K-Nearest Neighbour Algorithm

PROBLEM STATEMENT:

To predict the Weight using KNN Algorithm without using inbuilt packages.

FORMULAS USED:

Euclidean Distance:

Distance between any two points (x1,y1) and (x2,y2) is given by

$[(x2-x1)^2 + (y2-y1)^2]^{1/2}$

ALGORITHM:

Step 1 – Load the training and test data.

Step 2 – Choose the value of K i.e. the nearest data points. K can be any integer (preferably not 1, but any other odd value)

Step 3 – For each data point, Calculate the distance between test data and each row of training data with Euclidean Distance Formula.

Step 4 – Based on the distance value, sort them in ascending order.

Step 5 – Next, it will choose the top K rows from the sorted array.

Step 6 – Compute the average of sum of the preceding rows and calculate the percentage error. The predicted value corresponds to the value with the least percentage error.

Step 7 – End

CODE:

@Script Author : Ashwin George

@Description : K-nearest neighbour algorithm without using packages

@Start Date : 07-01-2020

@Last Edited : 09-01-2020

@Python Version : Python 3.7

```
#training set
train=[[40,174,63,5.70,69],[50,126,73,5.00,66],[32,140,72,5.30,85],[48,1
23,64,6.10,88],[28,132,74,4.60,83],[27,178,74,5.20,82],[26,148,77,6.00,8
8],[23,120,68,5.80,75],[38,177,73,4.50,70],[29,101,72,6.30,88]]



#testing set
test=[50,130,70,5.00,80]



#calculating difference between test and train
diff,final_diff=[],[]
for i in range(len(train)):
    diff=[]
    for j in range(len(train[0])-1):
        d=test[j]-train[i][j]
        diff.append(d)
    final_diff.append(diff)
final_diff



#squaring elements in difference matrix
for i in range(len(final_diff)):
    for j in range(len(final_diff[0])):
        final_diff[i][j]=final_diff[i][j]*final_diff[i][j]
final_diff



#finds sqrt of sum of squares in each row
import math
final_sum=[]
sum1=0
for i in range(len(final_diff)):
    for j in range(len(final_diff[0])):
        sum1=sum1+final_diff[i][j]
    sum1=math.sqrt(sum1)
```

```
   final_sum.append(sum1)
final_sum



#a dictionary that maps sqrt values to its corr. training set
dict={final_sum[0]:train[0],final_sum[1]:train[1],final_sum[2]:train[2],
final_sum[3]:train[3],final_sum[4]:train[4],final_sum[5]:train[5],final_
sum[6]:train[6],final_sum[7]:train[7],final_sum[8]:train[8],final_sum[9]:
train[9]}
dict



#sorting the dictionary based on sqrt values
dict1=sorted(dict.items())
dict1



#finds cumulative sum of target values in training set as per the value of
k
sum=0
cum_sum=[]
for i in range(len(dict1)):
    sum=sum+dict1[i][1][4]
    cum=sum/(i+1)
    cum_sum.append(cum)
cum_sum



#finds the least abs percentage error (actual - predicted(cum_sum)/actual)
per_error=[]
for i in range(len(train)):
    di=test[4]-cum_sum[i]
    per_error.append(abs((di/test[4])*100))
per_error
dict_fin={per_error[0]:cum_sum[0],per_error[1]:cum_sum[1],per_error[2]:c
um_sum[2],per_error[3]:cum_sum[3],per_error[4]:cum_sum[4],per_error[5]:c
um_sum[5],per_error[6]:cum_sum[6],per_error[7]:cum_sum[7],per_error[8]:c
um_sum[8],per_error[9]:cum_sum[9]}
```

OUTPUT:

```
The Predicted value is 80.25
The actual value is 80
The percentage error is 0.3125
```