Homework1_432001358_CSCE_633_600

September 26, 2021

1 CSCE 633 600 (Machine Learning) Homework 1

1.1 Name: Rohan Chaudhury

1.2 UIN: 432001358

1.2.1 Question 1 (6 points)

Predicting patient post-surgery survival: Predicting a patient's risk of death during a breast cancer surgery can help physicians to make decisions for personalized post-surgery treatment and care. We will use the Haberman's Survival Dataset, collected by the University of Chicago's Billings Hospital, to predict survival for patients who had undergone surgery for breast cancer based on attributes that have been identified as important to our problem (i.e., patient age, year of surgical operation, number of detected positive axillary lymph nodes). We will use available data from the following UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival. Inside "Homework 1" on CANVAS you can find three files including the train and test data (named "data train.csv", "data dev.csv", and "data test.csv") for our experiments. The rows of those files refer to the data samples, while the columns denote the features (columns 1-3) and the class variable (column 4), as described bellow: 1. Age of patient at time of operation (integer) 2. Patient's year of operation (integer, ranging between 58 and 70, which correspond to years 1958-1970) 3. Number of positive axillary lymph nodes detected (integer) 4. class: the patient survived 5 years or longer (1), the patient died within 5 years after the surgery (2)

1.2.2 Answers and codes to assignments are provided below. Explanations for codes are provided as comments inside the code.

1.2.3 (a.i) (0.5 points) Data exploration: Using the training data, compute the number of samples belonging to each class. Are the classes equally distributed?

In [1]:
```
#### Storing the train, test and dev file names in variables
#### Files are stored in the same folder where source code \
#### is present so these variables act as file paths also

#### Also big lines of code are broken into 2 separate lines
#### to fit in the latex pdf format

import matplotlib.pyplot as plt
import pandas as pd
```

```
import collections

file_train="data_train.csv"
file_test="data_test.csv"
file_dev="data_dev.csv"


train_rows =pd.read_csv(file_train, names=["Age"\
                        ,"OperationYear","PositiveAxillarylymphNodes",\
                                         "Class"])

print ("  ")
print("Total no. of rows in data_train.csv: %d \n"%(len(train_rows)))

#### Calculating number of samples belonging to each class

count_of_classes=collections.Counter(train_rows["Class"])


print ("Number of samples belonging to \nclass 1 (the \
patient survived 5 years or longer) : %d"%(count_of_classes[1]))
print ("Number of samples belonging to \nclass 2 (the \
patient died within 5 years after the surgery): %d"%(count_of_classes[2]))
```

```
Total no. of rows in data_train.csv: 245

Number of samples belonging to
class 1 (the patient survived 5 years or longer) : 173
Number of samples belonging to
class 2 (the patient died within 5 years after the surgery): 72
```

1.2.4 Answer (a.i):

As we can see from the above output:

1. Number of samples belonging to class 1: 173
2. Number of samples belonging to class 2: 72

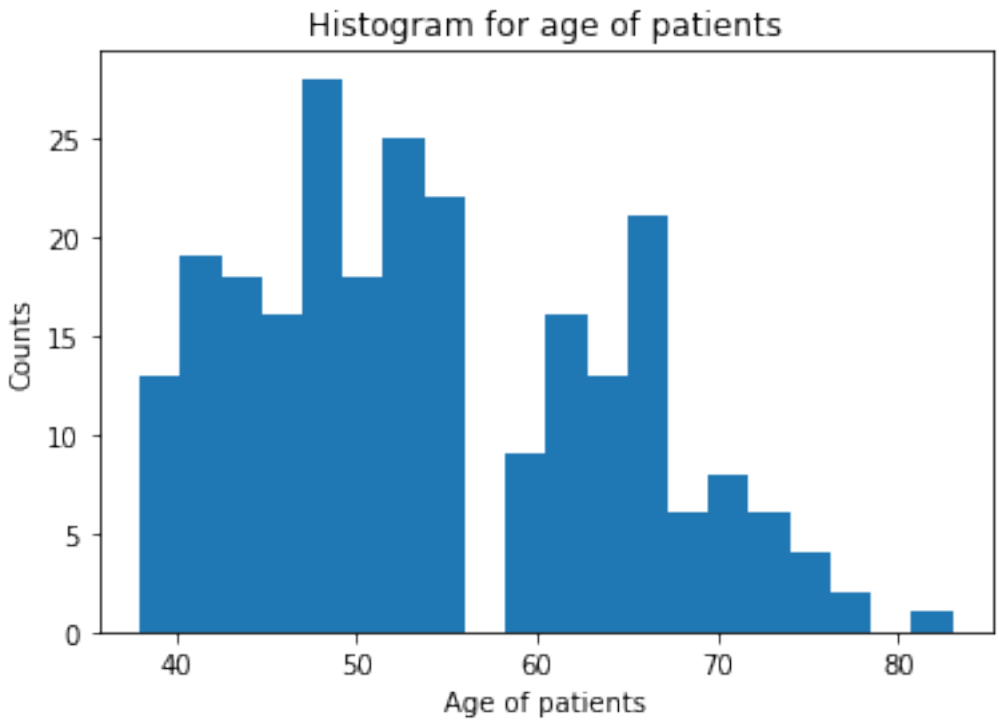
class 1: (the patient survived 5 years or longer)
class 2: (the patient died within 5 years after the surgery)
The classes are not equally distributed

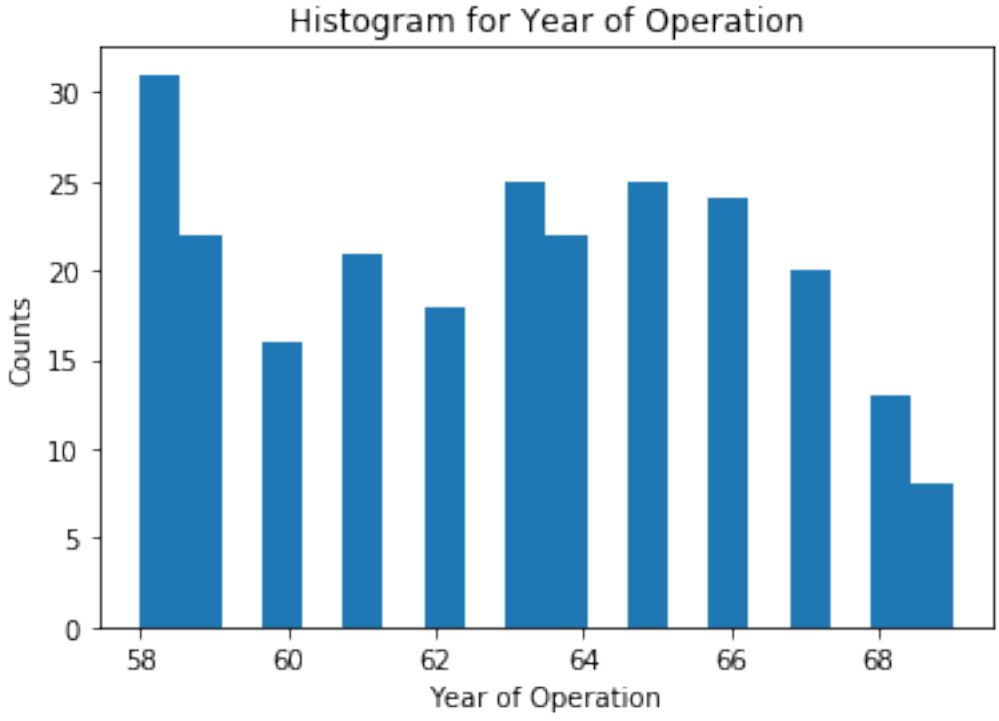
1.2.5 (a.ii) (0.5 points) Data exploration: Using the training data, plot the histogram of each feature (i.e., 3 total histograms). How are the features distributed (e.g., unimodal, bimodal, uniform distributions)?

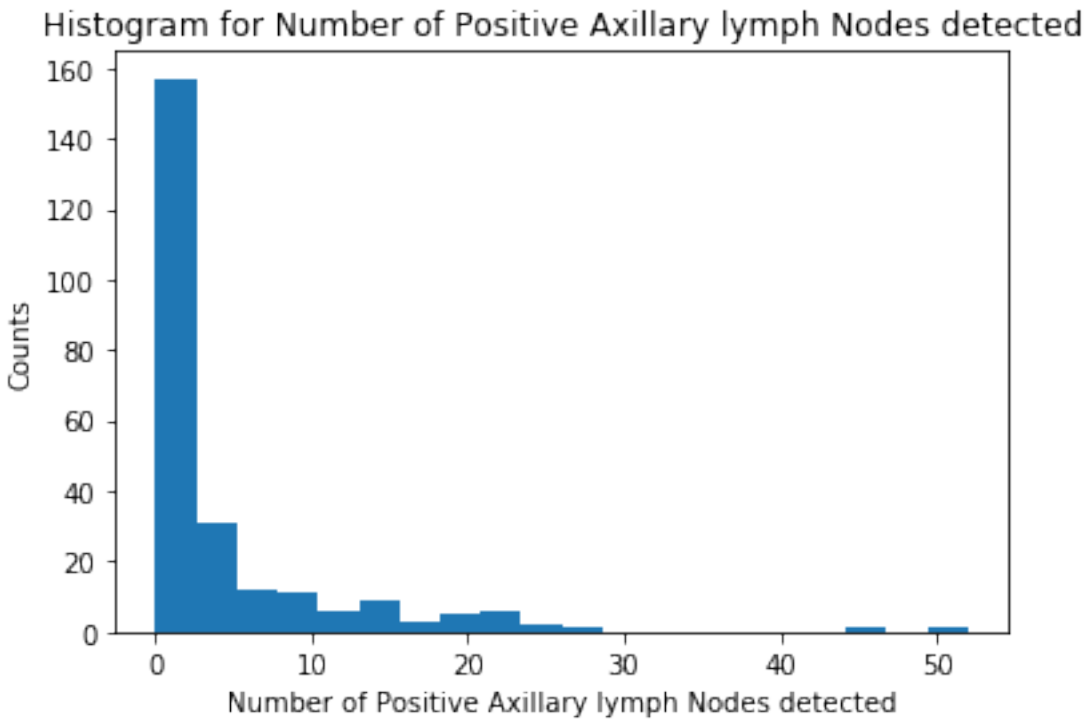
In [2]:
```
plt.figure(1)
plt.hist(train_rows['Age'], bins=20)
plt.xlabel('Age of patients')
plt.ylabel('Counts')
plt.title('Histogram for age of patients')
plt.show()
```

```
plt.figure(2)
plt.hist(train_rows['OperationYear'], bins=20)
plt.xlabel('Year of Operation')
plt.ylabel('Counts')
plt.title('Histogram for Year of Operation')
plt.show()
```

```
plt.figure(3)
plt.hist(train_rows['PositiveAxillarylymphNodes'], bins=20)
plt.xlabel('Number of Positive Axillary lymph Nodes detected')
plt.ylabel('Counts')
plt.title('Histogram for Number of Positive \
Axillary lymph Nodes detected')
plt.show()
```


Histogram for age of patients
Counts
0
5
10
15
20
25
40
50
60
70
80
Age of patients


Histogram for Year of Operation
Counts
0
5
10
15
20
25
30
58
60
62
64
66
68
Year of Operation



1.2.6 Answer (a.ii):

The histograms for the 3 features are plotted above. From the histograms we can see that:

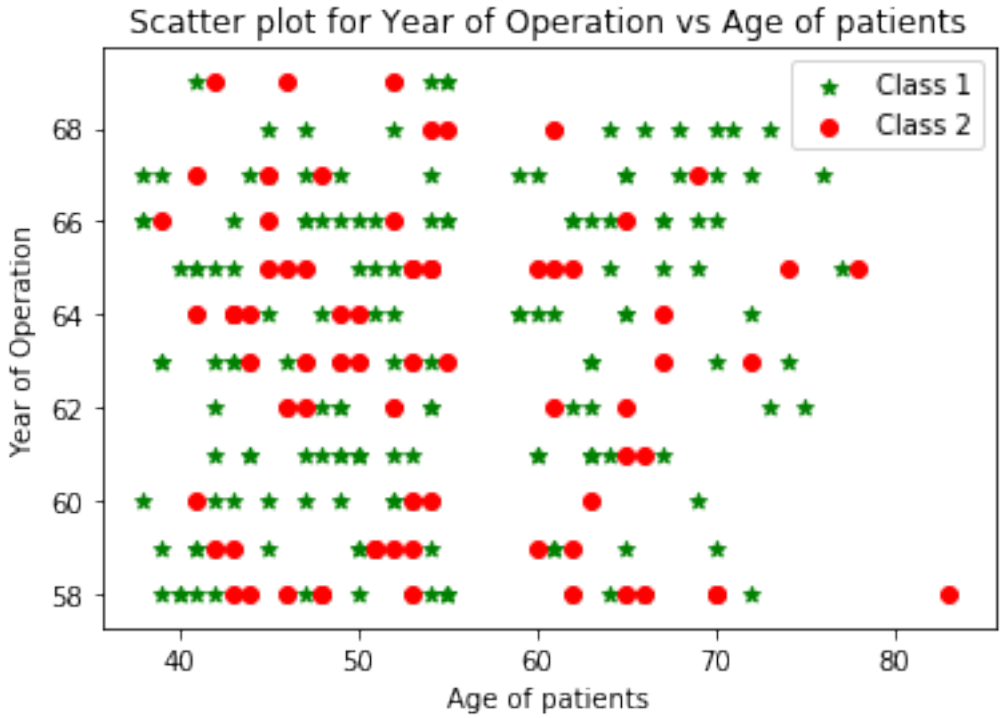
```
1. Histogram for age of patients is Bimodal

2. Histogram for Year of operation is Multimodal

3. Histogram for Number of Positive Axillary lymph Nodes detected is
Right-skewed
```

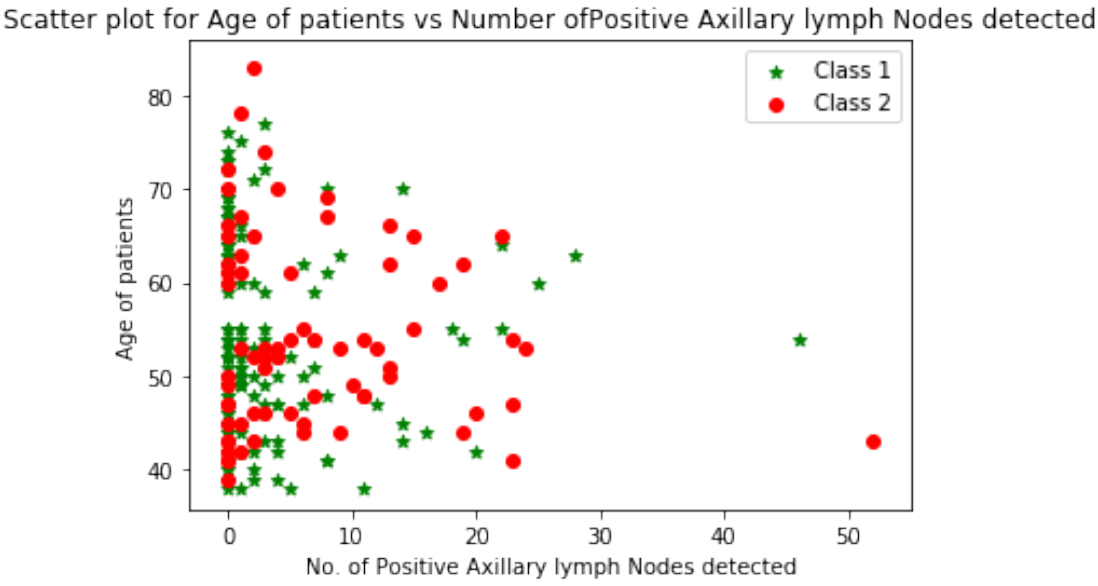
1.2.7 (a.iii) (0.5 points) Data exploration: Using the training data, plot scatter plots of all pairs of features (i.e., 3 total scatter plots). Use a color-coding to indicate the class in which the samples belong to (e.g., blue circle for class 1, green star for class 2). What do you observe? How separable do the classes look? Are there feature combinations for which the two classes are more separable?

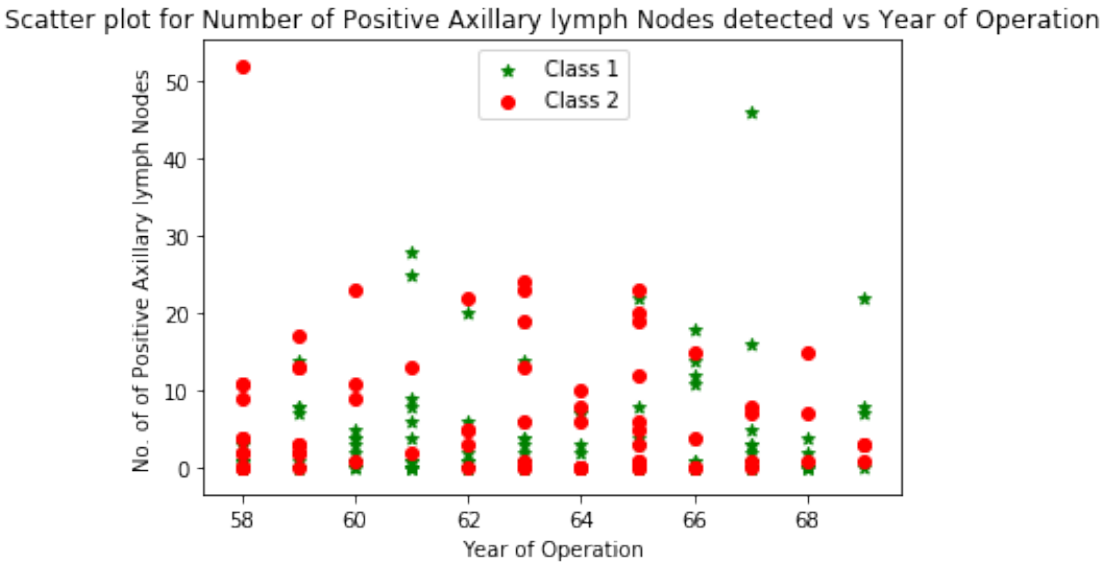
```
In [3]: print ("Green colored star indicates the patient \
        survived 5 years or longer (Class 1) and\n Red colored circle indicates the \
        patient died within 5 years after the  surgery (Class 2)")


        train_rows_class1=train_rows[train_rows["Class"]==1]
        train_rows_class2=train_rows[train_rows["Class"]==2]
```

```
plt.figure(1)
plt.scatter(train_rows_class1['Age'], \
            train_rows_class1['OperationYear'], \
            color='green', marker="*", label="Class 1")
plt.scatter(train_rows_class2['Age'], \
            train_rows_class2['OperationYear'], \
           color='red', label="Class 2")
plt.xlabel('Age of patients')
plt.ylabel('Year of Operation')
plt.title('Scatter plot for Year of \
Operation vs Age of patients')
plt.legend()
plt.show()


plt.figure(2)
plt.scatter(train_rows_class1['PositiveAxillarylymphNodes'],\
            train_rows_class1['Age'],  \
            color='green', marker="*", label="Class 1")

plt.scatter(train_rows_class2['PositiveAxillarylymphNodes'],\
            train_rows_class2['Age'],  \
           color='red', label="Class 2")
plt.xlabel('No. of Positive Axillary lymph Nodes detected')
plt.ylabel('Age of patients')
plt.title('Scatter plot for Age of patients vs Number of\
Positive Axillary lymph Nodes detected')
plt.legend()
plt.show()


plt.figure(3)
plt.scatter(train_rows_class1['OperationYear'],\
            train_rows_class1['PositiveAxillarylymphNodes'],\
            color='green', marker="*", label="Class 1")

plt.scatter(train_rows_class2['OperationYear'],\
            train_rows_class2['PositiveAxillarylymphNodes'],\
            color='red', label="Class 2")
plt.xlabel('Year of Operation')
plt.ylabel('No. of of Positive Axillary lymph Nodes')
plt.title('Scatter plot for Number of Positive \
Axillary lymph Nodes detected vs Year of Operation')
plt.legend()
plt.show()
```

Green colored star indicates the patient survived 5 years or longer (Class 1) and
Red colored circle indicates the patient died within 5 years after the surgery (Class 2)







1.2.8 Answer (a.iii):

The scatter plots for combinations of the 3 features are plotted above. Green colored star indicates the patient survived 5 years or longer (Class 1) and Red colored circle indicates the patient died within 5 years after the surgery (Class 2). From the scatter plots we can see that:

1. The classes in the scatter plot for Year of Operation vs Age of patients are not separable
2. The classes in the scatter plot for Age of patients vs Number of Positive Axillary lymph Nodes detected are not that much separable. But the classes in this scatter plots are more separable than the other scatter plots. From this scatter plot we can somewhat say that people with lesser number of positive axillary lymph nodes detected are more likely to survive.
3. The classes in the scatter plot for Number of Positive Axillary lymph Nodes detected vs Year of Operation are not separable

1.2.9 (b.i) (2 points) Classification: Implement a K-Nearest Neighbor classifier (K-NN) using the euclidean distance (l2-norm) as a distance measure to classify between the three classes. Please implement K-NN and do not use available libraries. In the report, please show your code.

1.2.10 Answer (b.i):

The code is as follows:

```
In [4]: import numpy as np
        import math
        ####Function to calculate the euclidean distance

        def euclidean_distance(x1,x2):
```

```
    l2norm=0
    for i in range(len(x1)):
        l2norm=l2norm+(x1[i]-x2[i])**2

    l2norm=np.sqrt(l2norm)
    return l2norm

####Function to calculate the Manhattan distance
def Manhattan_distance(x1,x2):
    l1norm=0
    for i in range(len(x1)):
        l1norm=l1norm+abs(x1[i]-x2[i])

    return l1norm

def get_element_in_first_position(x):
    return x[0]


def knn_for_one(K,data_train,\
                test_point_features, \
                features, Manhattan=False):

    #### Converting to dict to avoid using .loc in \
    #### pandas as that is very slow

    data_train_dict=data_train.to_dict()

    distances=[]
    train_classes=[]
    for i in range(len(data_train)):
        train_features=[]

        for j in features:
            train_features.append(data_train_dict[j][i])
        train_classes.append(data_train_dict["Class"][i])
        if (Manhattan==True):
            distance=Manhattan_distance(train_features,\
                                        test_point_features)
        else:
            distance=euclidean_distance(train_features,\
                                        test_point_features)
        distances.append(distance)



    sorted_distance_classes=[Class for _,Class in \
                             sorted(zip(distances,\
```

```
                                        train_classes), \
                                        key=get_element_in_first_position)]

    first_k=sorted_distance_classes[:K]

    class1=0
    class2=0

    for samples in first_k:
        if samples==1:
            class1=class1+1
        else:
            class2=class2+1

    if class1>class2:
        return 1
    elif class1<class2:
        return 2


def run_knn_on_all(K,train_rows, \
                   test_rows, features,\
                   Manhattan=False):
    error=0
    correct=0
    class_1_correct=0
    class_2_correct=0
    class_1_samples=0
    class_2_samples=0
    test_rows_dict=test_rows.to_dict()
    for i in range(len(test_rows)):
        test_row_features=[]
        for j in features:
            test_row_features.append(test_rows_dict[j][i])
        prediction=knn_for_one(K,train_rows,\
                               test_row_features,\
                               features, Manhattan)
        if (prediction!=test_rows_dict["Class"][i]):
            error=error+1
        else:
            correct+=1
        if (prediction==1 and test_rows_dict["Class"][i]==1):
            class_1_correct+=1
        if (prediction==2 and test_rows_dict["Class"][i]==2):
            class_2_correct+=1
        if (test_rows_dict["Class"][i]==1):
            class_1_samples+=1
        if (test_rows_dict["Class"][i]==2):
```

```
                class_2_samples+=1

        error_rate=error/len(test_rows)
        accuracy=correct/len(test_rows)
        balanced_accuracy=0.5*(class_1_correct/class_1_samples)\
                            +0.5*(class_2_correct/class_2_samples)
        return error_rate,accuracy,balanced_accuracy
```

In [5]:
```
#### Reading the test data

test_rows =pd.read_csv(file_test, \
                       names=["Age","OperationYear",\
                              "PositiveAxillarylymphNodes", \
                              "Class"])

#### Running KNN for K=1 on test data
error_rate, accuracy, balanced_accuracy = run_knn_on_all(\
                                                    1,train_rows, \
                                                      test_rows, \
                                                     ["Age","OperationYear"\
                                          ,"PositiveAxillarylymphNodes"])


print ("KNN classification accuracy for K=1 \
on test dataset: {}".format(accuracy))
print ("KNN balanced classification accuracy \
for K=1 on test dataset: {}".format( balanced_accuracy))
```

```
KNN classification accuracy for K=1 on test dataset: 0.9032258064516129
KNN balanced classification accuracy for K=1 on test dataset: 0.8133333333333332
```

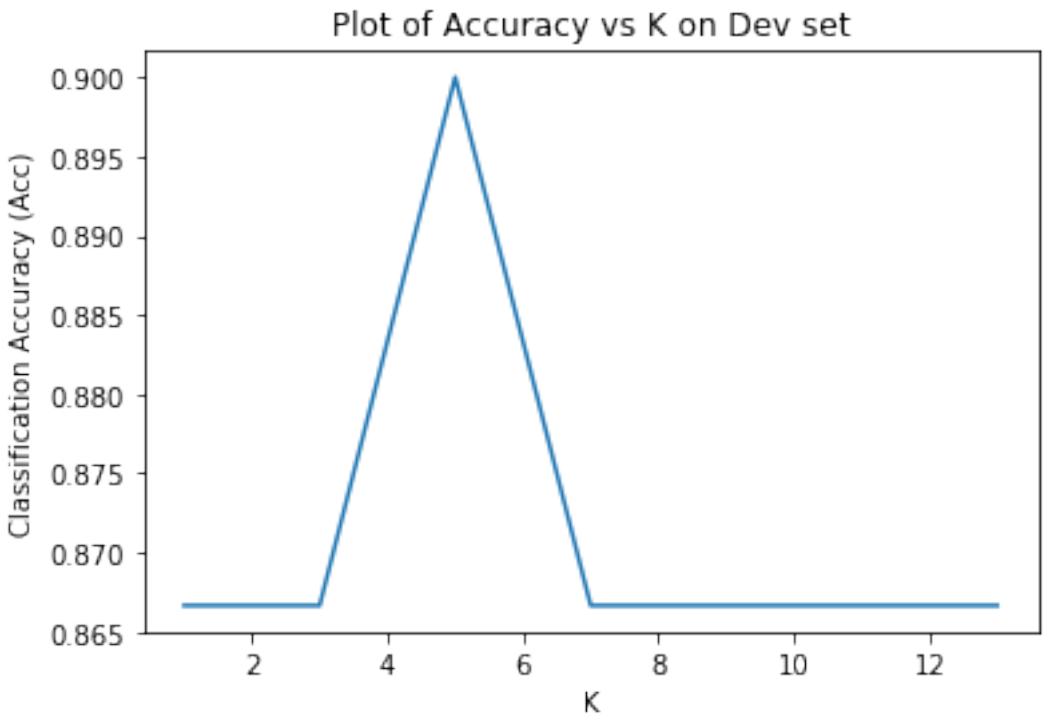
1.2.11 (b.ii) (1 point) Explore different values of K = 1; 3; 5; 7; 9; 11; 13. You will train one model for each of the seven values of K using the train data and compute the classification accuracy (Acc) and balanced classification accuract (BAcc) of the model on the development set. Plot the Acc and BAcc metrics on the dev set against the different values of K. Please report the best hyper-parameter K* based on the Acc metric and the best hyper-parameter K** based on the BAcc metric. Please implement this procedure, including computing the accuracy metrics, from scratch and do not use available libraries.

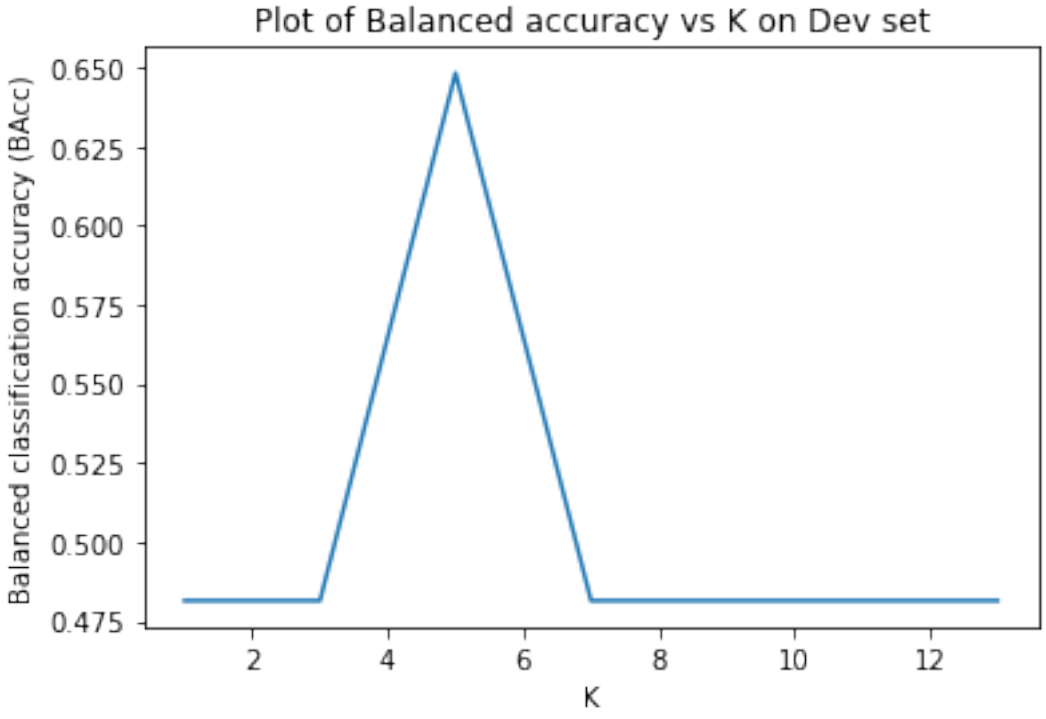
In [6]:
```
#### Reading the development data

dev_rows =pd.read_csv(file_dev,\
                      names=["Age","OperationYear",\
                             "PositiveAxillarylymphNodes", \
                             "Class"])
```

```
In [7]: K_set=[1,3,5,7,9,11,13]


        def find_best_K(train_rows,dev_rows,K_set,\
                        features, Manhattan=False):
            K_star=[]
            accuracies=[]
            K_starstar=[]
            balanced_accuracies=[]
            for K in K_set:
                error_rate, accuracy, balanced_accuracy = run_knn_on_all(\
                                                          K,train_rows, \
                                                             dev_rows, \
                                                           features, \
                                                             Manhattan)
                K_star.append((accuracy,K))
                K_starstar.append((balanced_accuracy,K))
                accuracies.append(accuracy)
                balanced_accuracies.append(balanced_accuracy)
            best_K_star=sorted(K_star)[-1]
            best_K_starstar=sorted(K_starstar)[-1]


            return accuracies, balanced_accuracies, best_K_star, best_K_starstar

In [8]: accuracies, balanced_accuracies, best_K_star, best_K_starstar= \
            find_best_K(train_rows,dev_rows,K_set, ["Age","OperationYear"\
                                            ,"PositiveAxillarylymphNodes"])

In [9]: def plot_k_vs_accuracy(K_set, accuracies, balanced_accuracies):
            plt.figure(1)
            plt.plot(K_set,accuracies)
            plt.title('Plot of Accuracy vs K on Dev set')
            plt.xlabel('K')
            plt.ylabel('Classification Accuracy (Acc)')
            plt.show()


            plt.figure(2)
            plt.plot(K_set,balanced_accuracies)
            plt.title('Plot of Balanced accuracy vs K on Dev set')
            plt.xlabel('K')
            plt.ylabel('Balanced classification accuracy (BAcc)')
            plt.show()

        def plot_k_vs_accuracy_test(K_set, accuracies, balanced_accuracies):
            plt.figure(1)
            plt.plot(K_set,accuracies)
```

```
        plt.title('Plot of Accuracy vs K on test set')
        plt.xlabel('K')
        plt.ylabel('Classification Accuracy (Acc)')
        plt.show()


        plt.figure(2)
        plt.plot(K_set,balanced_accuracies)
        plt.title('Plot of Balanced accuracy vs K on test set')
        plt.xlabel('K')
        plt.ylabel('Balanced classification accuracy (BAcc)')
        plt.show()


    for i in range(len(K_set)):
        print ("Acc for K={} on development dataset using Euclidean \
    distance is {}".format(K_set[i],accuracies[i]))
        print ("BAcc for K={} on development dataset using Euclidean \
    distance is {}".format(K_set[i],balanced_accuracies[i]))


    plot_k_vs_accuracy(K_set, accuracies, balanced_accuracies)
    print ("The best hyper-parameter K* based on the \
    Acc metric is: %d"%(best_K_star[1]))
    print ("The best hyper-parameter K** based on the\
    BAcc metric is: %d"%(best_K_starstar[1]))
    print ("The best classification accuracy obtained: {}".format(accuracies[K_set.index(be
    print ("The best balanced classification accuracy obtained: {}".format(balanced_accurac
```

```
Acc for K=1 on development dataset using Euclidean distance is 0.8666666666666667
BAcc for K=1 on development dataset using Euclidean distance is 0.48148148148148145
Acc for K=3 on development dataset using Euclidean distance is 0.8666666666666667
BAcc for K=3 on development dataset using Euclidean distance is 0.48148148148148145
Acc for K=5 on development dataset using Euclidean distance is 0.9
BAcc for K=5 on development dataset using Euclidean distance is 0.6481481481481481
Acc for K=7 on development dataset using Euclidean distance is 0.8666666666666667
BAcc for K=7 on development dataset using Euclidean distance is 0.48148148148148145
Acc for K=9 on development dataset using Euclidean distance is 0.8666666666666667
BAcc for K=9 on development dataset using Euclidean distance is 0.48148148148148145
Acc for K=11 on development dataset using Euclidean distance is 0.8666666666666667
BAcc for K=11 on development dataset using Euclidean distance is 0.48148148148148145
Acc for K=13 on development dataset using Euclidean distance is 0.8666666666666667
BAcc for K=13 on development dataset using Euclidean distance is 0.48148148148148145
```


Plot of Accuracy vs K on Dev set
Classification Accuracy (Acc)
0.900
0.895
0.890
0.885
0.880
0.875
0.870
0.865
2
4
6
8
10
12
K


Plot of Balanced accuracy vs K on Dev set
Balanced classification accuracy (BAcc)
0.650
0.625
0.600
0.575
0.550
0.525
0.500
0.475
2
4
6
8
10
12
K

```
The best hyper-parameter K* based on the Acc metric is: 5
The best hyper-parameter K** based on theBAcc metric is: 5
The best classification accuracy obtained: 0.9
The best balanced classification accuracy obtained: 0.6481481481481481
```

1.2.12 Answer (b.ii):

The plots for Acc and BAcc metrics on the dev set against the different values of K are shown above. Also:

1. The best hyper-parameter K* based on the Acc metric is: 5
2. The best hyper-parameter K** based on the BAcc metric is: 5
3. The best classification accuracy obtained: 0.9
4. The best balanced classification accuracy obtained: 0.6481481481481481

Both are same: 5

1.2.13 (b.iii) (0.5 points) Report the Acc and BAcc metrics on the test set using K* and K**.

```
In [10]: error_rate,accuracy,balanced_accuracy=\
                run_knn_on_all(best_K_star[1],train_rows,\
                               test_rows, ["Age","OperationYear",\
                                           "PositiveAxillarylymphNodes"])
         print ("Acc on test set with K* and K** \
         (both K* and K** is 5): {}".format(accuracy))
         print ("BAcc on test set with K* and K** (both K* and K** \
         is 5): {}".format(balanced_accuracy))
```

```
Acc on test set with K* and K** (both K* and K** is 5): 0.8387096774193549
BAcc on test set with K* and K** (both K* and K** is 5): 0.6466666666666666
```

1.2.14 Answer (b.iii):

The value of Classification Accuracy (Acc) and Balanced classification accuracy (BAcc) on test using K* and K** is:

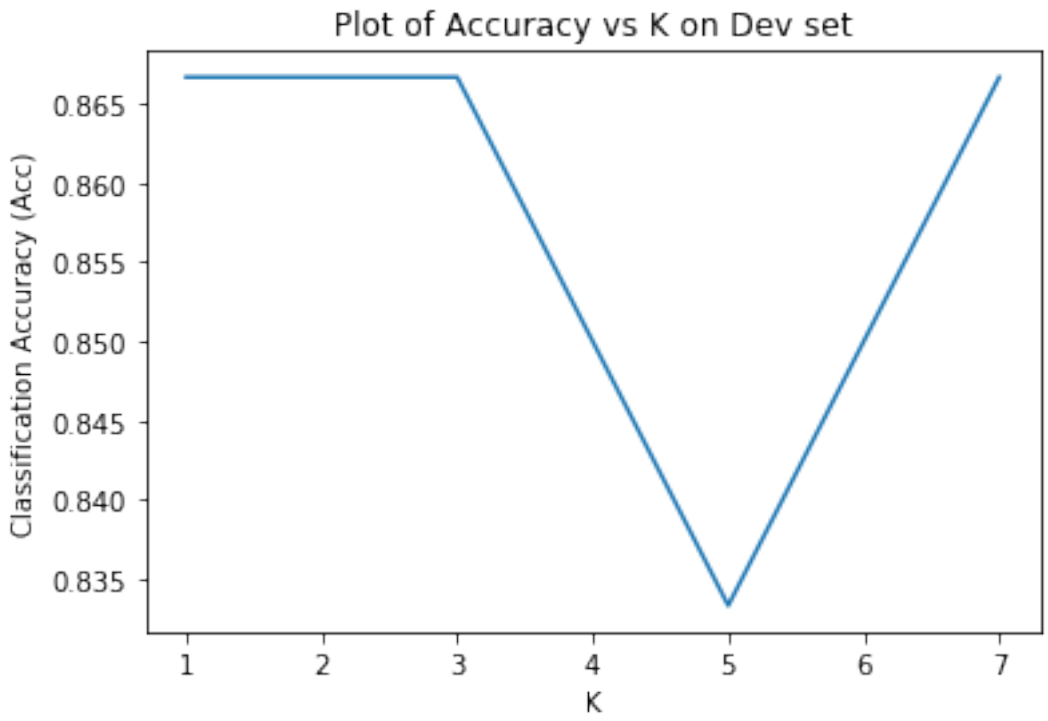
1. Acc on test set with K* and K** (both K* and K** is 5): 0.8387096774193549
2. BAcc on test set with K* and K** (both K* and K** is 5): 0.6466666666666666

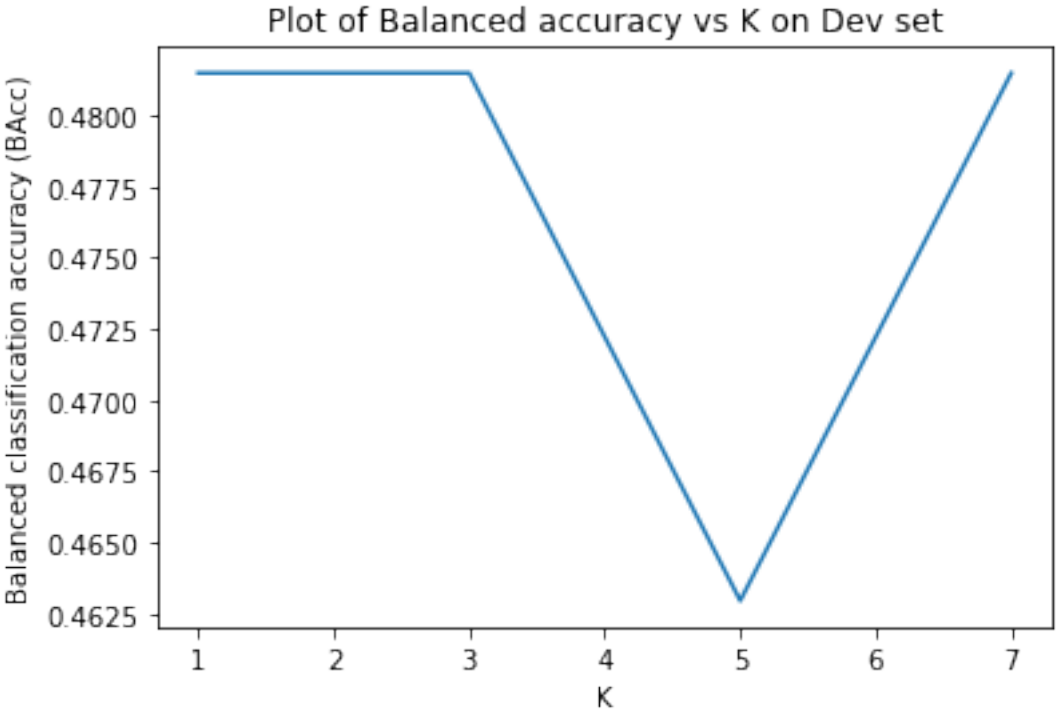
1.2.15 (b.iv) (0.5 points) Instead of using the euclidean distance, experiment with the l1-norm (i.e., Manhattan distance) for K = 1; 3; 5; 7. Report your findings.

```
In [11]: K_manhattan=[1,3,5,7]

         accuracies, balanced_accuracies, best_K_star, best_K_starstar=\
```

```
find_best_K(train_rows,dev_rows,K_manhattan, \
            ["Age","OperationYear","PositiveAxillarylymphNodes"],\
            Manhattan=True)
```

In [12]:
```
plot_k_vs_accuracy(K_manhattan, accuracies, balanced_accuracies)
for i in range(len(K_manhattan)):
    print ("Acc for K={} on development dataset using Manhattan \
distance is {}".format(K_manhattan[i],accuracies[i]))
    print ("BAcc for K={} on development dataset using Manhattan \
distance is {}".format(K_manhattan[i],balanced_accuracies[i]))
print ("  ")
print ("So the best values for K are [1,3,7] which gives highest \
classification accuracy\nand balanced classification accuracy using \
Manhattan distance on Development dataset")
print (" ")
error_rate,accuracy,balanced_accuracy=\
        run_knn_on_all(1,train_rows, test_rows, \
                       ["Age","OperationYear",\
                        "PositiveAxillarylymphNodes"],\
                       Manhattan=True)
print (" ")
print ("Using K=1 to get the Acc and BAcc on the testing set:")
print ("Acc for K=1 on test set using Manhattan distance \
is {}".format(accuracy))
print ("BAcc for K=1 on test set using Manhattan distance \
is {}".format(balanced_accuracy))


print ("   ")


print ("If we plot K vs Acc an BAcc on the test set we would get the following:")


accuracies, balanced_accuracies, best_K_star, best_K_starstar=\
find_best_K(train_rows,test_rows,K_manhattan, \
            ["Age","OperationYear","PositiveAxillarylymphNodes"],\
            Manhattan=True)

plot_k_vs_accuracy_test(K_manhattan, accuracies, balanced_accuracies)


for i in range(len(K_manhattan)):
    print ("Acc for K={} on test dataset using Manhattan \
distance is {}".format(K_manhattan[i],accuracies[i]))
    print ("BAcc for K={} on test dataset using Manhattan \
distance is {}".format(K_manhattan[i],balanced_accuracies[i]))
```


Plot of Accuracy vs K on Dev set
Classification Accuracy (Acc)
0.865
0.860
0.855
0.850
0.845
0.840
0.835
1
2
3
4
5
6
7
K


Plot of Balanced accuracy vs K on Dev set
Balanced classification accuracy (BAcc)
0.4800
0.4775
0.4750
0.4725
0.4700
0.4675
0.4650
0.4625
1
2
3
4
5
6
7
K

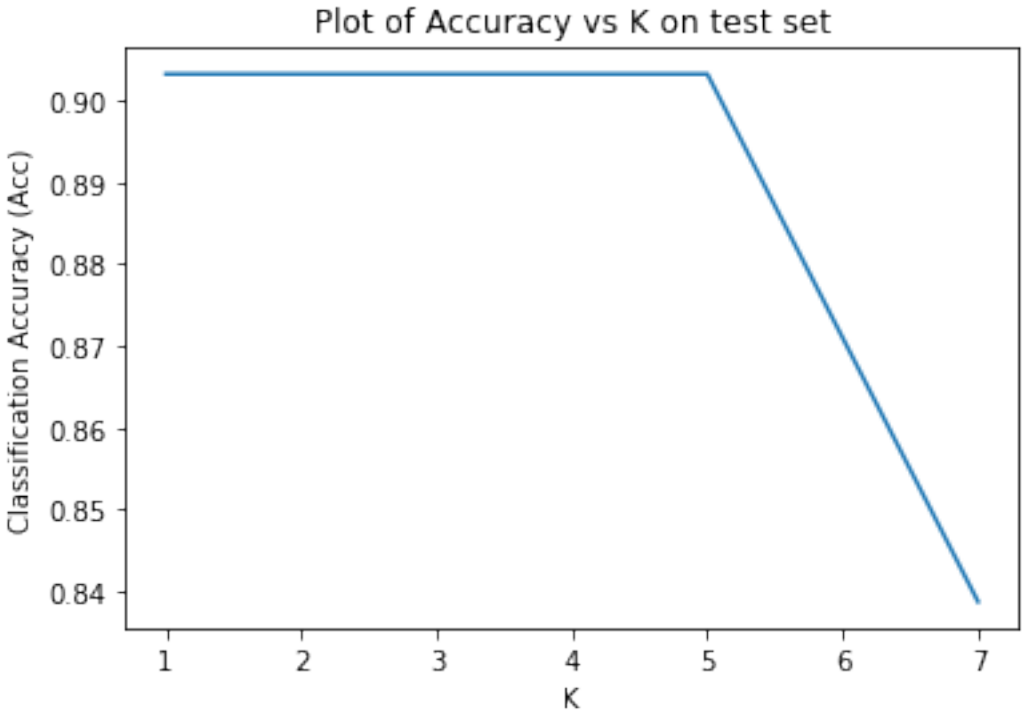
```
Acc for K=1 on development dataset using Manhattan distance is 0.8666666666666667
BAcc for K=1 on development dataset using Manhattan distance is 0.48148148148148145
Acc for K=3 on development dataset using Manhattan distance is 0.8666666666666667
BAcc for K=3 on development dataset using Manhattan distance is 0.48148148148148145
Acc for K=5 on development dataset using Manhattan distance is 0.8333333333333334
BAcc for K=5 on development dataset using Manhattan distance is 0.46296296296296297
Acc for K=7 on development dataset using Manhattan distance is 0.8666666666666667
BAcc for K=7 on development dataset using Manhattan distance is 0.48148148148148145
```

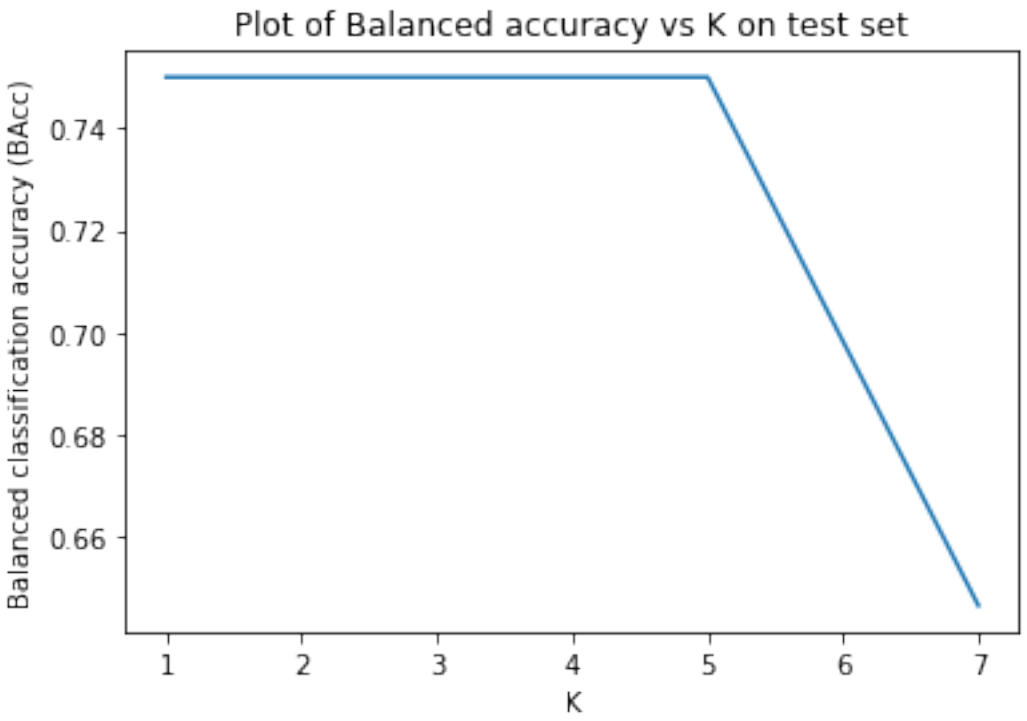
So the best values for K are [1,3,7] which gives highest classification accuracy and balanced classification accuracy using Manhattan distance on Development dataset

Using K=1 to get the Acc and BAcc on the testing set:

```
Acc for K=1 on test set using Manhattan distance is 0.9032258064516129
BAcc for K=1 on test set using Manhattan distance is 0.75
```

If we plot K vs Acc an BAcc on the test set we would get the following:






```
Acc for K=1 on test dataset using Manhattan distance is 0.9032258064516129
BAcc for K=1 on test dataset using Manhattan distance is 0.75
Acc for K=3 on test dataset using Manhattan distance is 0.9032258064516129
BAcc for K=3 on test dataset using Manhattan distance is 0.75
Acc for K=5 on test dataset using Manhattan distance is 0.9032258064516129
BAcc for K=5 on test dataset using Manhattan distance is 0.75
Acc for K=7 on test dataset using Manhattan distance is 0.8387096774193549
BAcc for K=7 on test dataset using Manhattan distance is 0.6466666666666666
```

1.2.16 Answer (b.iv):

Using Manhattan Distance we can see that:

1. Classification Accuracy (Acc) for K=[1,3,7] on the development dataset is maximum and the value is Acc = 0.8666666666666667
2. Balanced classification accuracy (BAcc) for K=[1,3,7] on the development data is maximum and the value is BAcc = 0.48148148148148145

Using K=1 (one value of K which gives the maximum Acc and BAcc on development dataset) for Manhattan Distance on the test dataset we can see that:

1. Classification Accuracy for K=1 on test set using Manhattan distance is 0.9032258064516129

2. Balanced classification accuracy for K=1 on test set using Manhattan distance is 0.75

From the above graphs we can see that the same maximum testing classification accuracy (0.9032258064516129) and balanced classification accuracy (0.75) are obtained on test dataset for K=[1,3,5]

1.2.17 (c) (0.5 points) ML deployment: Assume that the Memorial Herman Hospital in Houston, TX is planning to deploy this system over the next months in order to predict patient post- surgery mortality rate. What would be your thoughts / questions / concerns regarding this?

Answer (c): The testing classification accuracy for the model using Euclidean distance is 0.8387 (83.87%) and 0.9032 (90.32%) using Manhattan distance.

The testing balanced classification accuracy for the model using Euclidean distance is 0.6466 (64.66%) and 0.75 (75%) using Manhattan distance.

These accuracy numbers are not that high considering the fact that this model needs to be deployed for a critical healthcare system, therefore it is best not to take the model's prediction too seriously. Also we have very less features (only 3) to predict patient post- surgery mortality rate out of which Year of Operation is practically a useless feature. The most useful feature is the "Number of Positive Axillary lymph Nodes detected" feature which shows that greater the number of Positive Axillary lymph Nodes, lesser is the chance of survival. Similar to this, if we can find a few more useful features like whether the patient had other diseases etc. then we can improve the accuracy of the model. Also the training data is very less (only 245) in number and the classes are also not equally distributed (173 data points in Class 1 and 72 data points in Class 2). If we could find some more data points and balance the number of data points in each class then it might be possible to increase the accuracy of the model.

1.2.18 Question 2 (4 points)

Linear Perceptron Algorithm: The goal of this problem is to run a linear perceptron algo- rithm on paper and pencil. Assume that you have three training samples in the 2D space:

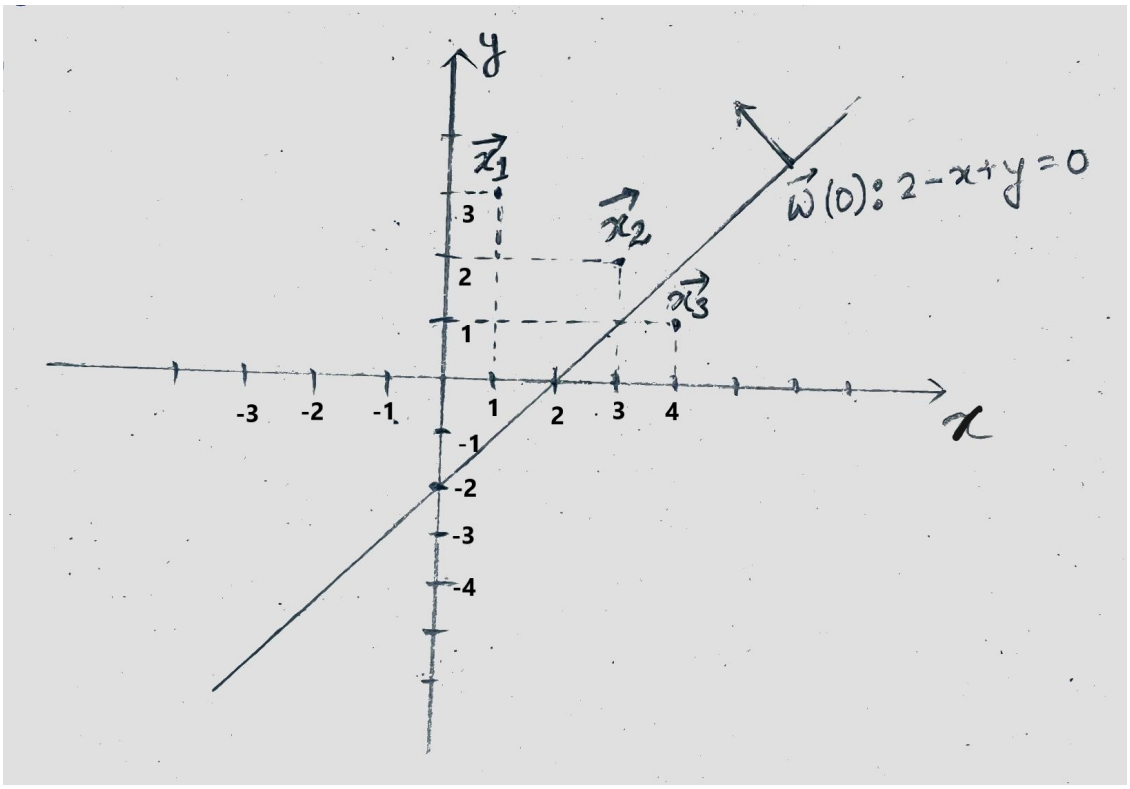
1. Sample x1 with coordinates (1; 3) belonging to Class 1 (y1 = 1)
2. Sample x2 with coordinates (3; 2) belonging to Class 2 (y2 = -1)
3. Sample x3 with coordinates (4; 1) belonging to Class 2 (y2 = -1)

The linear perceptron is initialized with a line with corresponding weight $w(0) = [2; -1; 1]^T$, or else the line $2 - x + y = 0$.

In contrast to the example that we have done in class, in this problem we will include the intercept term w0.

1.2.19 (0.5 points) (i) Plot x1, x2, and x3 in the given 2D space. Plot the line corresponding to weight w(0), as well as the direction of the weight w(0) on the line.

```
In [13]: from IPython.display import Image, display

         display(Image(filename="Plot_1.jpg", width = 600, height = 300))
```



1.2.20 Answer (i)

The points $x_1(1,3), x_2(3,2), x_3(4,1)$ are plotted in the above 2D space. The line corresponding to weight $w(0)$ is plotted along with the direction of weight $w(0)$ on the line.

1.2.21 (1 point) (ii) Using the rule $sign(w(t)^T x_n)$, please indicate the class in which samples x1, x2, and x3 are classified using the weight w(0). Which samples are not correctly classified based on this rule?

1.2.22 Answer (ii):

For x_1:

$w(0)^T x_1 = 2 - 1 + 3 = 4 > 0$

$sign(w(0)^T x_1) = sign(4) = 1$,

therefore the sample x_1 is classified as belonging to class 1 using weight w(0) and it is correctly classified

For x_2:

$w(0)^T x_2 = 2 - 3 + 2 = 1 > 0$

$sign(w(0)^T x_2) = sign(1) = 1$,

therefore the sample x_2 is classified as belonging to class 1 using weight w(0) and it is incorrectly classified

For x_3:

$w(0)^T x_3 = 2 - 4 + 1 = -1 < 0$

$sign(w(0)^T x_3) = sign(-1) = -1$,

therefore the sample x_3 is classified as belonging to class -1 using weight w(0) and it is correctly classified

Hence, sample x_2 $(3,2)$ is not correctly classified based on the rule using weight w(0) and samples x_1 and x_3 are correctly classified

1.2.23 (iii) Using the weight update rule from the linear perceptron algorithm, please find the value of the new weight w(1) based on the misclassified sample from question (ii). Find and plot the new line corresponding to weight w(1) in the 2D space, as well as the direction of the weight w(1) on the line. Indicate which samples are correctly classified and which samples are not correctly classified.

1.2.24 Answer (iii):

Weight update rule: $w(t+1) = w(t) + y_s x_s$ where x_s and y_s are the feature and class label of misclassified sample s

Using the weight update rule we get:

$w(t+1) = w(t) + y_s x_s$

$w(1) = w(0) + y_2 x_2$ [since, x_2 is not correctly classified]

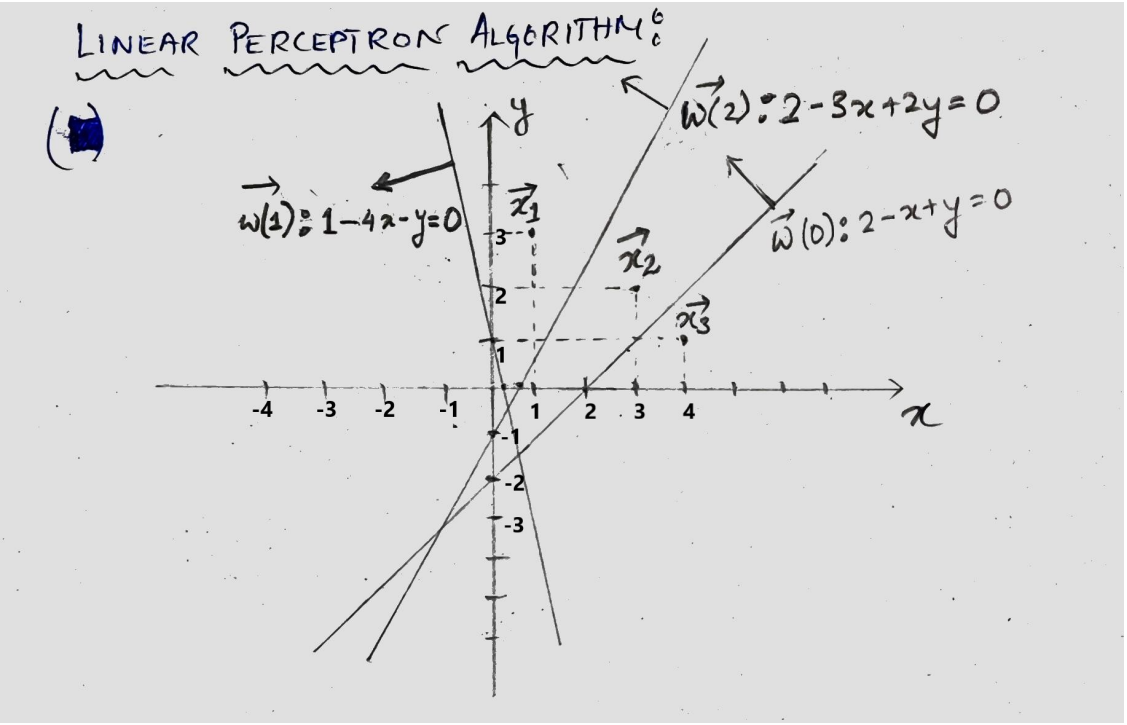
$w(1) = [2,-1,1]^T + (-1) * [1,3,2]^T$

$w(1) = [1,-4,-1]^T$

The line is plotted (below diagram) in the 2D space along with its direction. The line corresponding to w(1) is:

$1 - 4x - y = 0$

In [14]:
```
display(Image(filename="Plot.jpg", width = 600, height = 300))
```



Using weight $w(1)$:

For x_1:

$w(1)^T x_1 = 1 - 4 - 3 = -6 < 0$

$sign(w(1)^T x_1) = sign(-6) = -1$,

therefore the sample x_1 is classified as belonging to class -1 using weight w(1) and it is incorrectly classified

For x_2:

$w(1)^T x_2 = 1 - 4 * 3 - 2 = -13 < 0$

$sign(w(1)^T x_2) = sign(-13) = -1$,

therefore the sample x_2 is classified as belonging to class -1 using weight w(1) and it is correctly classified

For x_3:

$w(1)^T x_3 = 1 - 4 * 4 - 1 = -16 < 0$

$sign(w(1)^T x_1) = sign(-16) = -1$,

therefore the sample x_3 is classified as belonging to class -1 using weight w(1) and it is correctly classified

Hence, sample x_1 $(1, 3)$ is not correctly classified using weight w(1) and samples x_2 and x_3 are correctly classified based on the rule

1.2.25 (1 point) (iv) Using the rule $sign(w(t)^T x_n)$, run the linear perceptron algorithm, find and plot the weights w(2) and the corresponding line. Please indicate which samples are classified correctly and which samples are not classified correctly.

1.2.26 Answer (iv):

Weight update rule:

$w(t+1) = w(t) + y_s x_s$ where x_s and y_s is the feature and class label of misclassified sample s

Using the weight update rule we get:

$w(t+1) = w(t) + y_s x_s$

$w(2) = w(1) + y_1 x_1$ [since, x_1 is not correctly classified]

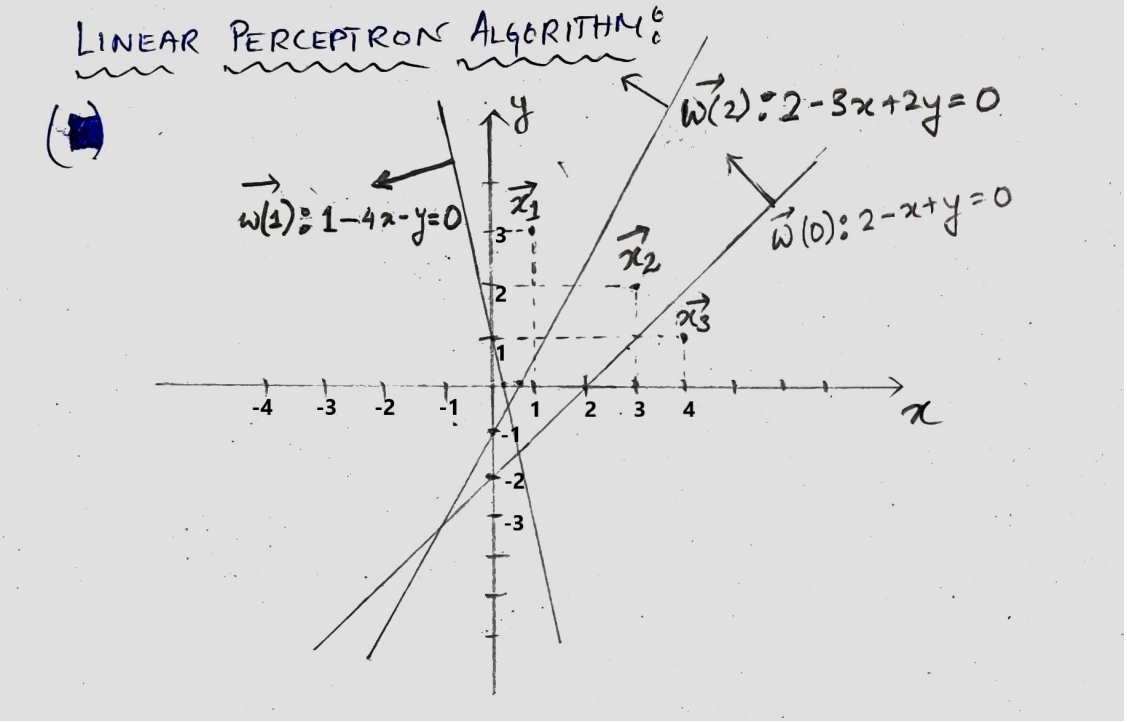
$w(2) = [1, -4, -1]^T + (1) * [1, 1, 3]^T$

$w(2) = [2, -3, 2]^T$

The line is plotted (below diagram) in the 2D space along with its direction. The line corresponding to w(2) is:

$2 - 3x + 2y = 0$

```
In [15]: display(Image(filename="Plot.jpg", width = 600, height = 300))
```



Using weight $w(2)$:

For x_1:

$w(2)^T x_1 = 2 - 3 * 1 + 2 * 3 = 5 > 0$

$sign(w(2)^T x_1) = sign(5) = 1$,

therefore the sample x_1 is classified as belonging to class 1 using weight w(2) and it is correctly classified

For x_2:

$w(2)^T x_2 = 2 - 3 * 3 + 2 * 2 = -3 < 0$

$sign(w(2)^T x_2) = sign(-3) = -1$,

therefore the sample x_2 is classified as belonging to class -1 using weight w(2) and it is correctly classified

For x_3:

$w(2)^T x_3 = 2 - 3 * 4 + 2 * 1 = -8 < 0$

$sign(w(2)^T x_3) = sign(-8) = -1$,

therefore the sample x_3 is classified as belonging to class -1 using weight w(2) and it is correctly classified

Hence, all the samples are correctly classified based on weight w(2)

1.2.27 Question 3 (3 Bonus points)

1.2.28 Please complete the survey provided in this link: https://tamu.qualtrics.com/jfe/form/SV_6QLUBI7p6N9AD0G. Please use your laptop to complete this assignment.

1.2.29 Answer 3:

I have completed the survey