Program – 13

AIM: To Implement K-nearest neighbours classifier in python

Introduction and Theory

The k-nearest-neighbor method was first described in the early 1950s. The method is labor intensive when given large training sets, and did not gain popularity until the 1960s when increased computing power became available. It has since been widely used in the area of pattern recognition.
Nearest-neighbor classifiers are based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it. The training tuples are described by n attributes. Each tuple represents a point in an n-dimensional space. In this way, all the training tuples are stored in an n-dimensional pattern space. When given an unknown tuple, a k-nearest-neighbor classifier searches the pattern space for the k training tuples that are closest to the unknown tuple. These k training tuples are the k "nearest neighbors" of the unknown tuple.

For k-nearest-neighbor classification, the unknown tuple is assigned the most common class among its k-nearest neighbors. When k = 1, the unknown tuple is assigned the class of the training tuple that is closest to it in pattern space. Nearest-neighbor classifiers can also be used for numeric prediction, that is, to return a real-valued prediction for a given unknown tuple. In this case, the classifier returns the average value of the real-valued labels associated with the k-nearest neighbors of the unknown tuple

Strengths of KNN

- K-NN is pretty intuitive and simple: K-NN algorithm is very simple to understand and equally easy to implement. To classify the new data point K-NN algorithm reads through whole dataset to find out K nearest neighbours.
- K-NN has no assumptions: K-NN is a non-parametric algorithm which means there are assumptions to be met to implement K-NN. Parametric models like linear regression has lots of assumptions to be met by data before it can be implemented which is not the case with K-NN.
- No Training Step: K-NN does not explicitly build any model, it simply tags the new data entry based learning from historical data. New data entry would be tagged with majority class in the nearest neighbour.
- It constantly evolves: Given it's an instance-based learning; k-NN is a memory-based approach. The classifier immediately adapts as we collect new training data. It allows the algorithm to respond quickly to changes in the input during real-time use.
- Very easy to implement for multi-class problem: Most of the classifier algorithms are easy to implement for binary problems and needs effort to implement for multi class whereas K-NN adjust to multi class without any extra efforts.
- Can be used both for Classification and Regression: One of the biggest advantages of K-NN is that K-NN can be used both for classification and regression problems.
- One Hyper Parameter: K-NN might take some time while selecting the first hyper parameter but after that rest of the parameters are aligned to it.
- Variety of distance criteria to be choose from: K-NN algorithm gives user the flexibility to choose distance while building K-NN model.
    - Euclidean Distance

Program – 13

- Hamming Distance
- Manhattan Distance
- Minkowski Distance

Weaknesses of KKN:

- K-NN slow algorithm: K-NN might be very easy to implement but as dataset grows efficiency or speed of algorithm declines very fast.
- Curse of Dimensionality: KNN works well with small number of input variables but as the numbers of variables grow K-NN algorithm struggles to predict the output of new data point.
- K-NN needs homogeneous features: If you decide to build k-NN using a common distance, like Euclidean or Manhattan distances, it is completely necessary that features have the same scale, since absolute differences in features weight the same, i.e., a given distance in feature 1 must means the same for feature 2.
- Optimal number of neighbors: One of the biggest issues with K-NN is to choose the optimal number of neighbors to be consider while classifying the new data entry.
- Imbalanced data causes problems: k-NN doesn't perform well on imbalanced data. If we consider two classes, A and B, and the majority of the training data is labeled as A, then the model will ultimately give a lot of preference to A. This might result in getting the less common class B wrongly classified.
- Outlier sensitivity: K-NN algorithm is very sensitive to outliers as it simply chose the neighbors based on distance criteria.
- Missing Value treatment: K-NN inherently has no capability of dealing with missing value problem

Code

```
# importing libraries

import pandas as pd
import numpy as np
import math
import operator


def get_train_test_split(dataset, split_ratio=0.8):
    datalen = len(dataset)
    shuffled = dataset.iloc[np.random.permutation(len(dataset))]
    train_data = shuffled[:int(split_ratio * datalen)]
    test_data = shuffled[int(split_ratio * datalen):]
    return train_data, test_data

def euclidean_dist(p1, p2):
    dist = 0.0
    for i in range(len(p1)-1):
        dist += pow((float(p1[i])-float(p2[i])),2)
        dist = math.sqrt(dist)
    return dist

def knn(training_dataset, test_value, k):
    distances = {}
```

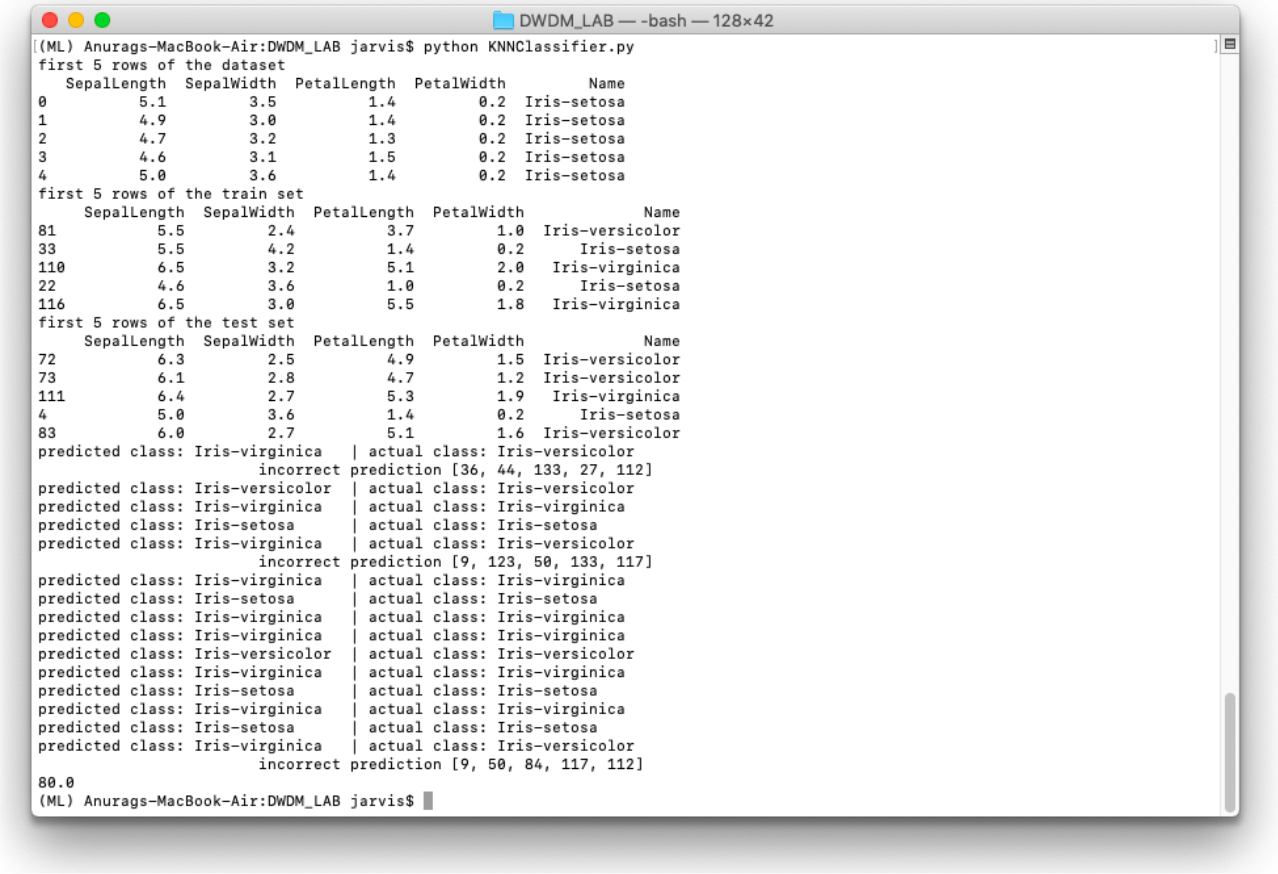
Program – 13

```
    sort = {}

    # length = test_value.shape[1]

    # Calculating euclidean distance between each row of training
data and test data
    for x in range(len(training_dataset)):
        dist = euclidean_dist(test_value, training_dataset.iloc[x])

        distances[x] = dist
    # Sorting them on the basis of distance
    sorted_d = sorted(distances.items(), key=operator.itemgetter(1))

    neighbors = []

    # Extracting top k neighbors
    for x in range(k):
        neighbors.append(sorted_d[x][0])
    class_votes = {}

    # Calculating the most freq class in the neighbors
    for x in range(len(neighbors)):
        response = training_dataset.iloc[neighbors[x]][-1]

        if response in class_votes:
            class_votes[response] += 1
        else:
            class_votes[response] = 1

    sorted_votes = sorted(class_votes.items(),
key=operator.itemgetter(1), reverse=True)
    return sorted_votes[0][0], neighbors

def cal_accuracy(data, test_data, K=5):
    correct = 0
    total = len(test_data)

    for i in range(total):
        pred_class, neigh = knn(data, test_data.iloc[i], K)
        print("predicted class: {:16} | actual class:
{:16}".format(pred_class, test_data.iloc[i]['Name']))
        if pred_class == test_data.iloc[i]['Name']:
            correct += 1
        else:
            print("\t\t\tincorrect prediction", neigh)

    return (correct/total) * 100

if __name__ == '__main__':

    dataset = pd.read_csv('iris.csv')
    print('first 5 rows of the dataset')
    print(dataset.head(5))
    train_data, test_data = get_train_test_split(dataset)
    print('first 5 rows of the train set')
    print(train_data.head(5))
    print('first 5 rows of the test set')
```

Program – 13

```
82      print(test_data.head(5))
83      print(cal_accuracy(train_data, test_data, 5))
84
```

Results and Outputs:



Findings and Learnings:

1. We have Implemented K-nearest-neighbours algorithm in python 3.
2. We have learned the nuances of the KNN-algorithm.
3. We have learnt about the applications, strengths and weaknesses of KNN-algorithm.