CMPE 462 Final

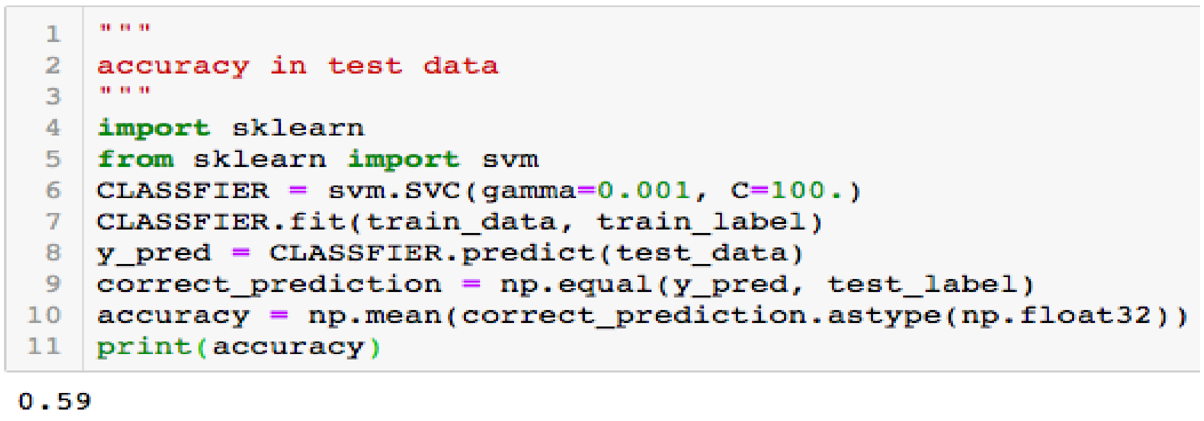
Şadi Uysal
2015400162

July,4,2020

Question 1

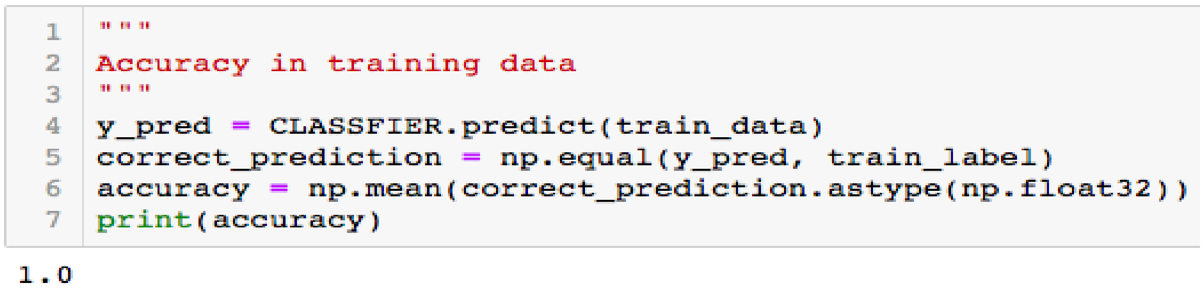
a

Since the classifier is fixed, I thought overfitting can be the reason of poor performance. So I checked immediately test and train data accuracy.



```
"""
accuracy in test data
"""
import sklearn
from sklearn import svm
CLASSFIER = svm.SVC(gamma=0.001, C=100.)
CLASSFIER.fit(train_data, train_label)
y_pred = CLASSFIER.predict(test_data)
correct_prediction = np.equal(y_pred, test_label)
accuracy = np.mean(correct_prediction.astype(np.float32))
print(accuracy)
```

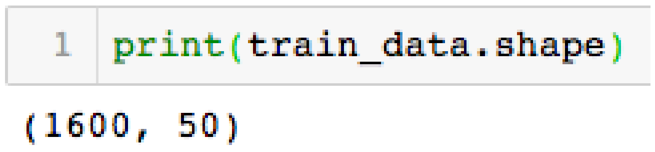
```
0.59
```



```
"""
Accuracy in training data
"""
y_pred = CLASSFIER.predict(train_data)
correct_prediction = np.equal(y_pred, train_label)
accuracy = np.mean(correct_prediction.astype(np.float32))
print(accuracy)
```

```
1.0
```

As I thought, there is huge gap between train and test data accuracy. Then as a next step, I checked data dimensions for analyzing data complexity.



```
print(train_data.shape)
```

```
(1600, 50)
```

I saw that data has 50 feature dimensions which can be the reason for poor accuracy.Using all of 50 feature dimension could be the reason since it would be over complex for this problem.The other possibility could be feature value's range, since values could be not normalized ones.

b

I prefer first possibility and started to work on it.I decided to take feature columns which contributes validation set accuracy.

If any feature has no effect or negative effect, I did not take them to training data.

I started by taking a random feature column, trained classifier with it and tested it on the validation data.

I also assigned a variable named max-iteration-count for maximum column count.

As long as accuracy was improving, I added random feature columns to my train data.

I also used cross-validation to find optimal feature set.

At the end, I selected the necessary columns of test data and predict it via those columns.

I got the following results:

Accuracy: 0.9925

c

[1]:
```
import numpy as np
train_data = np.load("data/train_data1.npy")
train_label = np.load("data/train_label1.npy")
test_data = np.load("data/test_data1.npy")
test_label = np.load("data/test_label1.npy")
```

[2]:
```
"""
accuracy in test data
"""
import sklearn
from sklearn import svm
CLASSFIER = svm.SVC(gamma=0.001, C=100.)
CLASSFIER.fit(train_data, train_label)
y_pred = CLASSFIER.predict(test_data)
correct_prediction = np.equal(y_pred, test_label)
accuracy = np.mean(correct_prediction.astype(np.float32))
print(accuracy)
```

```
0.59
```

[3]:
```
"""
Accuracy in training data
"""
y_pred = CLASSFIER.predict(train_data)
```

```
correct_prediction = np.equal(y_pred, train_label)
accuracy = np.mean(correct_prediction.astype(np.float32))
print(accuracy)
```

```
1.0
```

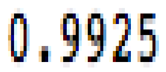
[4]:
```
"""
find best feature indicies to use in classfier by cross_validation
"""
def k_fold_cross_validation(k, X ,Y,numpySeedNumber):
    N = X.shape[0]
    part_N = int(np.floor(N / k))
    values = []  # this list will hold the mean,std dev,r2_score in each validation
    max_accuracy=0
    max_column_indicies=None
    for i in range(k):
        # use i'th part as the validation set
        validation_X = X[i*part_N:(i+1)*part_N,:]
        training_X = np.concatenate((X[:i*part_N,:],X[(i+1)*part_N:,:]), axis=0)
        validation_Y = Y[i*part_N:(i+1)*part_N]
        training_Y = np.concatenate((Y[:i*part_N],Y[(i+1)*part_N:]), axis=0)
        ␣
 ↪accuracy,column_indicies=execute(training_X,validation_X,training_Y,validation_Y,max_itr_count=25,nu
        if accuracy>max_accuracy:
            max_column_indicies=column_indicies
            max_accuracy=accuracy
    return max_column_indicies,max_accuracy
```

[5]:
```
"""
execute given data with execution_repeat_count number of times
return best values for "max_exe_acc,clf_exe,column_indicies"
"""
def execute(X_train,X_test,Y_train,Y_test,max_itr_count,numpySeedNumber):
    np.random.seed(numpySeedNumber)
    training_X=None
    curr_column_indicies=[]
    itr=0
    classifier=None
    max_accuracy=0
    while itr<max_itr_count:
        itr+=1
        clf = CLASSFIER #Same classifier
        ␣
 ↪X_new,new_col_index,curr_column_indicies=rand_col_sampling(X_old=training_X,curr_column_indicies=cur
        clf.fit(X_new, Y_train)
        ␣
 ↪test_data_new=rand_col_sampling(X_old=None,curr_column_indicies=curr_column_indicies+[new_col_index]
        y_pred = clf.predict(test_data_new)
        correct_prediction = np.equal(y_pred, Y_test)
        accuracy = np.mean(correct_prediction.astype(np.float32))
        if accuracy>max_accuracy:
            curr_column_indicies+=[new_col_index]
            max_accuracy=accuracy
            training_X=X_new

    return max_accuracy,curr_column_indicies
```

[6]:
```
"""
takes X_old and add new random column of data into it, if only_selection=True
just select given column indicies from data and return them.
"""

def rand_col_sampling(X_old,curr_column_indicies,data,only_selection=False):

    if only_selection: #if only selection then just return columns at column_indicies
        X_new=None
        for ind in curr_column_indicies:
            if X_new is None: #first column
                X_new = data[:,ind]
            else: #other columns
                X_new = np.concatenate((X_new,data[:,ind]), axis=1)
        return X_new
    else: #if not only_selection then add random column to X_old
        max_col_count=data.shape[1]
        X_new=X_old
        rand=np.random.randint(max_col_count, size=(1))
        while rand in curr_column_indicies:
            rand=np.random.randint(max_col_count, size=(1))
        if X_old is None:
            X_new = data[:,rand]
        else:
            X_new = np.concatenate((X_old,data[:,rand]), axis=1)
        return X_new,rand,curr_column_indicies
```

[ ]:
```
#k_fold to find best classfier trained with features at column_indicies
column_indicies,validation_accuracy=k_fold_cross_validation(k=10, X=train_data␣
 ↪,Y=train_label,numpySeedNumber=0)
```

[ ]:
```
train_data_new=rand_col_sampling(X_old=None,curr_column_indicies=column_indicies,data=train_data,only_s
#use selected columns of test_data to predict labels
test_data_new=rand_col_sampling(X_old=None,curr_column_indicies=column_indicies,data=test_data,only_sel
CLASSFIER.fit(train_data_new, train_label)
#predict test_data
y_pred = CLASSFIER.predict(test_data_new)
#get accuracy
correct_prediction = np.equal(y_pred, test_label)
accuracy = np.mean(correct_prediction.astype(np.float32))
print(accuracy)
```



```
0.9925
```

Question 2

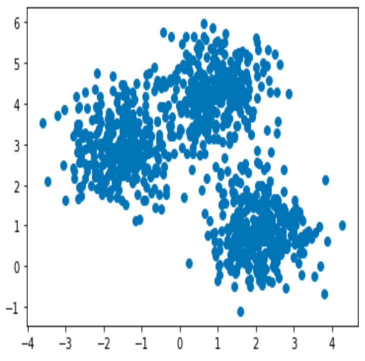
Since we do not have labels, we need to use non-supervised clustering methods to find clusters and label them.

We implemented K-means clustering algorithm before, so I get used to code and decided to use it.

But at the beginning I do not know the number of clusters and I needed to assign cluster count as hyper parameter. So, I decided to try multiple k-values and find best accuracy.

At first, I draw scatter plot to get insight about data. I saw 3 clusters but decided to try multiple k values from 3 to 8.

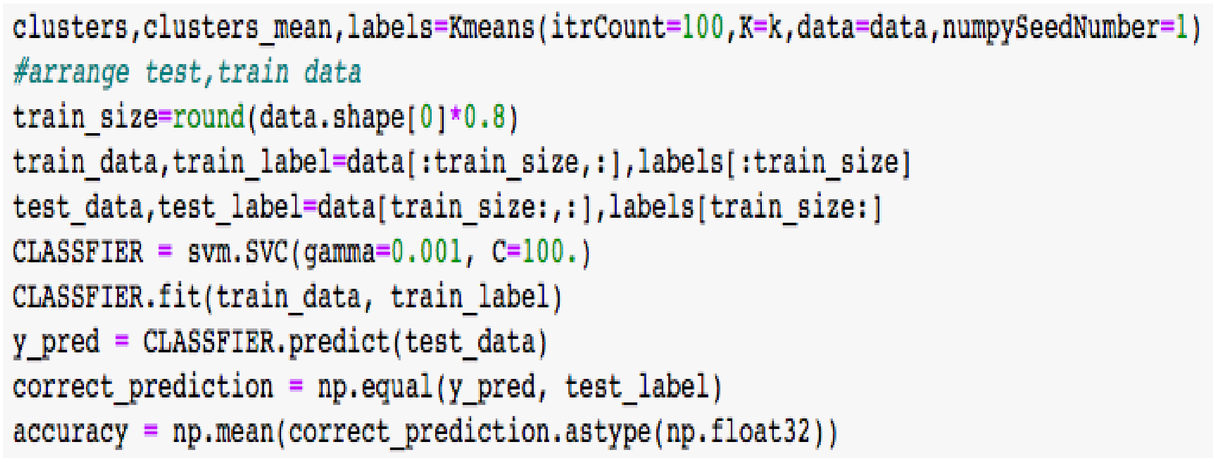
```
[4]: #Draw scatter plot the get insight about data
     plt.scatter(data[:,0], data[:,1])
```



At each K-means execution, I get clusters,clusters-mean and labels.

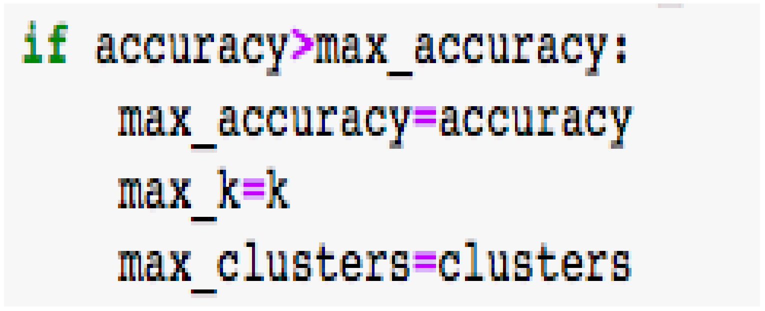
I rearranged train-test data and corresponding labels.

Then I used classifier in Q1 to get accuracy score for that K-means execution.



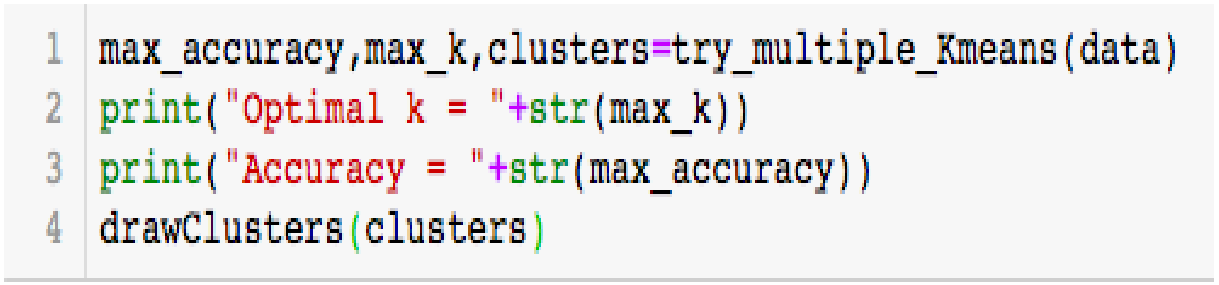
```
clusters,clusters_mean,labels=Kmeans(itrCount=100,K=k,data=data,numpySeedNumber=1)
#arrange test,train data
train_size=round(data.shape[0]*0.8)
train_data,train_label=data[:train_size,:],labels[:train_size]
test_data,test_label=data[train_size:,:],labels[train_size:]
CLASSFIER = svm.SVC(gamma=0.001, C=100.)
CLASSFIER.fit(train_data, train_label)
y_pred = CLASSFIER.predict(test_data)
correct_prediction = np.equal(y_pred, test_label)
accuracy = np.mean(correct_prediction.astype(np.float32))
```

Then I store the values of best configuration:

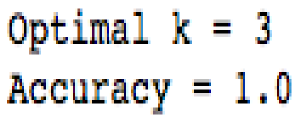


```
if accuracy>max_accuracy:
    max_accuracy=accuracy
    max_k=k
    max_clusters=clusters
```

Lastly returned those best configuration values and cluster labels:

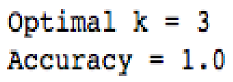


```
max_accuracy,max_k,clusters=try_multiple_Kmeans(data)
print("Optimal k = "+str(max_k))
print("Accuracy = "+str(max_accuracy))
drawClusters(clusters)
```

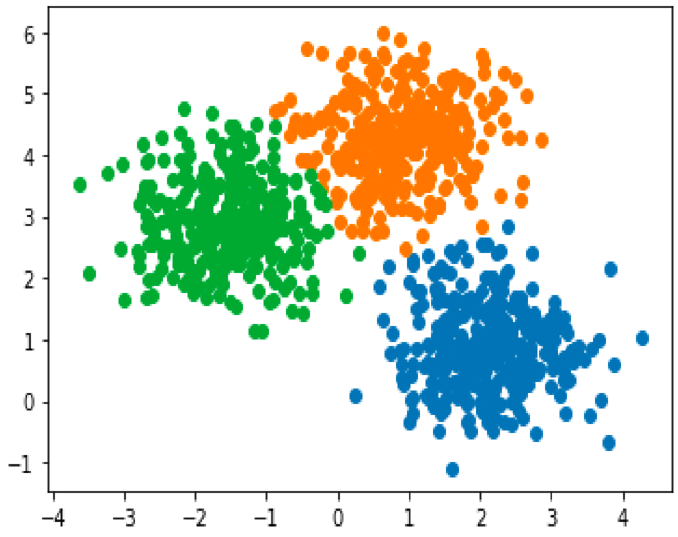


```
Optimal k = 3
Accuracy = 1.0
```

Draw clusters for double check:



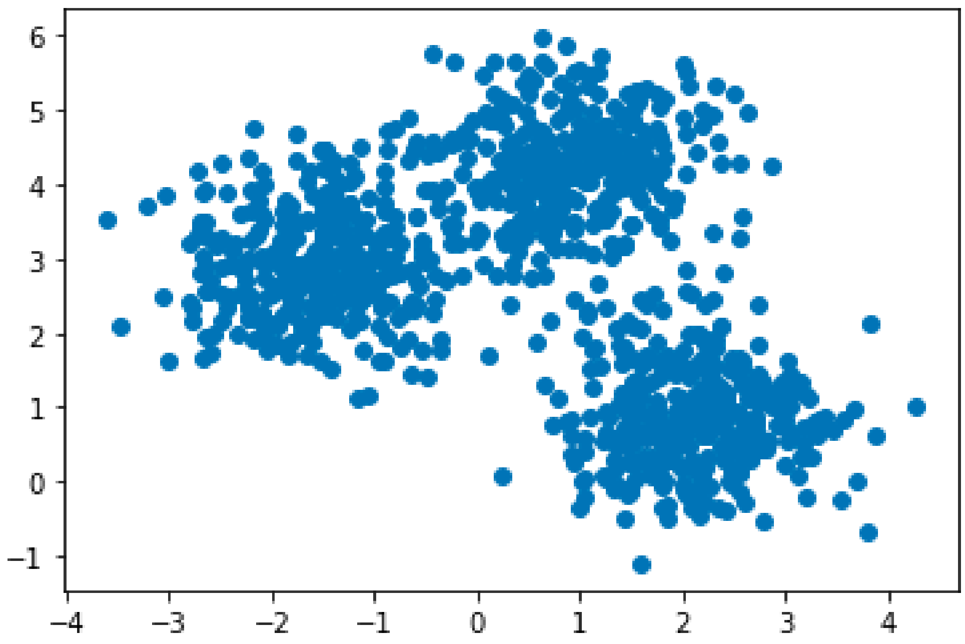
```
Optimal k = 3
Accuracy = 1.0
```



Code PART:

```
[4]: ##########################-------SECOND QUESTION-----------#####################
import matplotlib.pyplot as plt
import numpy as np
import sklearn
from sklearn import svm
data = np.load("data/data2.npy")
#Draw scatter plot the get insight about data
plt.scatter(data[:,0], data[:,1])
```

```
[4]: <matplotlib.collections.PathCollection at 0x1a1598da90>
```

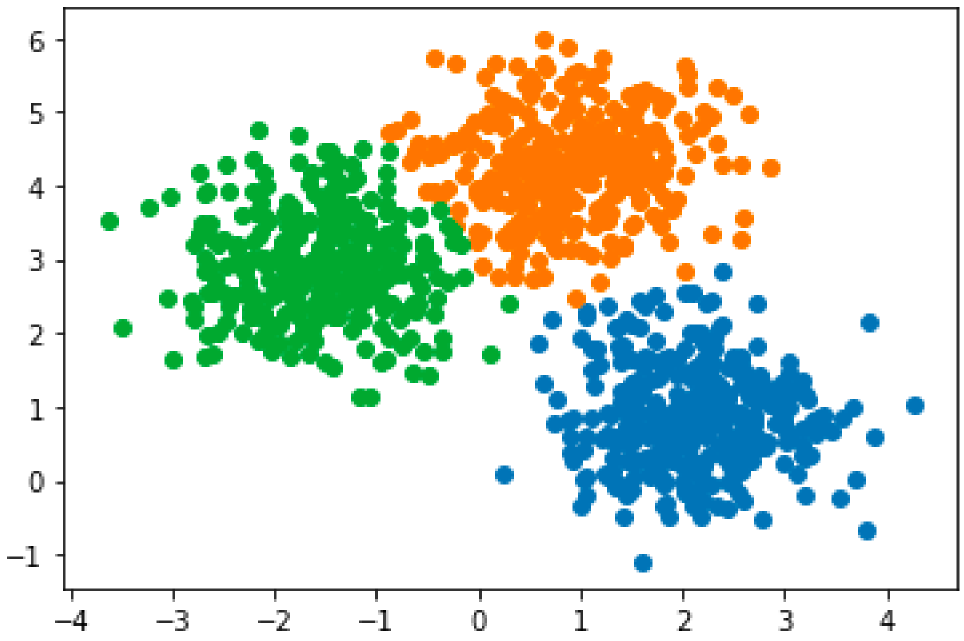


```
[5]: def Kmeans(itrCount,K,data,numpySeedNumber):
    np.random.seed(numpySeedNumber)
    initials=[data[int(rand)] for rand in np.floor(data.shape[0]*np.random.rand(K))]
    labels=-np.ones(data.shape[0])
    clusters_mean=initials
    itr=0
    while itr<itrCount:
        itr+=1
        clusters=[[] for i in range(K)]
        for ind in range(data.shape[0]):
            distances=distanceToClusters(data[ind,:],clusters_mean,K)
            clusterIndex=distances.index(min(distances))
            clusters[clusterIndex].append(data[ind,:])
            labels[ind]=clusterIndex
        clusters_mean=computeCentroids(clusters)
    return clusters,clusters_mean,labels
def distanceToClusters(x,clusters_mean,K):
    distances=[0 for i in range(K)]
    for i in range(K):
        distVec=np.subtract(x, clusters_mean[i])
        distances[i]=distVec.dot(distVec)
    return distances

def computeCentroids(clusters):
    return [np.mean(np.array(cluster), axis=0) for cluster in clusters]

def drawClusters(clusters):
    for cluster in clusters:
        plt.scatter(np.array(cluster)[:,0], np.array(cluster)[:,1])
def try_multiple_Kmeans(data):
    max_accuracy=0
    max_k=None
    max_clusters=None
    for k in range(3,8):
        ␣
 ↪clusters,clusters_mean,labels=Kmeans(itrCount=100,K=k,data=data,numpySeedNumber=1)
        #arrange test,train data
        train_size=round(data.shape[0]*0.8)
        train_data,train_label=data[:train_size,:],labels[:train_size]
        test_data,test_label=data[train_size:,:],labels[train_size:]
        CLASSFIER = svm.SVC(gamma=0.001, C=100.)
        CLASSFIER.fit(train_data, train_label)
        y_pred = CLASSFIER.predict(test_data)
        correct_prediction = np.equal(y_pred, test_label)
        accuracy = np.mean(correct_prediction.astype(np.float32))
        if accuracy>max_accuracy:
            max_accuracy=accuracy
            max_k=k
            max_clusters=clusters
    return max_accuracy,max_k,max_clusters
```

[6]:
```
max_accuracy,max_k,clusters=try_multiple_Kmeans(data)
print("Optimal k = "+str(max_k))
print("Accuracy = "+str(max_accuracy))
drawClusters(clusters)
```

```
Optimal k = 3
Accuracy = 1.0
```

```
Optimal k = 3
Accuracy = 1.0
```



```
[ ]:
```

Question 3

We should choose the attribute with the largest information gain as the split.

$InformationGain = EntropyBefore - EntropyAfter$

Since the most informative feature is glucose information in terms of health vs. diabetes decision, Glucose gives the lowest entropy after splitting the data.

Since patient ID is a unique number for each patient, all the attributes above are not meaningful to learn a generalizable tree.

We can not use patient ID feature to create generalizable tree, since it is different for every patient.

Question 4

Given a sample of n observations on a vector of p variables.

$$\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \in \mathbb{R}^p, \mathbf{x}_j = \left(x_{1j}, x_{2j}, \ldots, x_{pj}\right) \tag{1}$$

p is the attribute count in our case.

$$\begin{aligned} S &= \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i - \overline{\mathbf{x}}\right)\left(\mathbf{x}_i - \overline{\mathbf{x}}\right)^T : \text{ the covariance matrix} \\ \overline{\mathbf{x}} &= \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i : \text{ the mean} \end{aligned} \tag{2}$$

When there are no missing values, the normalized matrix S is just a covariance matrix.But when missing values exists and we fulfilled them, there could be some changes since missing values fulfilled with the assumptions of perfect fit between the model and data.

When we discard the customers with missing values:

As you see above, we are also changing mean of other attribute's mean.

So it completely effects whole matrix S. We lose information gained by 200 customer's attribute's data

In the other case, imputing the missing attribute with the average value of 800 customers would not change the other attribute's means, so principal components would be different in each case.

Question 5

a

$$\begin{array}{cc} \min_{w,b} & \frac{1}{2}w^T w \\ \text{subject to} & y_i\left(w^T x_i + b\right) \geq 1, \forall i \end{array}$$

Training Data:

$$X = \begin{bmatrix} 1 & 0 \\ 3 & 0 \end{bmatrix} \quad y = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad W = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

$$\begin{aligned} &\mathit{Constraints}: \\ &1)(-1)\left(\begin{bmatrix} w_1 w_2 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} + b\right) \geqslant 1 \\ &2)(1)\left(\begin{bmatrix} w_1 w_2 \end{bmatrix}\begin{bmatrix} 3 \\ 0 \end{bmatrix} + b\right) \geqslant 1 \end{aligned}$$

Constraints becomes:

$$-\left(w_1+b\right) \geqslant 1$$
$$(3w_1+b) \geqslant 1$$

$$So, w_1 \geqslant 1$$

$$Optimal Solution:$$

$$w^* = \left[\begin{array}{c} w_1 = 1 \\ w_2 = 0 \end{array} \right], b^* = -2$$

b

$$\max_{\alpha \in \mathbb{R}^N} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m x_n^T x_m$$

$$\text{subject to } \sum_{n=1}^{N} y_n \alpha_n = 0$$

$$\alpha_n \geq 0, n = 1, \cdots, N$$

$$y_1 = -1 \quad y_2 = 1$$
$$y_1\alpha_1 + y_2\alpha_2 = 0$$

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

$$\alpha_2 = \alpha_1 \tag{3}$$

Maximize:

$$2\alpha - \tfrac{1}{2}\left(\quad \alpha^2 x_1^\top x_1 - \alpha^2 x_1^\top x_2 - \alpha^2 x_2^\top x_1 + \alpha^2 x_2^\top x_2\right)$$

$$x_1^\top x_1 = 1 \quad x_1^\top x_2 = 3 \quad x_2^\top x_1 = 3 \quad x_2^\top x_2 = 9$$

It becomes:

$$2\alpha - \frac{1}{2}\left(\alpha^2 - 3\alpha^2 - 3\alpha^2 + 9\alpha^2\right) \tag{4}$$

$$2\alpha - 2\left(\alpha^2\right) \tag{5}$$

Take derivative for maximum:

$$\alpha_n \geq 0 \tag{6}$$

$$2 - 4\left(\alpha\right) = 0 \tag{7}$$

$$\alpha = 0.5 \tag{8}$$

c

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n \tag{9}$$

$$\begin{aligned} w &= \alpha y_1 x_1 + \alpha y_2 x_2 \\ &= \tfrac{1}{2}\left((-1)\begin{bmatrix} 1 \\ 0 \end{bmatrix} + (1)\begin{bmatrix} 3 \\ 0 \end{bmatrix}\right) \\ &= \begin{bmatrix} w_1 = 1 \\ w_2 = 0 \end{bmatrix} \end{aligned} \tag{10}$$

Same result obtained.