K-Nearest Neighbour Algorithm

PROBLEM STATEMENT:

An attempt to predict the Weight using KNN Algorithm without any inbuilt packages.

IMPORTANT FORMULAS USED:

Euclidean Distance Formula:

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

$\sqrt{[(x2-x1)^2 + (y2-y1)^2]}$

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 point in the test data do the following –

- **3.1** – Calculate the distance between test data and each row of training data with Euclidean Distance Formula.
- **3.2** – Based on the distance value, sort them in ascending order.
- **3.3** – Next, it will choose the top K rows from the sorted array.
- **3.4** – 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 4 – End

CODE:

```
@Script Author : linto sebastian
@Description   : K-nearest neighbour algorithm without any packages
@Start Date    : 07-01-2020
@Last Edited   : 11-01-2020
@Python Version : Python 3.7.3



#Defining the train and test data
#initialising empty  lists
dist,final,diff=[],[],[]

#training data
train=[[5,8,15],[7,9,20],[2,3,11],[8,10,22],[4,5,7]]

#testing data
test=[10,15,25]
n=len(train) #length of training data

# finding the difference and appending the difference into final
list as lists
for i in range(n):
    diff=[]
    for j in range(len(test)-1):
        x=test[j]-train[i][j]
        diff.append(x)
    final.append(diff)

# finding the euclidean distance
for i in range(n):
```

```
    s=0
    for j in range(len(test)-1):
        s=s+final[i][j]**2
    dist.append(s**0.5)
dist


# mapping the distance to corresponding element in training data in
dictionary
dic={}
for m in range(len(dist)):
    dic[dist[m]]=train[m]
dic


#sorting the distance in ascending order
q=sorted(dic.items())
q

#calculating the cumilative sum and hence the average using k values
predicted=[]
c_sum=0
for u in range(n):
    c_sum=c_sum+q[u][1][2]
    avg=c_sum/(u+1)
    print(c_sum)
    print(avg)
    predicted.append(avg)
predicted

#calculating the percentage error
z=0
per_error=[]
for s in range(n):
    z=abs(((test[2]-predicted[s])*100)/(test[2]))
    per_error.append(z)
per_error
```

```
#defining a dictionary which maps percentage error to predicted
value
dictn={}
for w in range(n):
    dictn[per_error[w]]=predicted[w]
dictn


#printing the predicting value
print("predicted value is ",dictn[min(per_error)])


#printing the actual value
print("actual value is ",test[len(test)-1])


#printing the minimum percentage error
print("percentage error is ",min(per_error))
```

OUTPUT:

```
predicted value is  22.0
actual value is  25
percentage error is  12.0
```