EECS 545 – Machine Learning - Homework #3

Mina Jafari
group member: Joshua Kammeraad

1) **Question 1.**

(a)

$$
\begin{aligned}
&t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b) \geq 1-\xi_i \\
&1-t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b) \leq \xi_i \text{ and } \xi_i \geq 0 \implies \\
&\xi_i = \max\left(0, 1-t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b)\right) \\
&\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \text{ , substitute for } \xi_i \\
&\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\max\left(0, 1-t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b)\right) \implies (1) \equiv (2)
\end{aligned}
$$

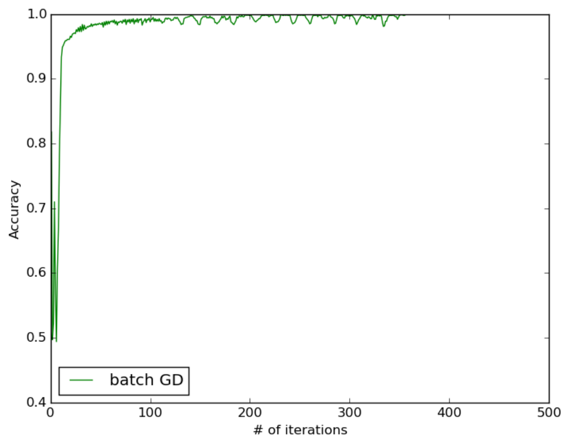
(b)

$$
\begin{aligned}
&\text{if } \xi_i^* > 0 \implies \xi_i^* = 1-t^{(i)}(\mathbf{w}^{*T}\mathbf{x}^{(i)}+b^*) \rightarrow \mathbf{w}^{*T}\mathbf{x}^{(i)} = \frac{-\xi_i^*+1}{t^{(i)}} - b^* \\
&\text{plane: } t^{(i)}((\mathbf{w}^*)^T\mathbf{x}+b^*)-1=0 \rightarrow \mathbf{w}^{*T}\mathbf{x} = \frac{1}{t^{(i)}} - b^* \\
&D = \frac{\|(\mathbf{x}^{(i)}-\mathbf{x})\cdot\mathbf{w}^*\|}{\|\mathbf{w}^*\|} = \frac{|\,(\mathbf{x}^{(i)})^\top\mathbf{w}^* - \mathbf{x}^\top\mathbf{w}^*\,|}{\|\mathbf{w}^*\|} = \frac{|\,\mathbf{w}^{*\top}\mathbf{x}^{(i)} - \mathbf{w}^{*\top}\mathbf{x}\,|}{\|\mathbf{w}^*\|} = \frac{|\,\frac{-\xi_i^*+1}{t^{(i)}} - b^* - \frac{1}{t^{(i)}} - b^*\,|}{\|\mathbf{w}^*\|} \\
&D = |\,\frac{-\xi_i}{t^{(i)}\|\mathbf{w}^*\|}\,|
\end{aligned}
$$

(c)

$$
\begin{aligned}
\nabla_{\mathbf{w}}E(\mathbf{w},b) &= \nabla_{\mathbf{w}}\frac{1}{2}\|\mathbf{w}\|^2 + \nabla_{\mathbf{w}}C\sum_{i=1}^{N}\max\left(0, 1-t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b)\right) \\
&= \mathbf{w} + C\sum_{i=1}^{N}\begin{cases} 0, & \text{if } t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b) \geq 1 \\ -t^{(i)}\mathbf{x}^{(i)}, & \text{if } t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b) < 1 \end{cases} \\
\frac{\partial}{\partial b}E(\mathbf{w},b) &= \frac{\partial}{\partial b}\frac{1}{2}\|\mathbf{w}\|^2 + \frac{\partial}{\partial b}C\sum_{i=1}^{N}\max\left(0, 1-t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b)\right) \\
&= C\sum_{i=1}^{N}\begin{cases} 0, & \text{if } t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b) \geq 1 \\ -t^{(i)}, & \text{if } t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b) < 1 \end{cases}
\end{aligned}
$$

(d)



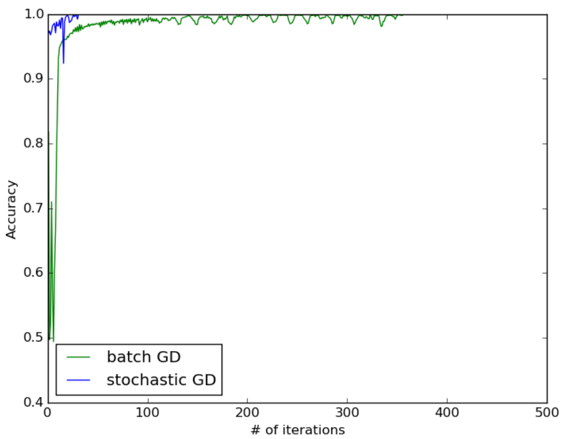

```
import numpy as np
from matplotlib import pyplot as plt

trainingData = np.loadtxt("digits_training_data.csv", delimiter=",")
B = np.loadtxt("digits_training_labels.csv", delimiter=",")

targetVector_train = np.zeros(B.shape)
i = 0
for elem in B:
    if elem == 4:
        targetVector_train[i] = -1
    elif elem == 9:
        targetVector_train[i] = 1
    i += 1

numOfRows =  trainingData.shape[0]
numOfCols = trainingData.shape[1]

wStar = np.zeros(numOfCols)
bStar = 0.0
wGrad = np.zeros(numOfCols)
bGrad = 0.0
numOfIterations = 500
pred = np.zeros(numOfIterations)

for j in range(0, numOfIterations):
    sumW = np.zeros(numOfCols)
    sumB = 0.0
    for k in range(0, numOfRows):
        test = np.add( np.dot(targetVector_train[k], np.dot(wStar, trainingData[k])), \
                np.dot(targetVector_train[k], bStar) )
        if test < 1.0:
            tempVec = np.dot(np.dot(targetVector_train[k], trainingData[k]), -3)
            tempB = targetVector_train[k] * -3
        else:
            tempVec = np.zeros(numOfCols)
            tempB = 0.0
        sumW = np.add(sumW, tempVec)
        sumB = sumB + tempB
    wGrad = np.add(wStar, sumW)
    bGrad = sumB

    wStar -= (0.001 / (1 + j*0.001)) * wGrad
    bStar -= (0.001 / (1 + j*0.001)) * bGrad
    correct = 0.0
    target = np.zeros(numOfRows)
    for m in range(0, numOfRows):
        target[m] = np.dot(wStar, trainingData[m]) + bStar
        if target[m] * targetVector_train[m] > 0:
            correct += 1
    perCorr = correct / numOfRows
    pred[j] = perCorr

x = range(1, 501)
plt.plot(x, pred, 'g', label="batch_GD")
```

```
plt.ylabel('Accuracy')
plt.xlabel('#_of_iterations')
plt.legend(loc=3)
plt.show()
```

(e)

$$\begin{aligned}
\nabla_{\mathbf{w}} E^{(i)}(\mathbf{w}, b) &= \nabla_{\mathbf{w}} \frac{1}{2N}\|\mathbf{w}\|^2 + \nabla_{\mathbf{w}} C \max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right) \\
&= \frac{1}{N}\mathbf{w} + C \begin{cases} 0, & \text{if } t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 \\ -t^{(i)}\mathbf{x}^{(i)}, & \text{if } t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1 \end{cases} \\
\frac{\partial}{\partial b} E^{(i)}(\mathbf{w}, b) &= \frac{\partial}{\partial b}\frac{1}{2N}\|\mathbf{w}\|^2 + \frac{\partial}{\partial b} C \max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right) \\
&= C \begin{cases} 0, & \text{if } t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 \\ -t^{(i)}, & \text{if } t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1 \end{cases}
\end{aligned}$$

(f)




```
import numpy as np
from matplotlib import pyplot as plt

trainingData = np.loadtxt("digits_training_data.csv", delimiter=",")
B = np.loadtxt("digits_training_labels.csv", delimiter=",")

targetVector_train = np.zeros(B.shape)
i = 0
for elem in B:
    if elem == 4:
        targetVector_train[i] = -1
    elif elem == 9:
        targetVector_train[i] = 1
    i += 1

numOfRows =  trainingData.shape[0]
numOfCols = trainingData.shape[1]

wStar = np.zeros(numOfCols)
bStar = 0.0
wGrad = np.zeros(numOfCols)
bGrad = 0.0
numOfIterations = 500
pred = np.zeros(numOfIterations)

for j in range(0, numOfIterations):
    array = np.random.permutation(numOfRows)
    for k in array:
```

```python
        test = np.add( np.dot(targetVector_train[k], np.dot(wStar, trainingData[k])), \
               np.dot(targetVector_train[k], bStar) )
        if test < 1.0:
            tempVec = np.dot(np.dot(targetVector_train[k], trainingData[k]), -3)
            tempB = targetVector_train[k] * -3
        else:
            tempVec = np.zeros(numOfCols)
            tempB = 0.0
        wGrad = np.add(np.dot(1.0/numOfRows, wStar), tempVec)
        bGrad = tempB

        wStar -= (0.001 / (1 + j*0.001)) * wGrad
        bStar -= (0.001 / (1 + j*0.001)) * bGrad
    correct = 0.0
    target = np.zeros(numOfRows)
    for m in range(0, numOfRows):
        target[m] = np.dot(wStar, trainingData[m]) + bStar
        if target[m] * targetVector_train[m] > 0:
            correct += 1
    perCorr = correct / numOfRows
    pred[j] = perCorr

x = range(1, 501)
plt.plot(x, pred, 'b', label="stochastic gradient descent")
plt.ylabel('Accuracy')
plt.xlabel('# of iterations')
plt.legend(loc=3)
plt.show()
```

(g)

Stochastic GD converges after about 30 iterations while it takes about 350 steps for the batch GD to converge, so in this case, it's roughly 12 times faster. Based on the plot!

(h)

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

$$g_1 : \xi_i \geq 0, g_2 : t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) - 1 + \xi_i \geq 0$$

$$\mathcal{L}(\mathbf{w}, b, \xi_i, \alpha, \mu) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\big(t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) - 1 + \xi_i\big) - \sum_{i=1}^{N}\mu_i\xi_i$$

$$\nabla_{\mathbf{w}}\mathcal{L} = \mathbf{w} - \sum_{i=1}^{N}\alpha_i t^{(i)}\mathbf{x}^{(i)} = 0 \to \mathbf{w} = \sum_{i=1}^{N}\alpha_i t^{(i)}\mathbf{x}^{(i)}$$

$$\frac{\partial\mathcal{L}}{\partial b} = -\sum_{i=1}^{N}\alpha_i t^{(i)} = 0$$

$$\frac{\partial\mathcal{L}}{\partial\xi_i} = C - \alpha_i - \mu_i = 0 \to \alpha_i = C - \mu_i$$

$$\mathcal{L}_{\mathcal{D}}(\alpha, \mu) = \frac{1}{2}\left(\sum_{i=1}^{N}\alpha_i t^{(i)}\mathbf{x}^{(i)}\right)^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\big(t^{(i)}\sum_{j=1}^{N}\alpha_j t^{(j)}\mathbf{x}^{(j)}\mathbf{x}^{(i)} + t^{(i)}b - 1 + \xi_i\big)$$

$$-\sum_{i=1}^{N}\mu_i\xi_i$$

$$= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j t^{(i)}t^{(j)}(\mathbf{x}^{(i)})^\top\mathbf{x}^{(j)} + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j t^{(i)}t^{(j)}(\mathbf{x}^{(i)})^\top\mathbf{x}^{(j)}$$

$$-\sum_{i=1}^{N}\alpha_i t^{(i)}b - \sum_{i=1}^{N}\alpha_i - \sum_{i=1}^{N}\alpha_i\xi_i - \sum_{i=1}^{N}\mu_i\xi_i$$

substitute α_i by $C - \mu_i$ in second to the last term $\implies$

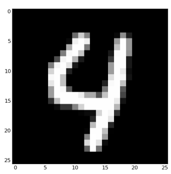
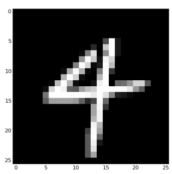
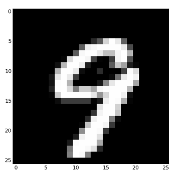
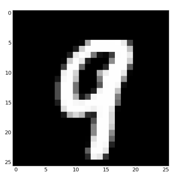
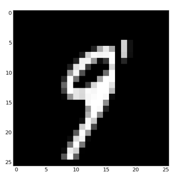
$$= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j t^{(i)}t^{(j)}(\mathbf{x}^{(i)})^\top\mathbf{x}^{(j)} - \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j t^{(i)}t^{(j)}(\mathbf{x}^{(i)})^\top\mathbf{x}^{(j)} - b\sum_{i=1}^{N}\alpha_i t^{(i)} - \sum_{i=1}^{N}\alpha_i$$

$$\mathcal{L}_{\mathcal{D}}(\alpha, \mu) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j t^{(i)}t^{(j)}(\mathbf{x}^{(i)})^\top\mathbf{x}^{(j)} - \sum_{i=1}^{N}\alpha_i$$

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}^{(i)})^\top\mathbf{x}^{(j)} \implies \mathcal{L}_{\mathcal{D}}(\alpha, \mu) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j t^{(i)}t^{(j)}k(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{N}\alpha_i$$

(i)

Training accuracy: 99.9%
Test accuracy: 98%
Parameters: C=1.0, gamma=5×10^{-7}, tol=0.0001

```python
import numpy as np
from sklearn.svm import SVC
from matplotlib import pyplot as plt

trainingData = np.loadtxt("digits_training_data.csv", delimiter=",")
A = np.loadtxt("digits_training_labels.csv", delimiter=",")
testData = np.loadtxt("digits_test_data.csv", delimiter=",")
B = np.loadtxt("digits_test_labels.csv", delimiter=",")

targetVector_train = np.zeros(A.shape)
i = 0
for elem in A:
    if elem == 4:
        targetVector_train[i] = -1
    elif elem == 9:
        targetVector_train[i] = 1
    i += 1

targetVector_test = np.zeros(B.shape)
i = 0
for elem in B:
    if elem == 4:
        targetVector_test[i] = -1
    elif elem == 9:
        targetVector_test[i] = 1
    i += 1

clf = SVC(C=1.0, kernel='rbf', gamma=0.0000005, tol=0.001)
clf.fit(trainingData, targetVector_train)
target_pred = clf.predict(testData)
print clf.score(trainingData, targetVector_train)
print clf.score(testData, targetVector_test)
for i in range(0, targetVector_test.shape[0]):
    if targetVector_test[i] != target_pred[i]:
        print i
```

(j)

2) **Question 2.**

```
import numpy as np
from sklearn.svm import SVC

trainingData = np.loadtxt("trainingData", delimiter=",")
targetVector_train = np.loadtxt("trainingLabels", delimiter=",")
testData = np.loadtxt("testData", delimiter=",")

clf = SVC(C=1.0, kernel='rbf', gamma=0.0000005, tol=0.001)
clf.fit(trainingData, targetVector_train)
target_pred = clf.predict(testData)
f = open('outPut', 'w')
f.write("id,category\n")
for i in range(1, target_pred.shape[0]+1):
        f.write(str(i))
        f.write(",")
        f.write(str(int(target_pred[i-1])))
        f.write("\n")
f.close()
```

3) **Question 3.**

(a)

$$\begin{aligned}
k(\mathbf{u}, \mathbf{v}) &= (\langle \mathbf{u}, \mathbf{v} \rangle + 1)^4 \\
&= (\mathbf{u}^\top \mathbf{v} + 1)^4 \\
&= (\mathbf{u}^\top \mathbf{v})^4 + 4(\mathbf{u}^\top \mathbf{v})^3 + 6(\mathbf{u}^\top \mathbf{v})^2 + 4(\mathbf{u}^\top \mathbf{v}) + 1 \\
&= \big(\sum_{i=1}^{d} \mathbf{u}_i \mathbf{v}_i\big)^4 + 4\big(\sum_{i=1}^{d} \mathbf{u}_i \mathbf{v}_i\big)^3 + 6\big(\sum_{i=1}^{d} \mathbf{u}_i \mathbf{v}_i\big)^2 + 4\big(\sum_{i=1}^{d} \mathbf{u}_i \mathbf{v}_i\big) + 1 \\
&= \sum_{k_1+...+k_m=4} \binom{4}{k_1, ..., k_m} \prod_{1 \le t \le m} (\mathbf{u}_t \mathbf{v}_t)^{k_t} + 4 \sum_{k_1+...+k_m=3} \binom{3}{k_1, ..., k_m} \prod_{1 \le t \le m} (\mathbf{u}_t \mathbf{v}_t)^{k_t} \\
&+ 6 \sum_{k_1+...+k_m=2} \binom{2}{k_1, ..., k_m} \prod_{1 \le t \le m} (\mathbf{u}_t \mathbf{v}_t)^{k_t} + 4\big(\sum_{i=1}^{d} \mathbf{u}_i \mathbf{v}_i\big) + 1 \\
&= \phi(\mathbf{u})^\top \phi(\mathbf{v}) \\
&\implies \phi(\mathbf{x}) = \sqrt{\sum_{k_1+...+k_m=4} \binom{4}{k_1, ..., k_m}} \prod_{1 \le t \le m} (\mathbf{x}_t)^{k_t} \\
&+ \sqrt{4 \sum_{k_1+...+k_m=3} \binom{3}{k_1, ..., k_m}} \prod_{1 \le t \le m} (\mathbf{x}_t)^{k_t} + \sqrt{6 \sum_{k_1+...+k_m=2} \binom{2}{k_1, ..., k_m}} \prod_{1 \le t \le m} (\mathbf{x}_t)^{k_t} \\
&+ 2\big(\sum_{i=1}^{d} \mathbf{x}_i\big) + 1
\end{aligned}$$

(b)

(i)

PD

$$\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= k_1(\mathbf{x}, \mathbf{z}) + k_2(\mathbf{x}, \mathbf{z}) \\
&= \phi_1(\mathbf{x})^\top \phi_1(\mathbf{z}) + \phi_2(\mathbf{x})^\top \phi_2(\mathbf{z}) \\
&= \sum_{i=1}^{m} \phi_1(\mathbf{x})_i \phi_1(\mathbf{z})_i + \sum_{j=1}^{n} \phi_2(\mathbf{x})_j \phi_2(\mathbf{z})_j \\
&= \sum_{i=1}^{m} \phi_1(\mathbf{x}_1)_i \phi_1(\mathbf{z}_1)_i + \ldots + \sum_{i=1}^{m} \phi_1(\mathbf{x}_D)_i \phi_1(\mathbf{z}_D)_i \\
&+ \sum_{j=1}^{n} \phi_2(\mathbf{x}_1)_j \phi_2(\mathbf{z}_1)_j + \ldots + \sum_{j=1}^{n} \phi_2(\mathbf{x}_D)_j \phi_2(\mathbf{z}_D)_j \\
&\implies \phi(\mathbf{y}) = \{\phi_1(\mathbf{y}), \phi_2(\mathbf{y})\}
\end{aligned}$$

(ii)

Not PD

$$\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= k_1(\mathbf{x}, \mathbf{z}) - k_2(\mathbf{x}, \mathbf{z}) \\
&= \mathbf{x}^\top \mathbf{z} - (\mathbf{x}^\top \mathbf{z})^2 \\
&= 2 \times 1 - (2 \times 1)^2 = -2
\end{aligned}$$

(iii)

PD

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= a k_1(\mathbf{x}, \mathbf{z}) \\
&= a \phi_1(\mathbf{x})^\top \phi_1(\mathbf{z}) \\
&= a \sum_{i=1}^{m} \phi_1(\mathbf{x})_i \phi_1(\mathbf{z})_i \\
&= \sum_{i=1}^{m} \sqrt{a} \phi_1(\mathbf{x})_i \sqrt{a} \phi_1(\mathbf{z})_i \\
&= \left(\sqrt{a} \phi_1(\mathbf{x})\right)^\top \left(\sqrt{a} \phi_1(\mathbf{z})\right) \\
&\implies \phi(\mathbf{y}) = \sqrt{a} \phi_1(\mathbf{y})
\end{aligned}
$$

(iv)

PD

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= k_1(\mathbf{x}, \mathbf{z}) k_2(\mathbf{x}, \mathbf{z}) \\
&= \phi_1(\mathbf{x})^\top \phi_1(\mathbf{z}) \phi_2(\mathbf{x})^\top \phi_2(\mathbf{z}) \\
&= \sum_{i=1}^{m} \phi_1(\mathbf{x})_i \phi_1(\mathbf{z})_i \sum_{j=1}^{n} \phi_2(\mathbf{x})_j \phi_2(\mathbf{z})_j \\
&= \sum_{i=1}^{m} \sum_{j=1}^{n} \phi_1(\mathbf{x})_i \phi_1(\mathbf{z})_i \phi_2(\mathbf{x})_j \phi_2(\mathbf{z})_j \\
&\implies \phi(\mathbf{y}) = \phi_1(\mathbf{y}) \phi_2(\mathbf{y})
\end{aligned}
$$

(v)

PD

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= f(\mathbf{x}) f(\mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z}) \\
&\text{the output of } f(\mathbf{y}) \text{ is a scalar } \implies f(\mathbf{y}) f(\mathbf{y}) \text{ is an inner product} \\
\phi(\mathbf{y}) &= f(\mathbf{y})
\end{aligned}
$$

(vi)

PD

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= c_0 + c_1 k_1(\mathbf{x}, \mathbf{z}) + c_2 (k_1(\mathbf{x}, \mathbf{z}))^2 + \cdots + c_n (k_1(\mathbf{x}, \mathbf{z}))^n \\
k_1(\mathbf{x}, \mathbf{z}) &= \phi(\mathbf{x})^\top \phi(\mathbf{z}) \\
k(\mathbf{x}, \mathbf{z}) &= c_0 + c_1 \phi(\mathbf{x})^\top \phi(\mathbf{z}) + c_2 (\phi(\mathbf{x})^\top \phi(\mathbf{z}))^2 + \cdots + c_n (\phi(\mathbf{x})^\top \phi(\mathbf{z}))^n \\
&= c_0 + \sqrt{c_1} \phi(\mathbf{x})^\top \sqrt{c_1} \phi(\mathbf{z}) + \sqrt{c_2} (\phi(\mathbf{x})^\top)^2 \sqrt{c_2} (\phi(\mathbf{z}))^2 + \cdots + \sqrt{c_n} (\phi(\mathbf{x})^\top)^n \sqrt{c_n} (\phi(\mathbf{z}))^n \\
&\implies \phi(\mathbf{y}) = \{\sqrt{c_0}, \sqrt{c_1} \phi(\mathbf{y}), \sqrt{c_2} (\phi(\mathbf{y}))^2, \cdots, \sqrt{c_n} (\phi(\mathbf{x}))^n\}
\end{aligned}
$$

(vii)

$$
\begin{aligned}
e^x &= \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \\
k(\mathbf{x}, \mathbf{z}) &= \exp\left(\frac{-\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}\right) = \exp\left(\frac{-(\mathbf{x}-\mathbf{z})(\mathbf{x}-\mathbf{z})}{2\sigma^2}\right) \\
&= \exp\left(\frac{-\mathbf{x}^2}{2\sigma^2}\right) \exp\left(\frac{-\mathbf{z}^2}{2\sigma^2}\right) \exp\left(\frac{2\mathbf{x}\mathbf{z}}{2\sigma^2}\right) \\
&= \sum_{n=0}^{\infty} \frac{1}{n!}\left(\frac{-\mathbf{x}^2}{2\sigma^2}\right)^n \sum_{n=0}^{\infty} \frac{1}{n!}\left(\frac{-\mathbf{z}^2}{2\sigma^2}\right)^n \sum_{n=0}^{\infty} \frac{1}{n!}\left(\frac{\mathbf{x}\mathbf{z}}{\sigma^2}\right)^n \\
&= \sum_{n=0}^{\infty} \frac{1}{n!}\left(\frac{-\mathbf{x}^2}{2\sigma^2}\right)^n \sum_{n=0}^{\infty} \frac{1}{n!}\left(\frac{-\mathbf{z}^2}{2\sigma^2}\right)^n \sum_{n=0}^{\infty} \frac{1}{n!}\left(\frac{\mathbf{x}}{\sigma}\right)^n \left(\frac{\mathbf{z}}{\sigma}\right)^n \\
&\implies \phi(\mathbf{y}) = \sum_{n=0}^{\infty} \frac{1}{n!}\left(\frac{-\mathbf{y}^2}{2\sigma^2}\right)^n \sum_{n=0}^{\infty} \sqrt{\frac{1}{n!}}\left(\frac{\mathbf{y}}{\sigma}\right)^n
\end{aligned}
$$

4) **Question 4.**

(a)

$$
\begin{aligned}
\hat{f}(\mathbf{x}) &= \hat{\mathbf{w}}^T\phi(\mathbf{x}) = \mathbf{t}^T\Phi(\Phi^T\Phi+\lambda I)^{-1}\phi(\mathbf{x}) \\
(P+QRS)^{-1} &= P^{-1} - P^{-1}Q(R^{-1}+SP^{-1}Q)^{-1}SP^{-1}, P=\lambda I, Q=\Phi^\top, R=I, S=\Phi \\
\hat{f}(\mathbf{x}) &= \mathbf{t}^T\Phi\big[\lambda^{-1}I - \lambda^{-1}I\Phi^T(I+\Phi\lambda^{-1}I\Phi^T)^{-1}\Phi\lambda^{-1}I\big]\phi(\mathbf{x})\ , \mathbf{K}=\Phi\Phi^T, k(\mathbf{x}) = \Phi\phi(\mathbf{x}) \\
&= \mathbf{t}^T\big[\Phi\lambda^{-1}I - \Phi\lambda^{-1}I\Phi^T(I+\Phi\lambda^{-1}I\Phi^T)^{-1}\Phi\lambda^{-1}I\big]\phi(\mathbf{x}) \\
&= \mathbf{t}^T\big[\Phi\lambda^{-1}I\phi(\mathbf{x}) - \Phi\lambda^{-1}I\Phi^T(I+\Phi\lambda^{-1}I\Phi^T)^{-1}\Phi\lambda^{-1}I\phi(\mathbf{x})\big] \\
&= \mathbf{t}^T\big[\lambda^{-1}Ik(\mathbf{x}) - \lambda^{-1}I\mathbf{K}(I+\lambda^{-1}I\mathbf{K})^{-1}\lambda^{-1}Ik(\mathbf{x})\big] \\
&= \mathbf{t}^T\big[I - \lambda^{-1}I\mathbf{K}(I+\lambda^{-1}I\mathbf{K})^{-1}\big]\big(\lambda^{-1}Ik(\mathbf{x})\big) \\
&= \mathbf{t}^T\big[I - \lambda^{-1}I\mathbf{K}(I+\lambda^{-1}I\mathbf{K})^{-1}\big](I+\lambda^{-1}I\mathbf{K})(I+\lambda^{-1}I\mathbf{K})^{-1}\big(\lambda^{-1}Ik(\mathbf{x})\big) \\
&= \mathbf{t}^T\big[(I+\lambda^{-1}I\mathbf{K}) - \lambda^{-1}I\mathbf{K}\big](I+\lambda^{-1}I\mathbf{K})^{-1}\big(\lambda^{-1}Ik(\mathbf{x})\big) \\
&= \mathbf{t}^TI(I+\lambda^{-1}I\mathbf{K})^{-1}\big(\lambda^{-1}Ik(\mathbf{x})\big) \\
&= \big[\mathbf{t}^T(\lambda I+\mathbf{K})^{-1}k(\mathbf{x})\big]^T \\
&= k(\mathbf{x})(\lambda I+\mathbf{K})^{-1}\mathbf{t} \\
\hat{\mathbf{w}} &= (\Phi^T\Phi+\lambda I)^{-1}\Phi^T\mathbf{t} \\
&= \big[\lambda^{-1}I - \lambda^{-1}I\Phi^T(I+\Phi\lambda^{-1}I\Phi^T)^{-1}\Phi\lambda^{-1}I\big]\Phi^T\mathbf{t} \\
&= \big[\lambda^{-1}I\Phi^T - \lambda^{-1}I\Phi^T(I+\Phi\lambda^{-1}I\Phi^T)^{-1}\Phi\lambda^{-1}I\Phi^T\big]\mathbf{t} \\
&= \big[\lambda^{-1}I\Phi^T - \lambda^{-1}I\Phi^T(I+\lambda^{-1}I\mathbf{K})^{-1}\lambda^{-1}I\mathbf{K}\big]\mathbf{t} \\
&= (\lambda^{-1}I\Phi^T)(\lambda I+\mathbf{K})^{-1}(\lambda I+\mathbf{K})\big[I - (\lambda I+\mathbf{K})^{-1}\mathbf{K}\big]\mathbf{t} \\
&= (\lambda^{-1}I\Phi^T)(\lambda I+\mathbf{K})^{-1}\big[(\lambda I+\mathbf{K}) - \mathbf{K}\big]\mathbf{t} \\
&= (\lambda\lambda^{-1}I\Phi^T)(\lambda I+\mathbf{K})^{-1}\mathbf{t} \\
&= \Phi^T(\lambda I+\mathbf{K})^{-1}\mathbf{t} \\
&= \Phi^T\mathbf{a}
\end{aligned}
$$

(b)

(i)

RMSE = 10.8899710344

```
import numpy as np
from numpy import linalg
from sklearn.metrics import mean_squared_error
from math import exp
from sklearn.preprocessing import normalize

A = np.loadtxt("steel_composition_train.csv", delimiter=",", skiprows=1)
numOfRows =  A.shape[0]
numOfCols = A.shape[1]
trainingData_temp = np.delete(A, [0, numOfCols-1], 1)
targetVector_train = A[:,-1]

# normalization
trainingData = normalize(trainingData_temp, norm='l2', axis=1)

gramMatrix = np.zeros((numOfRows, numOfRows))
for i in range(0, numOfRows):
    for j in range(0, numOfRows):
        temp1 = np.dot(trainingData[i].transpose(), trainingData[j])
        gramMatrix[i][j] = np.power((temp1 + 1), 2)

eye = np.identity(numOfRows)
temp = np.add(gramMatrix, eye)
a = np.dot(linalg.pinv(temp), targetVector_train)
```

```
target_pred = np.zeros(numOfRows)
for i in range(0, numOfRows):
    target_pred[i] = np.dot(gramMatrix[:,i], a)
print np.sqrt(mean_squared_error(targetVector_train, target_pred))
```

(ii)

RMSE = 10.0965667265

```
import numpy as np
from numpy import linalg
from sklearn.metrics import mean_squared_error
from math import exp
from sklearn.preprocessing import normalize

A = np.loadtxt("steel_composition_train.csv", delimiter=",", skiprows=1)
numOfRows = A.shape[0]
numOfCols = A.shape[1]
trainingData_temp = np.delete(A, [0, numOfCols-1], 1)
targetVector_train = A[:,-1]

# normalization
trainingData = normalize(trainingData_temp, norm='l2', axis=1)

gramMatrix = np.zeros((numOfRows, numOfRows))
for i in range(0, numOfRows):
    for j in range(0, numOfRows):
        temp1 = np.dot(trainingData[i].transpose(), trainingData[j])
        gramMatrix[i][j] = np.power((temp1 + 1), 3)

eye = np.identity(numOfRows)
temp = np.add(gramMatrix, eye)
a = np.dot(linalg.pinv(temp), targetVector_train)
target_pred = np.zeros(numOfRows)
for i in range(0, numOfRows):
    target_pred[i] = np.dot(gramMatrix[:,i], a)
print np.sqrt(mean_squared_error(targetVector_train, target_pred))
```

(iii)

RMSE = 9.31271021985

```
import numpy as np
from numpy import linalg
from sklearn.metrics import mean_squared_error
from math import exp
from sklearn.preprocessing import normalize

A = np.loadtxt("steel_composition_train.csv", delimiter=",", skiprows=1)
numOfRows = A.shape[0]
numOfCols = A.shape[1]
trainingData_temp = np.delete(A, [0, numOfCols-1], 1)
targetVector_train = A[:,-1]

# normalization
trainingData = normalize(trainingData_temp, norm='l2', axis=1)

gramMatrix = np.zeros((numOfRows, numOfRows))
for i in range(0, numOfRows):
    for j in range(0, numOfRows):
        temp1 = np.dot(trainingData[i].transpose(), trainingData[j])
        gramMatrix[i][j] = np.power((temp1 + 1), 4)

eye = np.identity(numOfRows)
temp = np.add(gramMatrix, eye)
a = np.dot(linalg.pinv(temp), targetVector_train)
target_pred = np.zeros(numOfRows)
for i in range(0, numOfRows):
    target_pred[i] = np.dot(gramMatrix[:,i], a)
print np.sqrt(mean_squared_error(targetVector_train, target_pred))
```

(iv)

RMSE = 12.0441706978

```
import numpy as np
from numpy import linalg
from sklearn.metrics import mean_squared_error
from math import exp
```

```python
from sklearn.preprocessing import normalize

A = np.loadtxt("steel_composition_train.csv", delimiter=",", skiprows=1)
numOfRows =  A.shape[0]
numOfCols = A.shape[1]
trainingData_temp = np.delete(A, [0, numOfCols-1], 1)
targetVector_train = A[:,-1]

# normalization
trainingData = normalize(trainingData_temp, norm='l2', axis=1)

gramMatrix = np.zeros((numOfRows, numOfRows))
for i in range(0, numOfRows):
    for j in range(0, numOfRows):
        temp1 = np.subtract(trainingData[i], trainingData[j])
        temp2 = np.dot(temp1.transpose(), temp1)
        gramMatrix[i][j] = exp(temp2/-2)

eye = np.identity(numOfRows)
temp = np.add(gramMatrix, eye)
a = np.dot(linalg.pinv(temp), targetVector_train)
target_pred = np.zeros(numOfRows)
for i in range(0, numOfRows):
    target_pred[i] = np.dot(gramMatrix[:,i], a)
print np.sqrt(mean_squared_error(targetVector_train, target_pred))
```