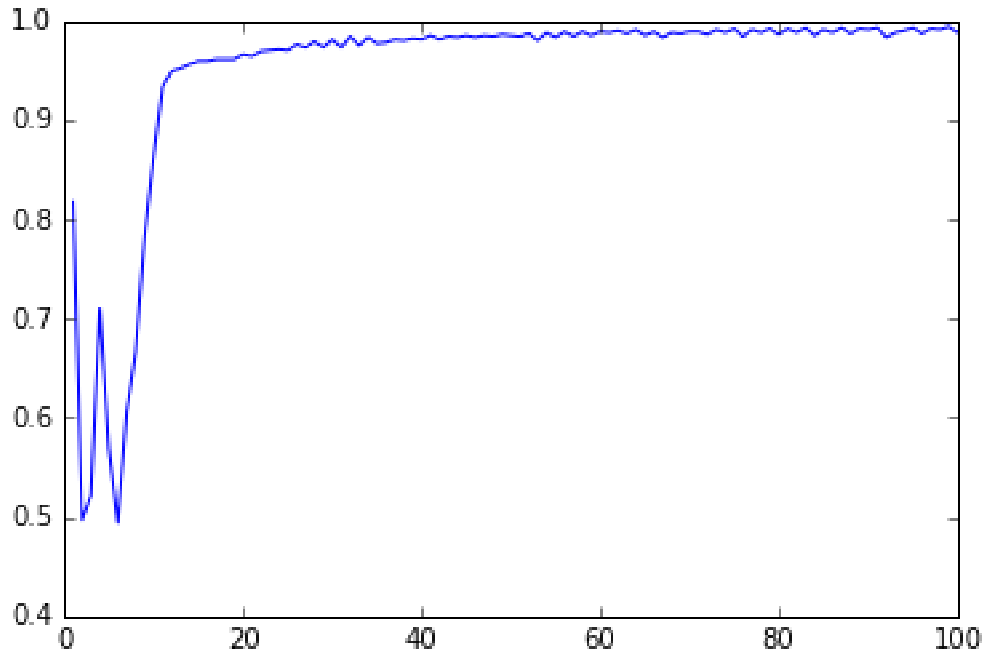
Homework 3

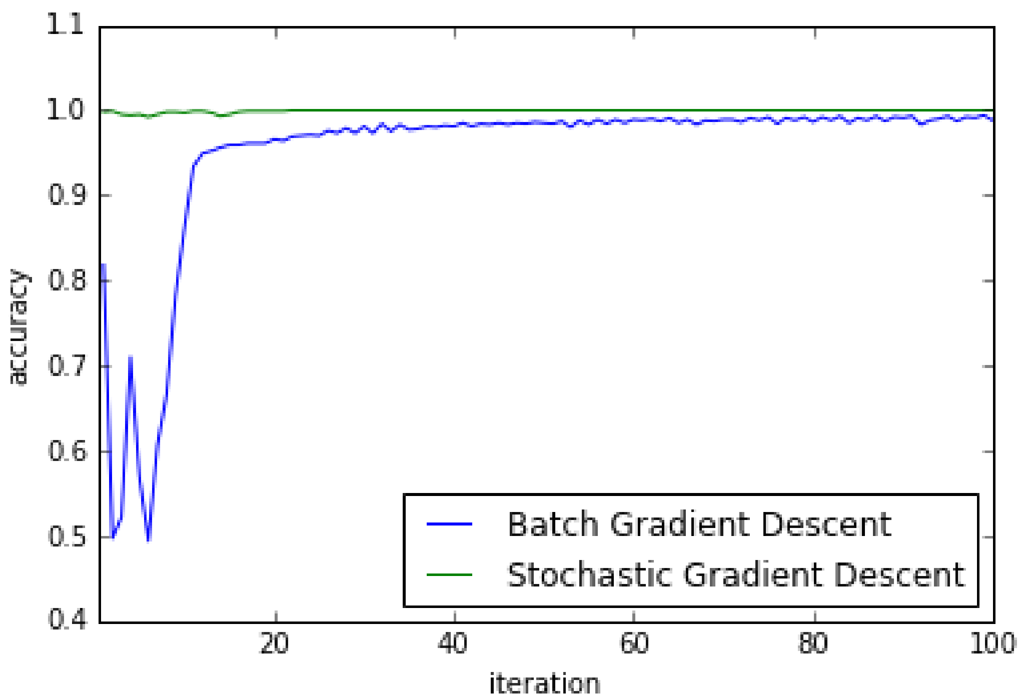
1(d)

The accuracy plot is the following:



1(f)

The accuracy plot is the following:

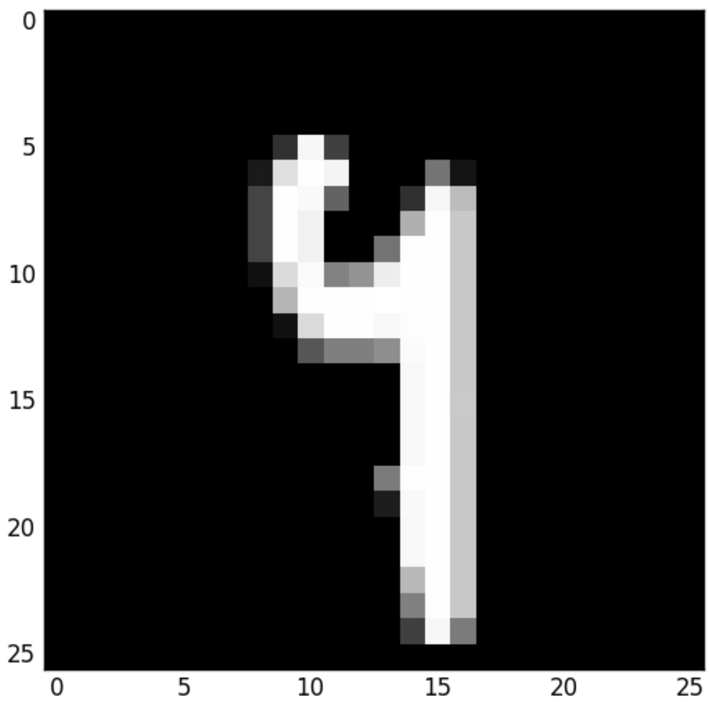


1(g)

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.

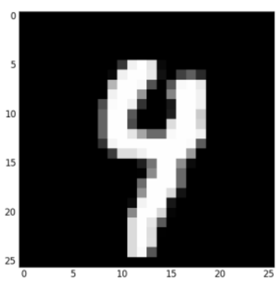
1(i)

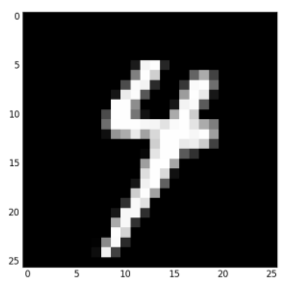
I choose Gamma=10^(-7), C=10 in this problem. The accuracy of training data and test data is 99.9% and 99.8%. Therefore, there is only 1 misclassified test image. It is the 164th of the test labels, it is "9" initially, but the SVM labels it "4". The picture is shown as follows:

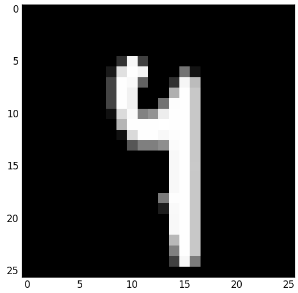


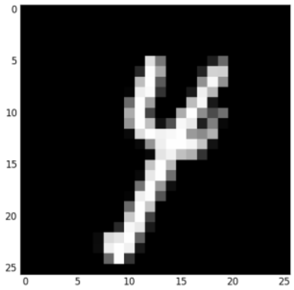
1(j)

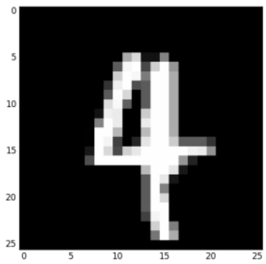
Based on the separate final accuracy, there is no significant difference between LDA and SVM.
The The accuracy of training data and test data is 99.7% and 90.4%. 5 of the misclassified test images are :











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))
```

1. (a) For problem (1), the boundary is

$t^{(i)}(w^T x^{(i)}+b) \geq 1-\xi_i$, $\xi_i \geq 0$, it is equivalent

to $\xi_i \geq \max(0, 1-t^{(i)}(w^T x^{(i)}+b))$,

Thus the problem (1) can be written as

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i,$$

subject to $\xi_i \geq \max(0, 1-t^{(i)}(w^T x^{(i)}+b))$.

Now let's consider the problem from the perspective of final optimal value of w, b, ξ, set their optimal value of this problem is w^*, b^*, ξ^*. Since $C>0$, we must have $\xi_i^* = \max(0, 1-t^{(i)}(w_*^T x^{(i)}+b^*))$, otherwise we can use $\max(0, 1-t^{(i)}(w^T x^{(i)}+b))$ to replace ξ_i^* and make the target function smaller. Therefore, whatever w^*, b^* is, the optimal ξ_i^* can be always represented by $\max(0, 1-t^{(i)}(w^T x^{(i)}+b))$.

Thus the problem is equivalent to:

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

(b) The margin hyperlane $t^{(i)}((w^*)^T x+b^*)=1$ can be written as $t^{(i)}(w^*)^T x + (t^{(i)}b^*-1)=0$, then the distance is

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

$$= |\xi_i^*| \cdot \frac{1}{\|t^{(i)} w^*\|}$$

Obviously, $|r|$ is proportional to ξ_i^* ($\xi_i^* \geq 0$)

(c) If for all i, $1-t^{(i)}(w^T x^{(i)}+b) \neq 0$, then

$$\nabla_w E(w,b) = w + C\sum_{i=1}^{N} -t^{(i)}x^{(i)} \cdot \mathbb{1}_{\{1-t^{(i)}(w^T x^{(i)}+b)>0\}}$$

$$\nabla_b E(w,b) = C\sum_{i}^{N} -t^{(i)} \mathbb{1}_{\{1-t^{(i)}(w^T x^{(i)}+b)>0\}}$$

If for some i, $1-t^{(i)}(w^T x^{(i)}+b)=0$, & for other j, $1-t^{(j)}(w^T x^{(j)}+b) \neq 0$, then the derivative is undifined at i, the subderivative is:

$$\nabla_W^- E(w,b) = w + C\left(\sum_i -t^{(i)}x^{(i)}\right) + C\left(\sum_j -t^{(j)}x^{(j)}\right) 1_{\{1-t^{(j)}(w^T x^{(j)}+b)>0\}}$$

$$\nabla_W^+ E(w,b) = w + C\sum_j -t^{(j)}x^{(j)} 1_{\{1-t^{(j)}(w^T x^{(j)}+b)>0\}}$$

$$\nabla_b^- E(w,b) = \left(C\sum_i -t^{(i)}\right) + C\sum_j -t^{(j)} 1_{\{1-t^{(j)}(w^T x^{(j)}+b)>0\}}$$

$$\nabla_b^+ E(w,b) = C\sum_j -t^{(j)} 1_{\{1-t^{(j)}(w^T x^{(j)}+b)>0\}}$$

(e) If for i, $1-t^{(i)}(w^T x^{(i)}+b) \neq 0$, then

$$\nabla_W E^{(i)}(w,b) = \frac{W}{N} + C(-t^{(i)} \cdot x^{(i)}) \cdot 1_{\{1-t^{(i)}(w^T x^{(i)}+b)>0\}}$$

$$\nabla_b E^{(i)}(w,b) = -Ct^{(i)} 1_{\{1-t^{(i)}(w^T x^{(i)}+b)>0\}}$$

if for i, $1-t^{(i)}(W^T x^{(i)}+b)=0$, then

$$\nabla_W^- E^{(i)}(w,b) = \frac{W}{N} + C(-t^{(i)} \cdot x^{(i)})$$

$$\nabla_W^+ E^{(i)}(w,b) = \frac{W}{N}$$

$$\nabla_b^- E^{(i)}(w,b) = -Ct^{(i)}$$

$$\nabla_b^+ E^{(i)}(w,b) = 0$$

(h) $L(w,b,\lambda,v) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^N \xi_i + \sum_{i=1}^N \lambda_i(1-\xi_i - t^{(i)}(w^T x^{(i)}+b)) - \sum_{i=1}^N v_i\xi_i$

Then the dual problem is $\max_{v,\lambda,\lambda_i,v_i\geq 0} \min_{w,b} L(w,b,\lambda,v)$

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^N \lambda_i t^{(i)}x^{(i)} = 0 \Rightarrow w = \sum_{i=1}^N \lambda_i t^{(i)} x^{(i)}$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^N \lambda_i t^{(i)} = 0, \quad \frac{\partial L}{\partial \xi_i} = C - \lambda_i - v_i = 0$$

$$\therefore \tilde{L}(\lambda,v) = \frac{1}{2}\left[\sum_{i=1}^N \lambda_i t^{(i)}x^{(i)}\right]\left[\sum_{i=1}^N \lambda_i t^{(i)}x^{(i)}\right] + \sum_{i=1}^N(\lambda_i+v_i)\xi_i - \sum_{i=1}^N v_i\xi_i + \sum_{i=1}^N \lambda_i(1-\xi_i - t^{(i)}((\sum_{i=1}^N \lambda_i t^{(i)}x^{(i)})^T x^{(i)}+b))$$

$$= \frac{1}{2}\left[\sum_{i=1}^N \lambda_i t^{(i)}x^{(i)}\right]^T\left[\sum_{i=1}^N \lambda_i t^{(i)}x^{(i)}\right] + \sum_{i=1}^N \lambda_i - \left(\sum_{i=1}^N \lambda_i t^{(i)}x^{(i)}\right)^T\left(\sum_{i=1}^N \lambda_i t^{(i)}x^{(i)}\right)$$

$$= \sum_{i=1}^N \lambda_i - \frac{1}{2}\left[\sum_{i=1}^N \lambda_i t^{(i)}x^{(i)}\right]^T\left[\sum_{i=1}^N \lambda_i t^{(i)}x^{(i)}\right]$$

set $k(x,z) = x^T z$ $\Leftarrow$ $= \sum_{i=1}^N \lambda_i - \frac{1}{2}\sum_{i=1}^N\sum_{j=1}^N \lambda_i\lambda_j t^{(i)}t^{(j)} x^{(i)T}x^{(j)} = \sum_{i=1}^N \lambda_i - \frac{1}{2}\sum_{i=1}^N\sum_{j=1}^N \lambda_i\lambda_j t^{(i)}t^{(j)} k(x^{(i)},x^{(j)})$

$\therefore$ The dual problem is: $\min_\lambda \hat{L}(\lambda) = \sum_{i=1}^N \lambda_i - \frac{1}{2}\sum_{i=1}^N\sum_{j=1}^N \lambda_i\lambda_j t^{(i)}t^{(j)} k(x^{(i)},x^{(j)})$

subject to: $0 \leq \lambda_i \leq C$,

$$\sum_{i=1}^N \lambda_i t^{(i)} = 0$$

3. (a) $K(u,v) = (\langle u,v\rangle + 1)^4$

$= \langle u,v\rangle^4 + 4\langle u,v\rangle^3 + 6\langle u,v\rangle^2 + 4\langle u,v\rangle + 1$

For $d=3$, $u = (u_1, u_2, u_3)$, $v = (v_1, v_2, v_3)$

For general mode, $\langle u,v\rangle^p = (u_1v_1 + u_2v_2 + u_3v_3)^p = \sum_{j_1, j_2, j_3, \Sigma j = p} \binom{p}{j_1, j_2, j_3} (u_1v_1)^{j_1}(u_2v_2)^{j_2}(u_3v_3)^{j_3}$

$\therefore \Phi_p(x) = [\cdots, \binom{p}{j_1 j_2 j_3}^{\frac{1}{2}} (u_1)^{j_1}(u_2)^{j_2}(u_3)^{j_3}, \cdots]$ (*)

$p=4$, $\Phi_4(u) = [u_1^4, u_2^4, u_3^4, 2u_1^3u_2, 2u_1^3u_3, 2u_1u_2^3, 2u_2^3u_3, 2u_1u_3^3,$
$2u_2u_3^3, \sqrt{6}u_1^2u_2^2, \sqrt{6}u_1^2u_3^2, \sqrt{6}u_2^2u_3^2, 2\sqrt{3}u_1^2u_2u_3, 2\sqrt{3}u_1u_2^2u_3, 2\sqrt{3}u_1u_2u_3^2]$

$p=3$, $\Phi_3(u) = [u_1^3, u_2^3, u_3^3, \sqrt{3}u_1^2u_2, \sqrt{3}u_1u_2^2, \sqrt{3}u_1^2u_3, \sqrt{3}u_2^2u_3, \sqrt{3}u_1u_3^2, \sqrt{3}u_2u_3^2, \sqrt{6}u_1u_2u_3]$

$p=2$, $\Phi_2(u) = [u_1^2, u_2^2, u_3^2, \sqrt{2}u_1u_2, \sqrt{2}u_1u_3, \sqrt{2}u_1u_3]$

$p=1$, $\Phi_1(u) = [u_1, u_2, u_3]$

Then for u, $\phi(u) = (\Phi_4(u), 2\Phi_3(u), \sqrt{6}\Phi_2(u), 2\Phi_1(u), 1)$

Thus $k(u,v) = \phi(u)^T\phi(v)$

For arbitrary dimension d, based on the conclusion (*) above,

$p=4$, $\Phi_4(u) = [u_1^4, \cdots u_d^4, 2u_1^3u_2, \cdots, \sqrt{6}u_1^2u_2^2, \cdots\cdots 2\sqrt{3}u_1^2u_2u_3 \cdots\cdots]$

$p=3$, $\Phi_3(u) = [u_1^3, \cdots u_d^3, \sqrt{3}u_1^2u_2 \cdots, \sqrt{6}u_1u_2u_3, \cdots]$

$p=2$, $\Phi_2(u) = [u_1^2, \cdots u_d^2, \sqrt{2}u_1v_2, \cdots\cdots]$

$p=1$, $\Phi_1(u) = [u_1, \cdots u_d]$.

Then for u, $\phi(u) = (\Phi_4(u), 2\Phi_3(u), \sqrt{6}\Phi_2(u), 2\Phi_1(u), 1)$

satisfying $k(u,v) = \phi(u)^T\phi(v)$

(b) From the definition of Gram matrix k, for a positive-definite kernel, k must be PSD, $k_i = \phi^{(i)}(x)\,\phi^{(i)}(z)$ is a Gram matrix, for $\forall\, a \in \mathbb{R}^n$, $a^T k_i a \geq 0$

(i) It is kernel. Since k_i is kernel, for $\forall a \in \mathbb{R}^n$,

$a^T k_1 a \geq 0$, $a^T k_2 a \geq 0 \Rightarrow a^T k a = a^T(k_1 + k_2)a = a^T k_1 a + a^T k_2 a \geq 0$

Thus k is kernel.

(ii) Not kernel. Set $k_2 = 2k_1$, then for $\forall a \in \mathbb{R}^n$, $a^T k_1 a \geq 0$

while $a^T k a = a^T(k_1 - 2k_1)a = -a^T k_1 a \leq 0$

(iii) It is kernel. For $\forall\, m \in \mathbb{R}^n$, $m^T k_1 m \geq 0$, then for $a > 0$

$m^T k m = m^T(a k_1)m = a(m^T k_1 m) \geq 0$

(iv) It is kernel. $k(x,z) = k_1(x,z)\,k_2(x,z)$

$$= \sum_i \phi_i^{(1)}(x)\,\phi_i^{(1)}(z) \cdot \sum_j \phi_j^{(2)}(x)\,\phi_j^{(2)}(z)$$

$$= \sum_i \sum_j \phi_i^{(1)}(x)\,\phi_i^{(1)}(z)\,\phi_j^{(2)}(x)\,\phi_j^{(2)}(z)$$

$$= \sum_i \sum_j [\phi_i^{(1)}(x)\,\phi_{(j)}^{(2)}(x)] \cdot [\phi_{(i)}^{(1)}(z)\,\phi_{(j)}^{(2)}(z)] = \sum_{i,j} \Phi_{i,j}(x)\,\Phi_{i,j}(z)$$

Thus k can be written as $k(x,z) = \Phi(x)^T\,\Phi(z) \Rightarrow k$ is knernel.

(v) It is kernel. Let $\varphi(x) = f(x)$, $f: \mathbb{R}^D \to \mathbb{R}$

$\therefore \varphi^T(x) = f^T(x) = f(x) = \varphi(x)$

$\therefore k(x,z) = \varphi(x) \cdot \varphi(z) = \varphi(x)^T \varphi(z) \Rightarrow k$ is kernel.

(vi) It is kernel: $k(x,z) = a_1 k_1(x,z) + \cdots + a_p k_1^p(x,z)$

From (iv), $k_1^p(x,z)$ is kernel, From (iii), $a_p k_1^p(x,z)$ is kernel.

Then From (i), $k(x,z) = \sum_i a_i k_1^i(x,z)$ is kernel.

(vii) $k(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right) = \exp\left(-\frac{x^T x}{2\sigma^2}\right)\exp\left(\frac{x^T z}{\sigma^2}\right)\exp\left(-\frac{z^T z}{2\sigma^2}\right)$

It can be written as $k(x,z) = f(x)\,\exp\left(\frac{x^T z}{\sigma^2}\right) f(z)$

By Taylor expansion, $\exp\left(\frac{x^T z}{\sigma^2}\right) = \sum_{i}^{\infty} \frac{\left(\frac{x^T z}{\sigma^2}\right)^n}{n!}$,

Noticing $x^T z$ is a kernel, From (vi), $\sum^{\infty} \left(\frac{x^T z}{\sigma^2}\right)^n / n!$ is a kernel.

Thus we can write $\exp\left(\frac{x^T z}{\sigma^2}\right)$ as $\phi(x)^T \phi(z)$, Then

$$k(x,z) = f(x) \langle \phi(x), \phi(z) \rangle f(z) = \langle f(x)\phi(x), f(z)\phi(z) \rangle$$

4(a)

$$P^{-1} - P^{-1}Q(R^{-1} + SP^{-1}Q)^{-1}SP^{-1}$$

$$= P^{-1}Q(R^{-1}+SP^{-1}Q)^{-1}(R^{-1}+SP^{-1}Q)Q^{-1} - P^{-1}Q(R^{-1}+SP^{-1}Q)^{-1}SP^{-1}$$

$$= P^{-1}Q(R^{-1}+SP^{-1}Q)^{-1}(R^{-1}+SP^{-1}Q - SP^{-1}Q)Q^{-1}$$

$$= P^{-1}Q(R^{-1}+SP^{-1}Q)^{-1}R^{-1}Q^{-1}$$

From the description of the question, we have

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

with $w = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T t$

Let $P = \lambda I$, $Q = \Phi^T$, $R = I$, $S = \Phi$, we have

$$(\lambda I + \Phi^T\Phi)^{-1} = (\lambda I)^{-1}\Phi^T(I^{-1} + \Phi(\lambda I)^{-1}\Phi^T)^{-1}(I)^{-1}(\Phi^T)^{-1}$$

$$(\lambda I + \Phi^T\Phi)^{-1} = \frac{1}{\lambda}\Phi(I + \frac{1}{\lambda}\Phi\Phi^T)^{-1}(\Phi^T)^{-1}$$

$$(\lambda I + \Phi^T\Phi)^{-1}\Phi^T = \Phi^T(\lambda I + \Phi\Phi^T)^{-1}$$

$$\Rightarrow w = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T t = \Phi^T(\lambda I + \Phi\Phi^T)^{-1}t = \Phi^T a,$$

where $a = (\lambda I + \Phi\Phi^T)^{-1}t$

$$f(x) = w^T\phi(x) = (\Phi^T a)^T\phi(x) = a^T\Phi(a)\phi(x) = a^T k(x)$$

where $k(x) = \Phi\phi(x) = [k(x_1, x), \dots k(x_n, x)]^T$

$$E(w) = (\Phi w - t)^T(\Phi w - t) + \lambda w^T w$$

$$= w^T\Phi^T\Phi w - 2t^T\Phi w + t^T t + \lambda a^T\Phi\Phi^T a$$

$$= a^T K K a - 2t^T K a + t^T t + \lambda a^T K a$$