

# **CmpE 462 Spring - 2020 Final Exam Report**

İbrahim Özgürçan Öztaş - 2016400198

July 4, 2020

# Contents

<b>1</b>	<b>Question 1:</b>	<b>II</b>
1.1	Part A: . . . . .	II
1.2	Part B: . . . . .	III
1.3	Part C: . . . . .	IV
<b>2</b>	<b>Question 2:</b>	<b>IX</b>
2.1	Part A: . . . . .	IX
2.2	Part B: . . . . .	XIII
<b>3</b>	<b>Question 3:</b>	<b>XVII</b>
<b>4</b>	<b>Question 4:</b>	<b>XVIII</b>
<b>5</b>	<b>Question 5:</b>	<b>XIX</b>
5.1	Part A: . . . . .	XIX
5.2	Part B: . . . . .	XIX
5.3	Part C: . . . . .	XX

# Chapter 1

## Question 1:

### 1.1 Part A:

If the classifier is fixed and yet the performance of the current classifier is low, than the current approach is not completely suitable with the current form of the data. There could be 2 crucial reasons for this to happen. First of all, the current kernel and its hyperparameters are not suitable to the current problem, hence the accuracy is 59%. However, it is explicitly stated that the current kernel and its hyperparameters are fixed, thus changing the current kernel to a different one with new hyperparameters are not the solution for this problem.

Another approach for the solution is that given data may not be completely random. Since the data can be collected from real events or created artificially, it is possible that there are a bunch of points in the given data which are not collected from real events. However, we can never be sure about that possibility, hence I'd like to focus on another perspective about this issue, which is the training and testing data sets are not randomly selected from the raw data set. To investigate further on this issue, I've merged the training and testing data into one complete set and applied several operations on the complete data set to create new training and testing data set.

## 1.2 Part B:

First of all, the initial sizes for training and testing data set is 1600 to 400, respectively. In addition to the former ratio, I've added a new partitioning ratio for training and testing data set, which is 9:1 rather than 4:1. Thus, I've now 2 types of size for the training and testing data set, which I hope it'll increase the accuracy. Afterwards, I've focused on outlier points in the training and testing data set, which can be quite a reason for the low performance of the classifier.

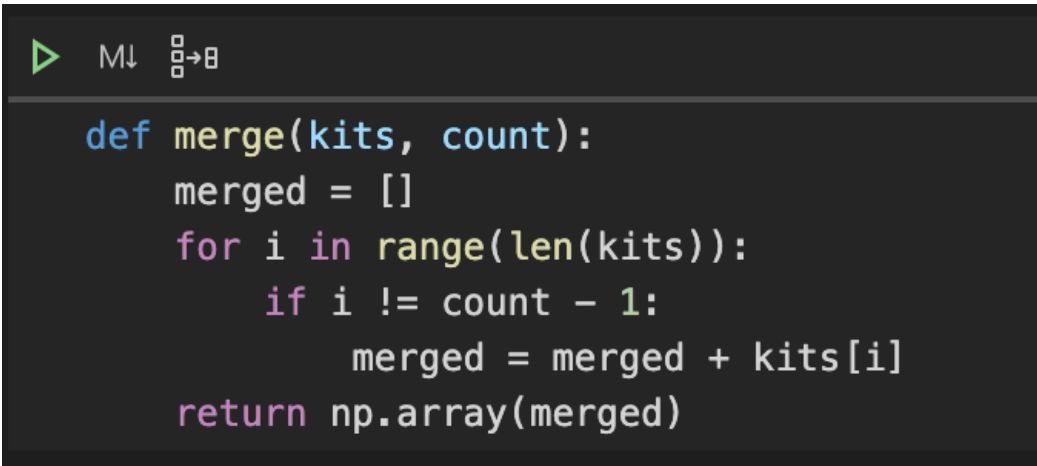
Then, I've implemented a random selection process to split the data into the desired number of partitions. At first, I've created the desired number of partitions as empty lists. Then, by selecting from 0 to 2000 randomly via random uniform distribution, I've got an index to select a point from the data. If the current index is selected before, it selects another index from the same random uniform distribution until the current index is not selected before. Hence, I've put the current value pointed by the index into a bucket by a relation which takes the iteration as input and outputs the modulo of the iteration by the number of partitions defined before. By shuffling all points in the data set, and re-partitioning them into two new sets, I believe that the classifier has higher accuracy than before.

To increase randomness and take control on the random values, I've parametrized the seed value and started my testing process. At first, I've used the same training:testing ratio as 4:1, which is initially given. Then, I've run my classification process 100 times with a different seed each, and collected the maximum accuracy of the classifier. The results are quite promising, since the initial value of accuracy is 0.59, the maximum value of the accuracy is 0.74, which happened when the seed parameter is 77 and the selected partition for testing is the last partition, which are the last 400 points out of 2000 points.

As a second trial, I've changed the training:testing ratio from 4:1 to 9:1, which means I've reduced the size of the testing data set to its 50%. Now, I've 1800 data points out of 2000 data points will be used for training the classifier and 200 data points out of 2000 data points will be used for testing purpose of the classifier. After changing the training:testing ratio, I've re-run the classification process 100 times with a different seed each, and

collected the maximum accuracy value of the classifier. The results are more promising than the first approach, since the initial value is 59%, the first trial resulted in with accuracy 0.74, and the second trial resulted in with accuracy 0.77 which happened when the seed value is 86 and the selected partition for testing is the third partition(400:600) out of ten partitions.

### 1.3 Part C:



```
def merge(kits, count):  
    merged = []  
    for i in range(len(kits)):  
        if i != count - 1:  
            merged = merged + kits[i]  
    return np.array(merged)
```

▶ M1 

```
def random_shuffle(data, label, seed):  
    np.random.seed(seed)  
  
    data_kits = []  
    data_kits.append([])  
    data_kits.append([])  
    data_kits.append([])  
    data_kits.append([])  
    data_kits.append([])  
  
    label_kits = []  
    label_kits.append([])  
    label_kits.append([])  
    label_kits.append([])  
    label_kits.append([])  
    label_kits.append([])  
  
    selected = []  
  
    for i in range(data.shape[0]):  
        index = np.random.randint(0, high=2000)  
        while(index in selected):  
            index = np.random.randint(0, high=2000)  
        data_kits[i%5].append(data[index])  
        label_kits[i%5].append(label[index])  
        selected.append(index)  
    return ( data_kits, label_kits )
```

▶ ML →B

```
max_acc = (0, 0, 0)

for j in range(0, 100):
    raw_kits = random_shuffle(raw_data, raw_label, j)
    raw_data_kits = raw_kits[0]
    raw_label_kits = raw_kits[1]

    for i in range(5):
        tr_data = merge(raw_data_kits, i+1)
        tr_label = merge(raw_label_kits, i+1)

        te_data = raw_data_kits[i]
        te_label = raw_label_kits[i]

        clf = svm.SVC(gamma=0.001, C=100.)
        clf.fit(tr_data, tr_label)
        y_pred = clf.predict(te_data)
        correct_prediction = np.equal(y_pred, te_label)
        accuracy = np.mean(correct_prediction.astype(np.float32))
        if max_acc[0] < accuracy:
            max_acc = (accuracy, j, i)

print(max_acc)

(0.7425, 77, 4)
```

▶ M↓ 

```
def random_shuffle_10(data, label, seed):
    np.random.seed(seed)

    data_kits = []
    data_kits.append([])
    data_kits.append([])
    data_kits.append([])
    data_kits.append([])
    data_kits.append([])
    data_kits.append([])
    data_kits.append([])
    data_kits.append([])
    data_kits.append([])
    data_kits.append([])

    label_kits = []
    label_kits.append([])
    label_kits.append([])
    label_kits.append([])
    label_kits.append([])
    label_kits.append([])
    label_kits.append([])
    label_kits.append([])
    label_kits.append([])
    label_kits.append([])
    label_kits.append([])

    selected = []

    for i in range(data.shape[0]):
        index = np.random.randint(0, high=2000)
        while(index in selected):
            index = np.random.randint(0, high=2000)
        data_kits[i%10].append(data[index])
        label_kits[i%10].append(label[index])
        selected.append(index)
    return ( data_kits, label_kits )
```



▶ MI  $\frac{0}{8} \rightarrow 8$

```
max_acc = (0, 0, 0)

for j in range(0, 100):

    raw_kits = random_shuffle_10(raw_data, raw_label, j)
    raw_data_kits = raw_kits[0]
    raw_label_kits = raw_kits[1]

    for i in range(10):
        tr_data = merge(raw_data_kits, i+1)
        tr_label = merge(raw_label_kits, i+1)

        te_data = raw_data_kits[i]
        te_label = raw_label_kits[i]

        clf = svm.SVC(gamma=0.001, C=100.)
        clf.fit(tr_data, tr_label)
        y_pred = clf.predict(te_data)
        correct_prediction = np.equal(y_pred, te_label)
        accuracy = np.mean(correct_prediction.astype(np.float32))
        if max_acc[0] < accuracy:
            max_acc = (accuracy, j, i)

print(max_acc)

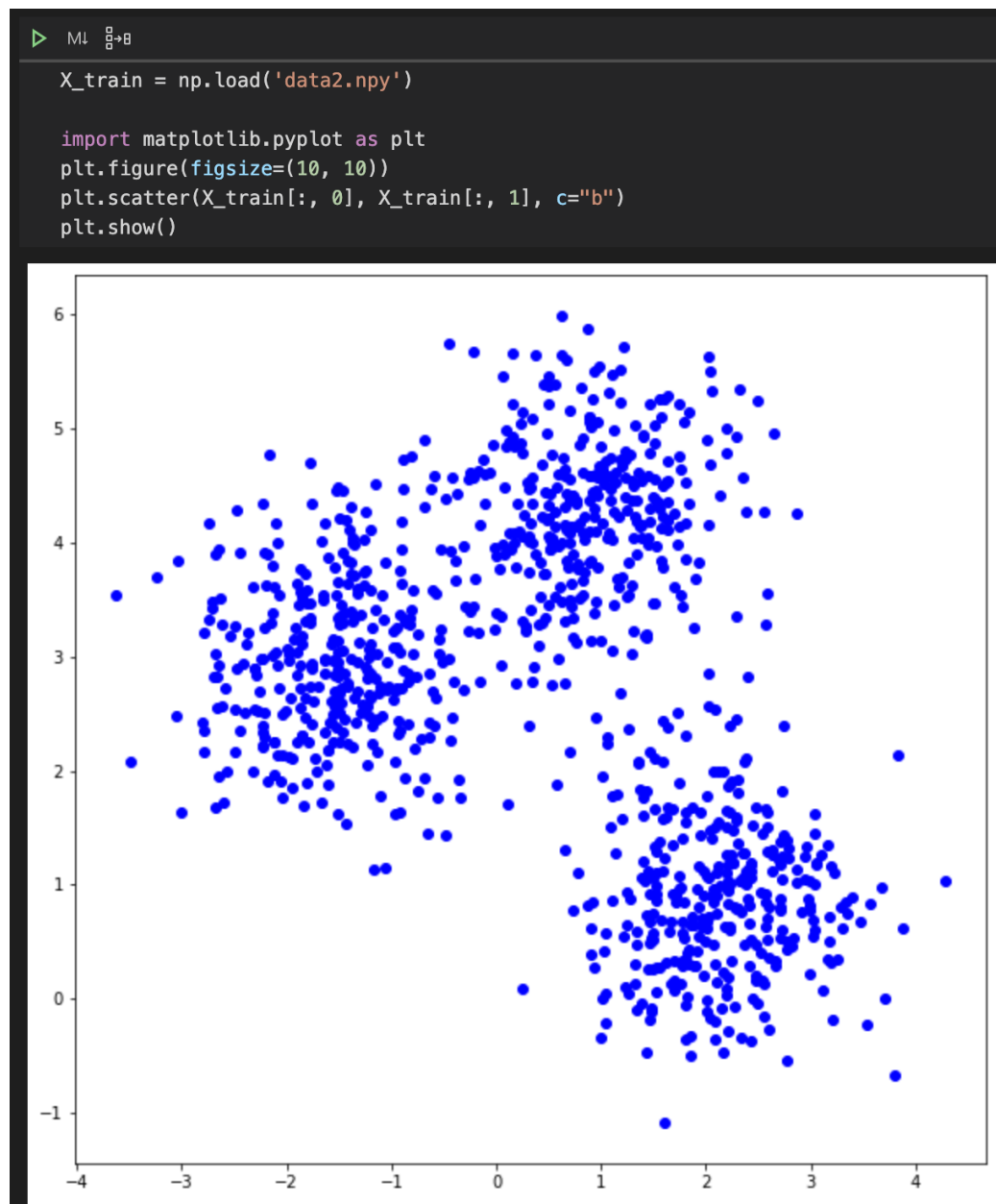
(0.77, 86, 2)
```

## Chapter 2

### Question 2:


#### 2.1 Part A:

In this question, since we've no information about the labels for the given data, I have used K-Means Clustering to obtain labels. I've decided to use K-Means Clustering algorithm after I've plotted the data by using pyplot in the matplotlib library of Python. After I've plotted the data, I've realized that there are 3 main clusters that the data points gather around. Thus, it seems useful to me to use K-Means Clustering algorithm to give all data points a label.




I've used our implementation of K-Means Clustering from Project 3 of this course and completed the labeling task. I've run the algorithm for 9 iterations and I've found out that at the seventh iteration, the result labels are not changing, thus the clusters are stabilized.


```

▶ ML 
def k_means(n, seed):
    np.random.seed(seed)
    min_x = min(X_train[:,0])
    min_y = min(X_train[:,1])
    range_x = (max(X_train[:,0]) - min(X_train[:,0]))
    range_y = (max(X_train[:,1]) - min(X_train[:,1]))
    c_1 = [min_x + range_x*np.random.rand(), min_y + range_y*np.random.rand()]
    c_2 = [min_x + range_x*np.random.rand(), min_y + range_y*np.random.rand()]
    c_3 = [min_x + range_x*np.random.rand(), min_y + range_y*np.random.rand()]
    for j in range(n):
        y_pred = np.zeros(len(X_train))
        for i in range(len(X_train)):
            c_1_dist = np.linalg.norm(X_train[i] - c_1)
            c_2_dist = np.linalg.norm(X_train[i] - c_2)
            c_3_dist = np.linalg.norm(X_train[i] - c_3)
            min_norm = min(c_1_dist, c_2_dist, c_3_dist)
            if (min_norm == c_1_dist):
                y_pred[i] = 0
            elif (min_norm == c_2_dist):
                y_pred[i] = 1
            else:
                y_pred[i] = 2
        c_1 = [np.sum(X_train[y_pred==0][:,0])/len(y_pred[y_pred==0]), np.sum(X_train[y_pred==0][:,1])/len(y_pred[y_pred==0])]
        c_2 = [np.sum(X_train[y_pred==1][:,0])/len(y_pred[y_pred==1]), np.sum(X_train[y_pred==1][:,1])/len(y_pred[y_pred==1])]
        c_3 = [np.sum(X_train[y_pred==2][:,0])/len(y_pred[y_pred==2]), np.sum(X_train[y_pred==2][:,1])/len(y_pred[y_pred==2])]
    return y_pred

```

```

▶ ML 
def same(it1, it2):
    isSame = True
    for i in range(1000):
        if output_list[it1-1][i] != output_list[it2-1][i]:
            isSame = False
    return isSame

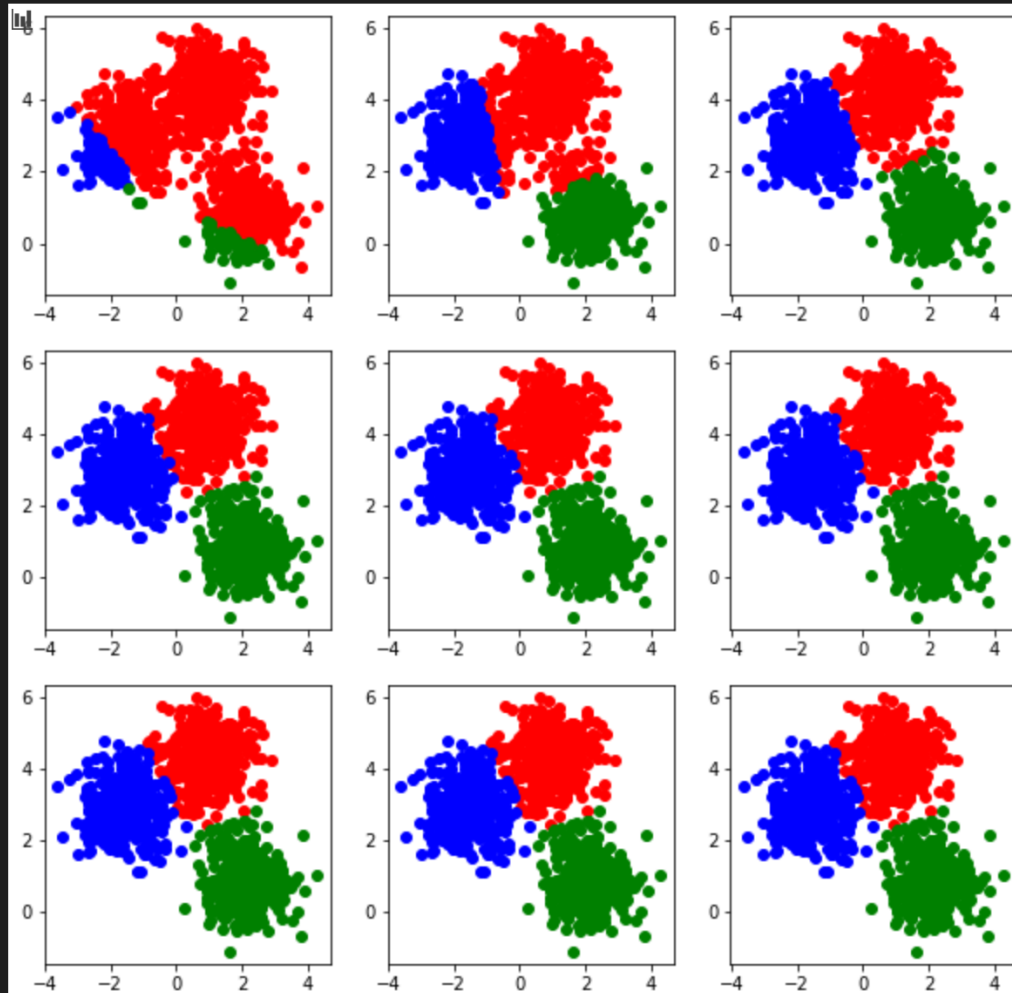
▶ ML 
print(same(1, 2))
print(same(2, 3))
print(same(3, 4))
print(same(4, 5))
print(same(5, 6))
print(same(6, 7))
print(same(7, 8)) # It seems that after the 7th iteration, the labels are stable.
print(same(8, 9))

False
False
False
False
False
False
True
True

```

ML 

```
fig = plt.figure(figsize=(10,10))
output_list = []
for i in range(9):
    a = k_means(i+1,1)
    output_list.append(a)
    fig.add_subplot(3,3,i+1)
    plt.scatter(X_train[a == 0][:,0],X_train[a == 0][:,1],c="r")
    plt.scatter(X_train[a == 1][:,0],X_train[a == 1][:,1],c="b")
    plt.scatter(X_train[a == 2][:,0],X_train[a == 2][:,1],c="g")
plt.show()
```




## 2.2 Part B:

In this part, I've used the given classifier and extracted the results as requested. Before training the classifier, I've shuffled the current data set and split the data and labels into 5 equal sized parts, which of each have 200 instance in it. And for each of the sets, I've chosen it as the testing part and the remaining four have become the training part. I've run the classification process 5 times and collected the results, which are listed as below:


```
▶ M↓ 8→8
def shuffle(X_train):
    data_kits = []
    data_kits.append([])
    data_kits.append([])
    data_kits.append([])
    data_kits.append([])
    data_kits.append([])
    for i in range(len(X_train)):
        if i % 5 == 0:
            data_kits[0].append(X_train[i])
        if i % 5 == 1:
            data_kits[1].append(X_train[i])
        if i % 5 == 2:
            data_kits[2].append(X_train[i])
        if i % 5 == 3:
            data_kits[3].append(X_train[i])
        if i % 5 == 4:
            data_kits[4].append(X_train[i])
    return data_kits

def merge(kits, count):
    merged = []
    for i in range(len(kits)):
        if i != count - 1:
            merged = merged + kits[i]
    return np.array(merged)
```


These are the functions that I've implemented to shuffle the data and merge the parts that are reserved for training process into one numpy array.

```
▶ M↓ 
labels = output_list[6]
shuf_data = shuffle(X_train)
shuf_label = shuffle(labels)
```

After implementation of these functions, I've shuffled the data and started my classification process.

```
▶ M↓ 
q2_train_data = merge(shuf_data, 1)
q2_train_label = merge(shuf_label, 1)

q2_test_data = shuf_data[0]
q2_test_label = shuf_label[0]

▶ M↓ 
import numpy as np
from sklearn import svm

clf = svm.SVC(gamma=0.001, C=100.)
clf.fit(q2_train_data, q2_train_label)
y_pred = clf.predict(q2_test_data)
correct_prediction = np.equal(y_pred, q2_test_label)
accuracy = np.mean(correct_prediction.astype(np.float32))
print(accuracy)

1.0
```

▶ ML  $\frac{B}{B} \rightarrow B$

```
q2_train_data = merge(shuf_data, 2)
q2_train_label = merge(shuf_label, 2)

q2_test_data = shuf_data[1]
q2_test_label = shuf_label[1]
```

▶ ML  $\frac{B}{B} \rightarrow B$

```
import numpy as np
from sklearn import svm

clf = svm.SVC(gamma=0.001, C=100.)
clf.fit(q2_train_data, q2_train_label)
y_pred = clf.predict(q2_test_data)
correct_prediction = np.equal(y_pred, q2_test_label)
accuracy = np.mean(correct_prediction.astype(np.float32))
print(accuracy)
```

0.995

▶ ML  $\frac{B}{B} \rightarrow B$

```
q2_train_data = merge(shuf_data, 3)
q2_train_label = merge(shuf_label, 3)

q2_test_data = shuf_data[2]
q2_test_label = shuf_label[2]
```

▶ ML  $\frac{B}{B} \rightarrow B$

```
import numpy as np
from sklearn import svm

clf = svm.SVC(gamma=0.001, C=100.)
clf.fit(q2_train_data, q2_train_label)
y_pred = clf.predict(q2_test_data)
correct_prediction = np.equal(y_pred, q2_test_label)
accuracy = np.mean(correct_prediction.astype(np.float32))
print(accuracy)
```

1.0



▶ ML  $\frac{B}{B} \rightarrow B$

```
q2_train_data = merge(shuf_data, 4)
q2_train_label = merge(shuf_label, 4)

q2_test_data = shuf_data[3]
q2_test_label = shuf_label[3]
```

▶ ML  $\frac{B}{B} \rightarrow B$

```
import numpy as np
from sklearn import svm

clf = svm.SVC(gamma=0.001, C=100.)
clf.fit(q2_train_data, q2_train_label)
y_pred = clf.predict(q2_test_data)
correct_prediction = np.equal(y_pred, q2_test_label)
accuracy = np.mean(correct_prediction.astype(np.float32))
print(accuracy)
```

0.995

▶ ML  $\frac{B}{B} \rightarrow B$

```
q2_train_data = merge(shuf_data, 5)
q2_train_label = merge(shuf_label, 5)

q2_test_data = shuf_data[4]
q2_test_label = shuf_label[4]
```

▶ ML  $\frac{B}{B} \rightarrow B$

```
import numpy as np
from sklearn import svm

clf = svm.SVC(gamma=0.001, C=100.)
clf.fit(q2_train_data, q2_train_label)
y_pred = clf.predict(q2_test_data)
correct_prediction = np.equal(y_pred, q2_test_label)
accuracy = np.mean(correct_prediction.astype(np.float32))
print(accuracy)
```

0.98

## Chapter 3

### Question 3:

In this question, if we choose '**Patient ID**' as root node, we'd find out the lowest entropy level among all features, since each and every patient has a **unique** 'Patient ID', there will be no uncertainty about reaching any patient because there's one to one relation between a patient and its patient id. Hence, if we choose the root node as 'Patient ID' and construct the tree afterwards, we'd reach out the lowest entropy of the system.

# Chapter 4

## Question 4:

To exert the full power of the Principal Component Analysis with the data set I have, I'd select the imputing approach, since it does not reduce the number of the data points in a data set while also protecting the variance of the troubled feature.

For the complete features, the removal of 200 data points from the data set results in a loss of information about the problem if and only if the removed 200 data points are not exactly identical among themselves, and the data point constructed by mean values of all components, since subtracting mean value from the mean value will only contribute 0 while calculating the variance. Let's assume that for a feature  $i$ , the power of carrying information for a data point  $j$  represents with  $c_{i,j}$ . For a feature  $i$ , the mean and the variance can be calculated as below:

$$\text{Mean}(i) = \frac{1}{n} \sum_{j=1}^n c_{i,j} \text{ and } \text{Variance}(i) = \frac{1}{n-1} \sum_{j=1}^n (c_{i,j} - \frac{1}{n} \sum_{j=1}^n c_{i,j})^2$$

If we remove 200 data points out of 1000, the newfound variance value is based on the remaining 800 points, yet while we try to recover the original data, the missing information that are supplied by the removed 200 data points cannot be represented although we want to represent the 1000 data points. That means the variance value of each feature is now less than the previous case, except the troubled feature, since it has already lost its power to represent a quant of information. This fact results in another fact, which is the newfound set of principal components are different, since the covariance matrix changed due to the changes in variance values of all features.

# Chapter 5

## Question 5:

### 5.1 Part A:

With the given data, we can rewrite the equations as below:

$$(-1) * (w_1 + b) \geq 1$$

$$1 * (3 * w_1 + b) \geq 1$$

Therefore,  $w_1 = 1$ , and  $b = -2$

With  $\min(\frac{1}{2}w^T w) = \min(\frac{1}{2}(w_1^2 + w_2^2))$

Take the derivative with respect to  $w_2 = \frac{1}{2}(2w_2) = 0$

Thus  $w_2 = 0$

### 5.2 Part B:

With the given data, we can rewrite the equations as below:

$$\sum_{i=1}^n y_i \alpha_i = 0$$

$$y_1 \alpha_1 + y_2 \alpha_2 = 0$$

$$(-1) \alpha_1 + 1 \alpha_2 = 0$$

$$\alpha_1 = \alpha_2$$

$$\max(\alpha_1 + \alpha_2 - \frac{1}{2}(y_1 y_1 \alpha_1 \alpha_1 x_1^T x_1 + y_1 y_2 \alpha_1 \alpha_2 x_1^T x_2 + y_2 y_1 \alpha_2 \alpha_1 x_2^T x_1 + y_2 y_2 \alpha_2 \alpha_2 x_2^T x_2))$$

$$\max(2\alpha_1 - \frac{1}{2}((-1) * (-1) * (\alpha_1) * (\alpha_1) * (1) + (-1) * (1) * (\alpha_1) * (\alpha_1) * (3) + (1) * (-1) * (\alpha_1) * (\alpha_1) * (3) + (1) * (1) * (\alpha_1) * (\alpha_1) * (9)))$$

$$\max(2\alpha_1 - \frac{1}{2}(1\alpha_1^2 - 3\alpha_1^2 - 3\alpha_1^2 + 9\alpha_1^2))$$

$$\max(2\alpha_1 - \frac{1}{2}(4\alpha_1^2))$$

$$\max(2\alpha_1 - 2\alpha_1^2)$$

Take the derivative with respect to  $\alpha_1 = 2 - 4\alpha_1 = 0$

$$\alpha_1 = \alpha_2 = \frac{1}{2}$$

### 5.3 Part C: