Experiment 7 ,8 Date:2/4/2025

Name-Priayansh Panchal

Roll no-24bee106

```
#Distance weighted KN regression for predicting selling price in Boston
#House price data. We are not going to use KNeighborsRegressor from sklearn

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
#read the data
data=pd.read_csv("BostonHousing.csv",header='infer').values

#separt input and target/output part of data
X=data[:,0:-1]
y=data[:,-1]

test_split=float(input("Enter a number between 0 to 1 to specify how
much data is required:"))
#split the data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=test_spli
t)

dist=np.zeros(shape=X_train.shape[0])
pred=np.zeros(shape=X_test.shape[0])

                                                    #for storing prediction


#ask the user about number of nearest neighbors to be used ,i.e. k
k=int(input("Enter the number of nearest neighbours to be used,i.e. k:
"))

for i in range(X_test.shape[0]):
      dist=np.sqrt(np.sum((X_train-X_test[i])**2,axis=1))                    #calc
Euclidean dist
      #between currnt test record and all training records
      kminind=np.argpartition(dist,k)[0:k] #finfing indices of k minimum
distances
invdist=1/(dist+10e-20)
denom=sum(invdist[kminind]) #for weight normalisation
pred[i]=np.dot(invdist[kminind]/denom,y_train[kminind])
#print(pred.shape)
#print(kminind)
#print(pred)
#UDF to calc MAE
def MAE(pred,y_test):

      return np.mean(abs(pred-y_test))

#UDF to calc MSE
def MSE(pred,y_test):
      return np.mean((pred-y_test)**2)
```

```
#UDF to calc MAPE
def MAPE(pred,y_test):
      return np.mean(abs((pred-y_test)/y_test))


#calc performance measures mae=MAE(pred,y_test) mse=MSE(pred,y_test)
rmse=np.sqrt(mse) mape=MAPE(pred,y_test) print(mae) print(mse) print(rmse)
print(mape)

#NOW using sklearn
from sklearn.neighbors import KNeighborsRegressor
regressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
#performance m

#creatign instance opf class                                    #class for KNN
model=KNeighborsRegressor(n_neighbors=k, weights='distance')

#training model
model.fit(X_train,y_train)
#using the trained model for making prediction
pred=model.predict(X_test)

#calc performance measure
mae=mean_absolute_error(y_test,pred)
mse=mean_squared_error(y_test,pred)
print("Using Sklear:\nMAE:",mae)
print("MSE:",mse)
```

```
Enter a number between 0 to 1 to specify how much data is required:
0.2
Enter the number of nearest neighbours to be used,i.e. k:

4.505199443581134
48.64868125211801
6.974860661842501
0.19504709173038556                                                5
Using Sklear:
MAE: 4.505199443581133
MSE: 48.64868125211801
```

```
#EXPERIMENT 8
#clustering iris flower using k-means and without sklearn for k-means
#we will use Euclidean distance as the distance measure

import pandas as pd
```

```
import numpy as np
from sklearn.model_selection import train_test_split              #to split data
in train and test
from sklearn.metrics import classification_report                 #for
classification report

data=pd.read_csv("Iris.csv",header='infer').values
X=data[:,1:-1]
y=data[:,-1]

test_split=float(input("Enter a number between 0 to 1 to specify how
much data is required:"))
#split the data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=test_spli
t,stratify=y)
#ask the user about number of nearest neighbors to be used ,i.e. k
k=int(input("Enter the number of nearest neighbours to be used,i.e. k:
"))
#number of iterations
n=int(input("Enter the number of iterations you want to run
algorithm:"))

#array to store centroids and final centroids
centroids=np.zeros(shape=(k,X_train.shape[1]))
#for selceting rand train poijnts as initial centroids
per=np.random.permutation(X_train.shape[0])
#select random train points as init centroids
for i in range(k):

    centroids[i,:]=X_train[per[i],:]

for it in range(n):
    dist=np.zeros(shape=(k,X_train.shape[0]))          #for storing dist bw
each
# pair of centroids and trainig data points
for i in range(k):          #compute these distance
                        dist[i,:]=np.sqrt(np.sum((X_train-centroids[i,:])**2,axis=1))
membership=np.argmin(dist,axis=0)          #decide membership for each
training dataponts
for i in range(k): #update centroids before we start with iteration
    centroids[i,:]=np.mean(X_train[membership==i,:],axis=0)
print("Centroids after "+ str(n) + "iterations:")
print(centroids)
#now inference time
dist=np.zeros(shape=(k,X_test.shape[0]))
for i in range(k):
    dist[i]=np.sqrt(np.sum(X_test-centroids[i])**2,axis=1)
```

```
membership=np.argmin(dist,axis=0)

#display predicted membership.
#class number and cluster number may not be same, so be careful while evaluating
print(y_test.astype(int))
print(membership)
```

```
Enter a number between 0 to 1 to specify how much data is required: .2
Enter the number of nearest neighbours to be used,i.e. k:               3
Enter the number of iterations you want to run algorithm: 80000

Centroids after 80000iterations:
[[4.54444444 3.05555556 1.38888889 0.22222222]
 ][5.                3.3                1.44117647 0.2
  [6.07978723 2.99042553 4.37553191 1.47021277]]
---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
Cell In[29], line 47
        45 dist=np.zeros(shape=(k,X_test.shape[0]))
        46 for i in range(k):
---> 47         dist[i]=np.sqrt(np.sum(X_test-centroids[i])**2,axis=1)
        48 membership=np.argmin(dist,axis=0)
        50 #display predicted membership.
        51 #class number and cluster number may not be same, so be
careful while evaluating

TypeError: sqrt() got an unexpected keyword argument 'axis'
```