Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

JUNAID GIRKAR
60004190057
TE COMPS A4

EXPERIMENT - 5

Aim: Implementation of K Nearest Neighbours (KNN) from scratch.

Theory:

K-Nearest Neighbours is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.
It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data.
We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

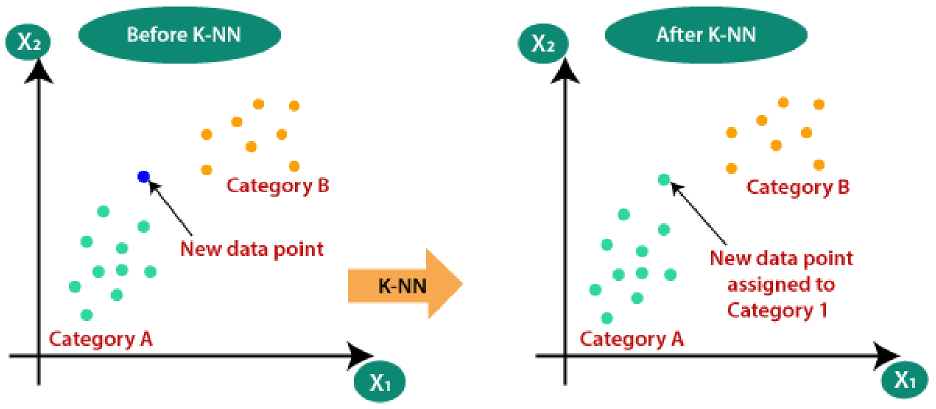
Algorithm
Let m be the number of training data samples. Let p be an unknown point.

- Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).

```
for i=0 to m:
  Calculate Euclidean distance d(arr[i], p).
```

- Make set S of K smallest distances obtained. Each of these distances corresponds to an already classified data point.
- Return the majority label among S.

K can be kept as an odd number so that we can calculate a clear majority in the case where only two groups are possible (e.g. Red/Blue). With increasing K, we get smoother, more defined boundaries across different classifications. Also, the accuracy of the above classifier increases as we increase the number of data points in the training set.



Code:

```
import csv
import random

# load iris dataset and randomly split it into test set and training set

def loadDataset(filename, split, trainingSet=[] , testSet=[]):
      with open(filename, 'rt') as csvfile:
          lines = csv.reader(csvfile)
          dataset = list(lines)
          for x in range(len(dataset)-1):
              for y in range(4):
                  dataset[x][y] = float(dataset[x][y])
              if random.random() < split:
                  trainingSet.append(dataset[x])
              else:
                  testSet.append(dataset[x])


#############################
# SIMILARITY CHECK FUNCTION #
#############################

# euclidean distance calcualtion

import math
def euclideanDistance(instance1, instance2, length):
      distance = 0
      for x in range(length):
            distance += pow((instance1[x] - instance2[x]), 2)
      return math.sqrt(distance)
```

```
#############################################################
# NEIGHBOURS - selecting subset with the smallest distance #
#############################################################

import operator
def getNeighbors(trainingSet, testInstance, k):
      distances = []
      length = len(testInstance)-1
      for x in range(len(trainingSet)):
            dist = euclideanDistance(testInstance, trainingSet[x],
length)
            distances.append((trainingSet[x], dist))
      distances.sort(key=operator.itemgetter(1))
      neighbors = []
      for x in range(k):
            neighbors.append(distances[x][0])
      return neighbors


######################
# PREDICTED RESPONSE #
######################

import operator
def getResponse(neighbors):
      classVotes = {}
      for x in range(len(neighbors)):
            response = neighbors[x][-1]
            if response in classVotes:
                  classVotes[response] += 1
            else:
                  classVotes[response] = 1
      sortedVotes = sorted(classVotes.items(),
key=operator.itemgetter(1), reverse=True)
      return sortedVotes[0][0]



######################
# MEASURING ACCURACY #
######################

def getAccuracy(testSet, predictions):
      correct = 0
      for x in range(len(testSet)):
            if testSet[x][-1] in predictions[x]:
                  correct = correct + 1

      return (correct/float(len(testSet))*100)

def main():
      # prepare data
      trainingSet=[]
      testSet=[]
      split = 0.67
      loadDataset('iris.data', split, trainingSet, testSet)
      print ('Train set: ' + repr(len(trainingSet)))
      print ('Test set: ' + repr(len(testSet)))
      # generate predictions
      predictions=[]
      k = 3
      for x in range(len(testSet)):
            neighbors = getNeighbors(trainingSet, testSet[x], k)
            result = getResponse(neighbors)
            predictions.append(result)
            print('> predicted=' + repr(result) + ', actual=' +
repr(testSet[x][-1]))
      accuracy = getAccuracy(testSet, predictions)
      print('Accuracy: ' + repr(accuracy) + '%')

main()
```

Output:

```
Train set size: 100
Test set size: 49
> predicted='Iris-setosa', actual='Iris-setosa'
> predicted='Iris-setosa', actual='Iris-setosa'
> predicted='Iris-setosa', actual='Iris-setosa'
> predicted='Iris-versicolor', actual='Iris-versicolor'
> predicted='Iris-versicolor', actual='Iris-versicolor'
> predicted='Iris-versicolor', actual='Iris-versicolor'
> predicted='Iris-versicolor', actual='Iris-versicolor'
> predicted='Iris-virginica', actual='Iris-versicolor'
> predicted='Iris-versicolor', actual='Iris-versicolor'
> predicted='Iris-virginica', actual='Iris-versicolor'
> predicted='Iris-versicolor', actual='Iris-versicolor'
> predicted='Iris-virginica', actual='Iris-virginica'
> predicted='Iris-virginica', actual='Iris-virginica'
> predicted='Iris-virginica', actual='Iris-virginica'
> predicted='Iris-virginica', actual='Iris-virginica'
> predicted='Iris-virginica', actual='Iris-virginica'
.
.
.
Accuracy: 95.91836734693877%
```

Conclusion: We learnt about the K-Nearest Neighbours (KNN) algorithm and implemented it from scratch using python on the iris dataset.