSEG
```

with the following output.



```
The RMSE of the 2 degree polynomial kernel is 6.91607278923
The RMSE of the 3 degree polynomial kernel is 3.86301282665
The RMSE of the 4 degree polynomial kernel is 1.93348612764
The RMSE of the Gaussian kernel is 15.6678900505
```

As a note, there were no results for the test data and therefore I could not find the error.