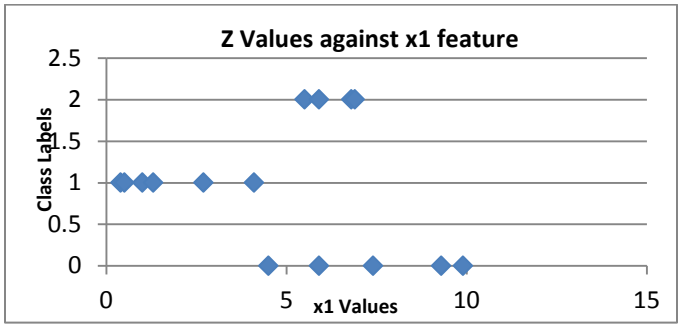
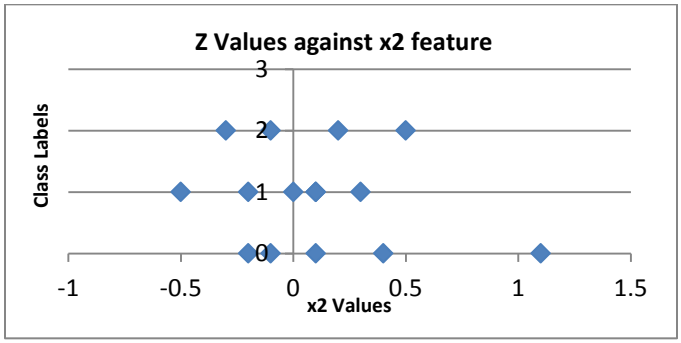
Name: Harshita Agarwala

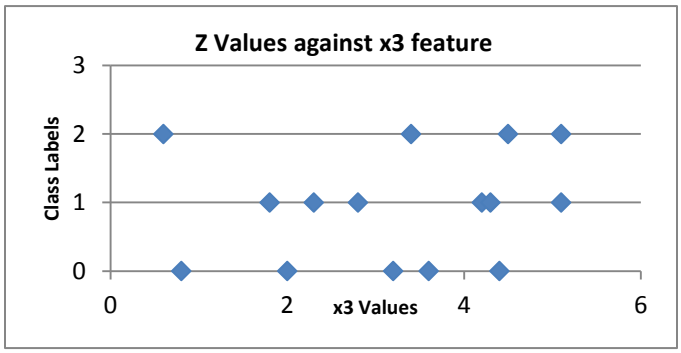
MACHINE LEARNING

Homework 01

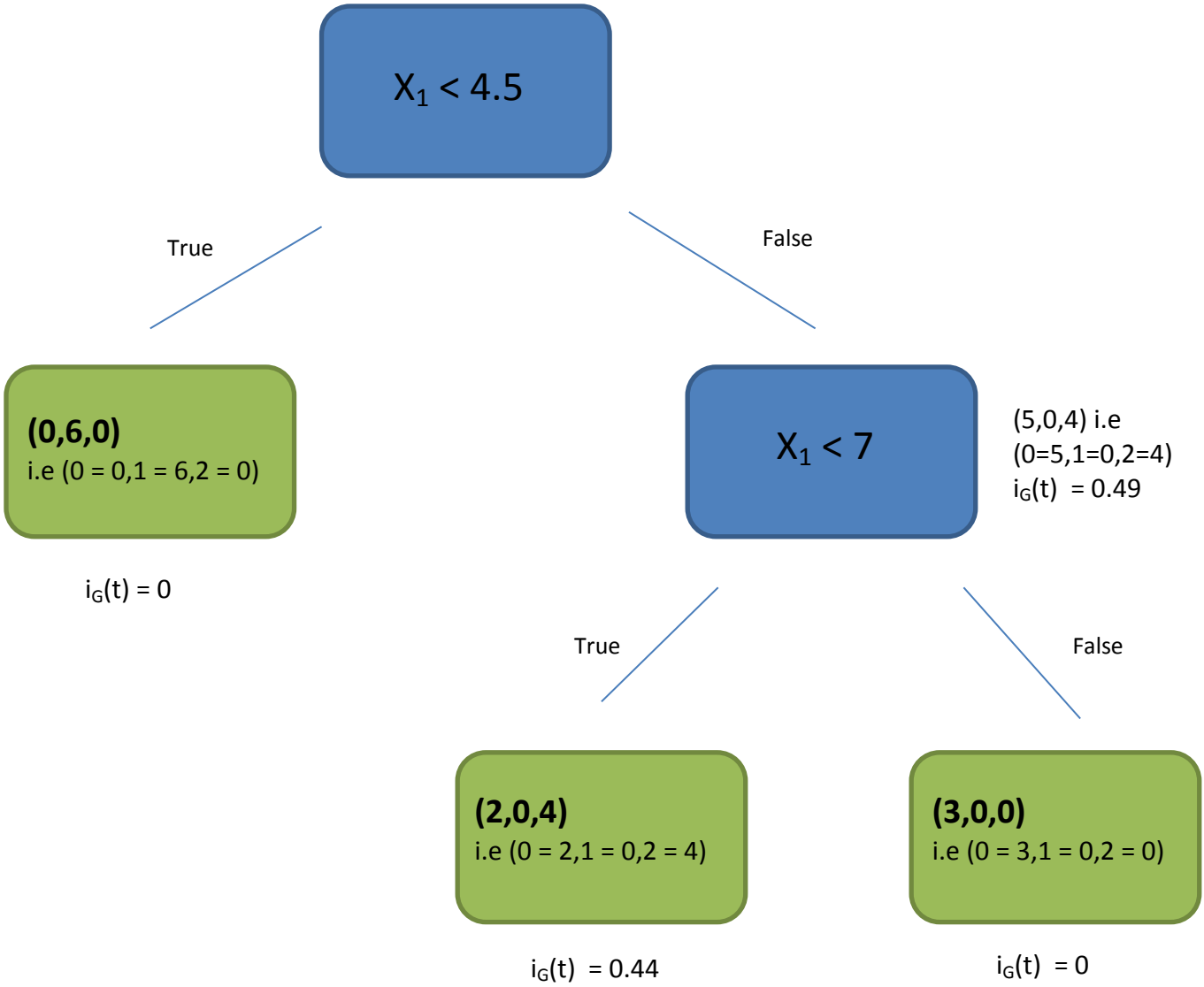
Answer 1:
Using Excel, we first plot the values of z against $x_{i,1}$, $x_{i,2}$ and $x_{i,3}$ respectively. The graphs look like these:







It is clear from the first graph that a split at x1<4.5 will create pure classes.
So, the first split is made at x1<4.5
Similarly, the second split is made at x1<7. The decision tree of a depth of 2 looks like this:


$X_1 < 4.5$
True
False
(0,6,0)
i.e (0 = 0,1 = 6,2 = 0)
$i_G(t) = 0$
$X_1 < 7$
(5,0,4) i.e
(0=5,1=0,2=4)
$i_G(t) = 0.49$
True
False
(2,0,4)
i.e (0 = 2,1 = 0,2 = 4)
$i_G(t) = 0.44$
(3,0,0)
i.e (0 = 3,1 = 0,2 = 0)
$i_G(t) = 0$

Answer 2:

Now, x_a = [4.1,-0.1,2.2] and x_b = [6.1,0.4,1.3]

So, as the first split in the decision tree is basis first feature x_1 we can see that for x_a , $x_1 = 4.1 < 4.5$
Therefore it falls under the classification (0,6,0). As the probability for class 1 is the maximum here, therefore $y_a = 1$ and
$P(c = y_a | x_a , T) = 6/6 = 1$

Again, for x_b , $x_1 = 6.1 > 4.5$ and $6.1 < 7$
Therefore it falls under the classification (2,0,4). As the probability for class 2 is the maximum here, therefore $y_b = 2$ and
$P(c = y_b | x_b , T) = 4/6 = 0.667$
.

Programming assignment 1: k-Nearest Neighbors classification

In [37]:
```
import numpy as np
from sklearn import datasets, model_selection
import matplotlib.pyplot as plt
%matplotlib inline
```

Introduction

For those of you new to Python, there are lots of tutorials online, just pick whichever you like best :)

If you never worked with Numpy or Jupyter before, you can check out these guides

- https://docs.scipy.org/doc/numpy-dev/user/quickstart.html (https://docs.scipy.org/doc/numpy-dev/user/quickstart.html)
- http://jupyter.readthedocs.io/en/latest/ (http://jupyter.readthedocs.io/en/latest/)

Your task

In this notebook code to perform k-NN classification is provided. However, some functions are incomplete. Your task is to fill in the missing code and run the entire notebook.

In the beginning of every function there is docstring, which specifies the format of input and output. Write your code in a way that adheres to it. You may only use plain python and `numpy` functions (i.e. no scikit-learn classifiers).

Once you complete the assignments, export the entire notebook as PDF using nbconvert (https://nbconvert.readthedocs.io/en/latest/) and attach it to your homework solutions. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

Load dataset

The iris data set (https://en.wikipedia.org/wiki/Iris_flower_data_set (https://en.wikipedia.org/wiki/Iris_flower_data_set)) is loaded and split into train and test parts by the function `load_dataset`.

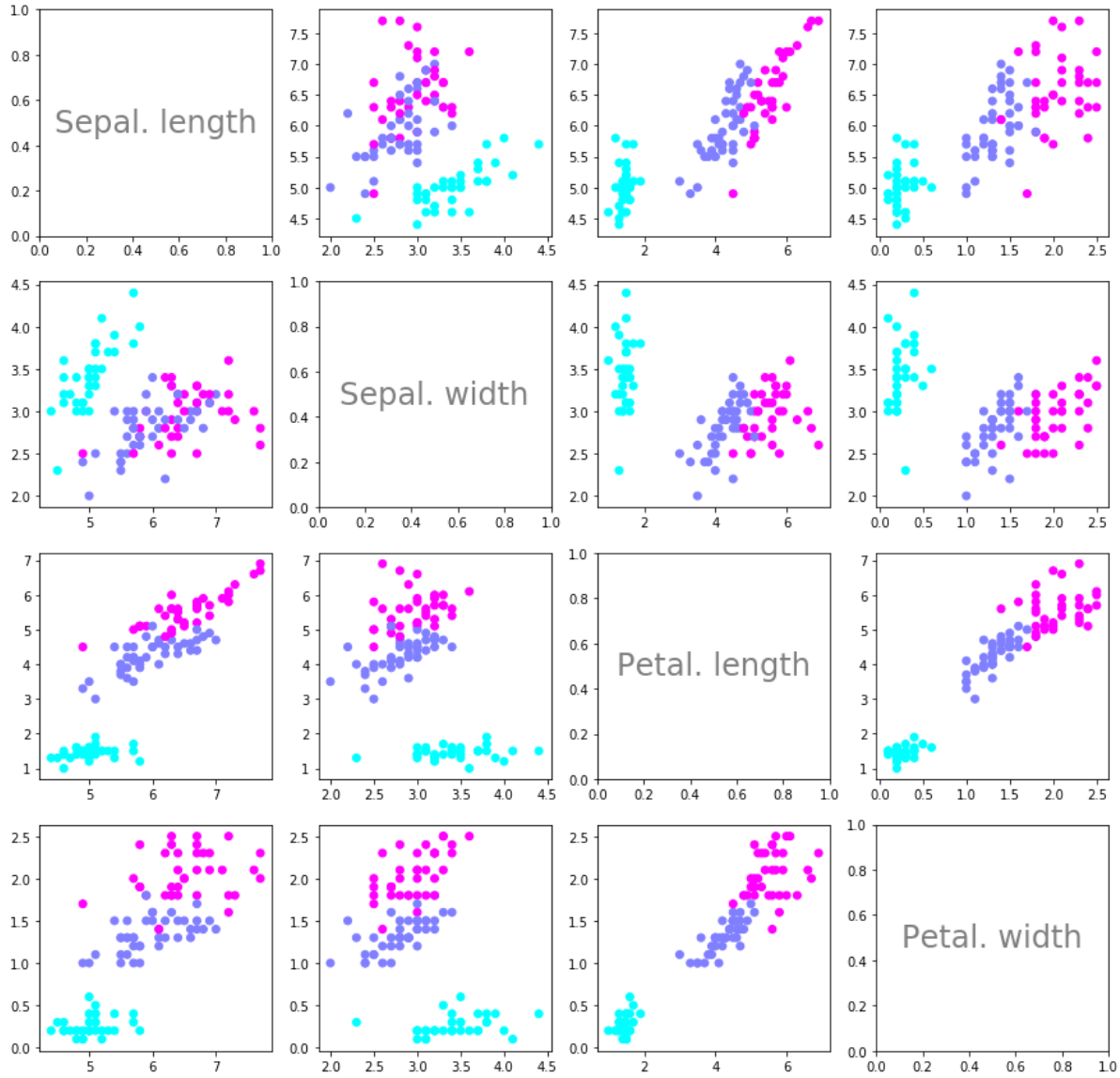
In [38]:
```
def load_dataset(split):
    """Load and split the dataset into training and test parts.

    Parameters
    ----------
    split : float in range (0, 1)
        Fraction of the data used for training.

    Returns
    -------
    X_train : array, shape (N_train, 4)
        Training features.
    y_train : array, shape (N_train)
        Training labels.
    X_test : array, shape (N_test, 4)
        Test features.
    y_test : array, shape (N_test)
        Test labels.
    """
    dataset = datasets.load_iris()
    X, y = dataset['data'], dataset['target']
    X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
random_state=123, test_size=(1 - split))
    return X_train, X_test, y_train, y_test
```

In [39]:
```
# prepare data
split = 0.75
X_train, X_test, y_train, y_test = load_dataset(split)
```

Plot dataset

Since the data has 4 features, 16 scatterplots (4x4) are plotted showing the dependencies between each pair of features.

In [40]:
```
f, axes = plt.subplots(4, 4,figsize=(15, 15))
for i in range(4):
    for j in range(4):
        if j == 0 and i == 0:
            axes[i,j].text(0.5, 0.5, 'Sepal. length', ha='center',
va='center', size=24, alpha=.5)
        elif j == 1 and i == 1:
            axes[i,j].text(0.5, 0.5, 'Sepal. width', ha='center', va='center',
 size=24, alpha=.5)
        elif j == 2 and i == 2:
            axes[i,j].text(0.5, 0.5, 'Petal. length', ha='center',
va='center', size=24, alpha=.5)
        elif j == 3 and i == 3:
            axes[i,j].text(0.5, 0.5, 'Petal. width', ha='center', va='center',
 size=24, alpha=.5)
        else:
            axes[i,j].scatter(X_train[:,j],X_train[:,i], c=y_train, cmap=plt.c
m.cool)
```



Task 1: Euclidean distance

Compute Euclidean distance between two data points.

In [41]:
```
import math

def euclidean_distance(x1, x2):
    """Compute Euclidean distance between two data points.

    Parameters
    ----------
    x1 : array, shape (4)
        First data point.
    x2 : array, shape (4)
        Second data point.

    Returns
    -------
    distance : float
        Euclidean distance between x1 and x2.
    """
    # TODO

    distance = 0
    combined_vector = zip(x1, x2)
    for x in combined_vector:
        distance += (x[1] - x[0]) ** 2
    return float((distance)**(0.5))
```

Task 2: get k nearest neighbors' labels

Get the labels of the *k* nearest neighbors of the datapoint *x_new*.

In [56]:
```
import operator
def get_neighbors_labels(X_train, y_train, x_new, k):
    """Get the labels of the k nearest neighbors of the datapoint x_new.

    Parameters
    ----------
    X_train : array, shape (N_train, 4)
        Training features.
    y_train : array, shape (N_train)
        Training labels.
    x_new : array, shape (4)
        Data point for which the neighbors have to be found.
    k : int
        Number of neighbors to return.

    Returns
    -------
    neighbors_labels : array, shape (k)
        Array containing the labels of the k nearest neighbors.
    """
    # TODO
    distances = []
    for x in range(0,len(X_train)):
        dist = euclidean_distance(x_new, X_train[x])
        distances.append((y_train [x], dist))

    distances.sort(key=operator.itemgetter(1))
    neighbors = []

    for y in range(0,k):
        neighbors.append(distances[y][0])
    return neighbors
```

Task 3: get the majority label

For the previously computed labels of the *k* nearest neighbors, compute the actual response. I.e. give back the class of the majority of nearest neighbors. In case of a tie, choose the "lowest" label (i.e. the order of tie resolutions is 0 > 1 > 2).

In [46]:
```
from scipy import stats

def get_response(neighbors_labels, num_classes=3):
    y=stats.mode(neighbors_labels)

    return y[0][0]
```

Task 4: compute accuracy

Compute the accuracy of the generated predictions.

In [59]:
```
def compute_accuracy(y_pred, y_test):
    """Compute accuracy of prediction.
    Parameters
    ----------
    y_pred : array, shape (N_test)
        Predicted labels.
    y_test : array, shape (N_test)
        True labels.
    """
    correct_count=0
    for i in range(0,len(y_pred)):
        #print(y_pred[i],i)
        if y_pred[i]==y_test[i]:
            correct_count += 1
    #print(correct_count)
    accuracy = float((correct_count/len(y_pred)))
    return accuracy
```

In [48]:
```
# This function is given, nothing to do here.
def predict(X_train, y_train, X_test, k):
    """Generate predictions for all points in the test set.

    Parameters
    ----------
    X_train : array, shape (N_train, 4)
        Training features.
    y_train : array, shape (N_train)
        Training labels.
    X_test : array, shape (N_test, 4)
        Test features.
    k : int
        Number of neighbors to consider.

    Returns
    -------
    y_pred : array, shape (N_test)
        Predictions for the test data.
    """
    y_pred = []
    for x_new in X_test:
        neighbors = get_neighbors_labels(X_train, y_train, x_new, k)
        y_pred.append(get_response(neighbors))
    return y_pred
```

Testing

Should output an accuracy of 0.9473684210526315.

In [60]:
```
# prepare data
split = 0.75
X_train, X_test, y_train, y_test = load_dataset(split)
print('Training set: {0} samples'.format(X_train.shape[0]))
print('Test set: {0} samples'.format(X_test.shape[0]))

# generate predictions
k = 3
y_pred = predict(X_train, y_train, X_test, k)
accuracy = compute_accuracy(y_pred, y_test)
print('Accuracy = {0}'.format(accuracy))
```

```
Training set: 112 samples
Test set: 38 samples
Accuracy = 0.9473684210526315
```

In [3]:
```
## Problem 4
"""
Classify the two vectors xa and xb given in Problem 2 with the k-nearest neighbors algo-
rithm. Use k = 3 and Euclidean distance.
"""
```

Out[3]:
```
'\nClassify the two vectors xa and xb given in Problem 2 with the k-nearest neighbors algo-\nrithm. Use k = 3 and Euclidean distance.\n'
```

In [37]:
```
#Importing Packages
import numpy as np
from sklearn import datasets, model_selection
import matplotlib.pyplot as plt
%matplotlib inline
```

In [36]:
```
#Defining Euclidean Distance function
def euclidean_distance(x1, x2):
    distance = 0
    combined_vector = zip(x1, x2)
    for x in combined_vector:
        distance += (x[1] - x[0]) ** 2
    return float((distance)**(0.5))
```

In [35]:
```
#Defining k-nearest neighbors
import operator
def get_neighbors_labels(X_train, y_train, x_new, k):

    distances = []
    for x in range(len(X_train)):
        dist = euclidean_distance(x_new, X_train[x])
        distances.append((y_train [x], dist))

    distances.sort(key=operator.itemgetter(1))
    neighbors = []

    for y in range(k):
        neighbors.append(distances[y][0])
    return neighbors
```

In [34]:
```
#Defining Maximum occuring label
from scipy import stats

def get_response(neighbors_labels, num_classes=3):
    y=stats.mode(neighbors_labels)

    return y[0][0]
```

In [47]:
```
k = 3
x_A = [4.1,-0.1,2.2]
x_B = [6.1,0.4,1.3]
import numpy as np
import pandas as pd
x_data = np.array([[5.5,0.5,4.5],[7.4,1.1,3.6],[5.9,0.2,3.4],[9.9,0.1,0.8],[6.
9,-0.1,0.6],[6.8,-0.3,5.1],[4.1,0.3,5.1],
                   [1.3,-0.2,1.8],[4.5,0.4,2],[0.5,0,2.3],[5.9,-0.1,4.4],
[9.3,-0.2,3.2],[1,0.1,2.8],[0.4,0.1,4.3],[2.7,-0.5,4.2],
                ])

y_data = np.array([2,0,2,0,2,2,1,1,0,1,0,0,1,1,1])

neighbors_A=get_neighbors_labels(x_data, y_data, x_A, k)
print("The neighbors of x_A are:" + str(neighbors_A)+ "and the predicted value
 of y_A is:" + str(get_response(neighbors_A)))

neighbors_B=get_neighbors_labels(x_data, y_data, x_B, k)
print("The neighbors of x_B are:" + str(neighbors_B)+ "and the predicted value
 of y_B is:" + str(get_response(neighbors_B)))
```

```
The neighbors of x_A are:[0, 2, 1]and the predicted value of y_A is:0
The neighbors of x_B are:[2, 0, 2]and the predicted value of y_B is:2
```

In [3]:
```
## Problem 5
"""
Now, consider yi to be real-valued targets rather than classes. Perform 3-NN r
egression to
label the vectors from Problem 2.
"""
```

Out[3]:
```
'\nNow, consider yi to be real-valued targets rather than classes. Perform 3-
NN regression to\nlabel the vectors from Problem 2.\n'
```

In [4]:
```
#Importing Packages
import numpy as np
from sklearn import datasets, model_selection
import matplotlib.pyplot as plt
%matplotlib inline
```

In [5]:
```
#Defining Euclidean Distance function
def euclidean_distance(x1, x2):
    distance = 0
    combined_vector = zip(x1, x2)
    for x in combined_vector:
        distance += (x[1] - x[0]) ** 2
    return float((distance)**(0.5))
```

In [9]:
```
#Defining k-nearest neighbors
import operator
def get_neighbors_labels(X_train, y_train, x_new, k):

    distances = []
    for x in range(len(X_train)):
        dist = euclidean_distance(x_new, X_train[x])
        distances.append((y_train [x], dist))

    distances.sort(key=operator.itemgetter(1))
    neighbors = []

    for y in range(k):
        neighbors.append((distances[y][0],distances[y][1]))
    return neighbors
```

In [15]:
```
#Defining Weighted mean to predict value
def get_response(neighbors_labels, num_classes=3):

    labels=[]
    dist=[]
    Z=0
    Mean=0
    for i in range(0,len(neighbors_labels)):
        labels.append(neighbors_labels[i][0])
        dist.append(neighbors_labels[i][1])
        Z+=float(1/(dist[i]))
        Mean+=float((1/dist[i])*labels[i])

    y=float((1/Z)*Mean)
    return round(y,5)
```

In [16]:
```
k = 3
x_A = [4.1,-0.1,2.2]
x_B = [6.1,0.4,1.3]
import numpy as np
import pandas as pd
x_data = np.array([[5.5,0.5,4.5],[7.4,1.1,3.6],[5.9,0.2,3.4],[9.9,0.1,0.8],[6.
9,-0.1,0.6],[6.8,-0.3,5.1],[4.1,0.3,5.1],
                   [1.3,-0.2,1.8],[4.5,0.4,2],[0.5,0,2.3],[5.9,-0.1,4.4],
[9.3,-0.2,3.2],[1,0.1,2.8],[0.4,0.1,4.3],[2.7,-0.5,4.2],
                ])

y_data = np.array([2,0,2,0,2,2,1,1,0,1,0,0,1,1,1])

neighbors_A=get_neighbors_labels(x_data, y_data, x_A, k)
print("The neighbors of x_A and their distances are:" + str(neighbors_A)+ "and
 the predicted value of y_A is:" +
      str(get_response(neighbors_A)))

neighbors_B=get_neighbors_labels(x_data, y_data, x_B, k)
print("The neighbors of x_B and their distances are:" + str(neighbors_B)+ "and
 the predicted value of y_B is:" +
      str(get_response(neighbors_B)))
```

```
The neighbors of x_A and their distances are:[(0, 0.6708203932499371), (2, 2.
1840329667841556), (1, 2.473863375370596)]and the predicted value of y_A is:
0.56102
The neighbors of x_B and their distances are:[(2, 1.1747340124470735), (0, 1.
7464249196572976), (2, 2.1189620100417086)]and the predicted value of y_B is:
1.39592
```

Answer 6:

It is very clear from Answers to Problems 4 and 5 of the variations that can be caused with the k-NN model. The main problem with Euclidean distance is it searches for the closest point in all directions. It might happen that the labels in a particular region are changing because of some one feature only. For example, it may increase/decrease horizontally basis one feature. In such a case the Euclidean distance does not give the correct result. Some other distance measure should be used for better results. During decision trees, the result is not based on nearest neighbors but smaller regions of the data. Hence, during decision trees, the Euclidean distance is not important