KNN – > K - Nearest – Neighbor

About:

1.KNN is a supervised learning algorithm.

2.It used for classification and regression problems, but mostly used for classification

3. Some of KNN application are Handwriting Recognition, satellite Image Recognition ECG pattern recognition

How does the KNN algorithm work?

Let's say you have you have 2 classes (BLUE/ RED) for classification. New data is inserted in the algorithm, the new data can be a RED / BLUE. The K in KNN algorithm is the nearest neighbor we wish to take the vote from. Let's say K=3, Now we make a circle with new data as center. when 3 train datapoint will come inside the circle we will close it and check the closet point are RED /Blue. Now, we will start the vote if 2 datapoints are RED then as per KNN it belong to RED class

When do we use KNN algorithm?

In General, we Use for its easy of interpretation and low calculation time . KNN can compete with most of accurate models because it makes highly accurate prediction, Therefore you can use the KNN model for application where you required high accuracy but that do not require a human readable model . The quality of the predictions depends on the distance measure.

How to choose K factor?

The training error rate and the validation error rate are two parameters we need to access different K-value based on the result we can say what should be the value of K and we can use that for all prediction.

Advantages:

1. No training period: KNN is called Lazy learning . It does not learn anything in the training period. It stores the training dataset and learn from it only at the time of prediction. This make KNN faster the other algorithm that requires training e.g. SVM, Linear regression
2. New Data can be added seamlessly which will not impact the accuracy of the algorithm.
3. KNN is very easy to implement There are only two parameters required to implement KNN

Disadvantage:

1. Does not work well with large dataset because calculation distance every time make it costly
2. Does not work well with high dimension because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.
3. Need feature scaling: we need it do feature scaling (Standardization and normalization) before applying KNN algorithm to any dataset. If we don't do so, KNN may generate wrong predictions
4. Sensitive to noising data, Missing the value of outliers: KNN is sensitive to noise in the dataset. we need to manually impute missing value and remove outliers

Implemention :

```
import numpy as np
from collections import Counter

# Function which help model to get the distance
def eucliendean_distance(x1,x2):
    return np.sqrt(np.sum((x1-x2)**2))

# Model creation
class KNN:
    def __init__(self,k=3,):
        self.k = k
    # Like sklearn we have a fit where we put the dataset
    def fit(self,X,y):
        self.X_train = X
        self.y_train = y
    #in predict we are pasing the X train dataset
    def predict(self,X):
        #Using _predict function to give us the result in list format
        predicted_labels = [self._predict(x) for x in X]
        #coverting the list into array
        return np.array(predicted_labels)
```

```
    def _predict(self,x):
        # compute the distances with def eucliendean_distance(x1,x2): function
        distances  = [eucliendean_distance(x,x_train) for x_train in self.X_train]
        # Get K nearest sample,label
        #Sort the list of distance and get the indices of k nearest sample
        k_indices = np.argsort(distances)[:self.k]
        #get the k nearest label for k_indices
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        # majority vote,most common class label
        #find the class label which come mostly for the value
        most_common =  Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]
```

Using the model to Predict:

```
# Here i m using iris dataset
from sklearn import datasets
iris = datasets.load_iris()
X,y = iris.data,iris.target
from sklearn.model_selection import train_test_split
Xtrain,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2, random_state = 1234)
# Creating and object of the class
KNN_clf = KNN(k=5)
KNN_clf.fit(Xtrain,y_train)
predict = KNN_clf.predict(X_test)
acc_score = np.sum(predict == y_test)/len(y_test)
print(acc_score)
#Output : 0.9666666666666667
```