Department of Computer Science and Engineering
(UG Studies)
PES University, Bangalore-560085

Session : Aug - Dec2017 **Credits :** 0-0-2-01	UE14CS405 : Machine Learning Lab
Lab # : 04	Implement a k- Near Neighbor finding and sorting algorithm for a data set size of (at least 1024) on 3 different metrics

Introduction K-Nearest Neighborhood Algoirthm

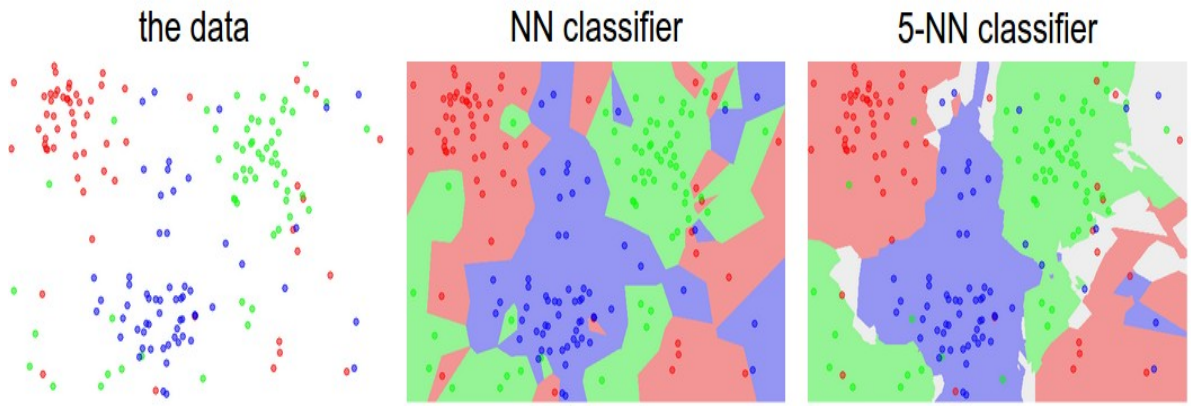
It is also called as lazy algorithm Which means that it does not use the training data points to do any generalization. In other words, there is no explicit training phase or it is very minimal. This means the training phase is pretty fast .In this case, we are given some data points for training and also a new unlabelled data for testing. Our aim is to find the class label for the new point. The algorithm has different behavior based on k.

Case 1 : k = 1 or Nearest Neighbor Rule

This is the simplest scenario. Let x be the point to be labeled . Find the point closest to x . Let it be y. Now nearest neighbor rule asks to assign the label of y to x. If the number of data points is very large, then there is a very high chance that label of x and y are same. Example: Assume there is a (potentially) biased coin. toss it for 1 million time and head has occured 900,000 times. Then most likely next one will be head. similar argument can be used. Consider a point x in the subspace which also has a lot of neighbors. Now let y be the nearest neighbor. If x and y are sufficiently close, then we can assume that probability that x and y belong to same class is fairly same – Then by decision theory, x and y have the same class.

Case 2 : k = K or k-Nearest Neighbor Rule

This is a straightforward extension of 1NN. Basically what we do is that we try to find the k nearest neighbor and do a majority voting. Typically k is odd when the number of classes is 2. Lets say k = 5 and there are 3 instances of C1 and 2 instances of C2. In this case , KNN says that new point has to labeled as C1 as it forms the majority. We follow a similar argument when there are multiple classes.



DataSet:

https://en.wikipedia.org/wiki/Iris_flower_data_set

https://osdn.net/projects/sfnet_irisdss/downloads/IRIS.csv/

Buggy code / Missing Function:

```
import csv
import random
import math
import operator

#Loading our dataset
def dataload(file,split,train=[],test=[]):
with open(file,'r') as csvfile:
lines=csv.reader(csvfile)
data=list(lines)
for x in range(len(data)-1):
for y in range(4):
data[x][y]=float(data[x][y])
if(random.random()<split):
train.append(data[x])
else:
test.append(data[x])

#Checking if this^ function works
trainingSet=[]
testSet=[]
dataload('iris.data', 0.66, trainingSet, testSet)
print('Train: ',len(trainingSet))
print('Test: ',len(testSet))

#Calculating Euclidean diastance
```

```
#Checking this function
x1 = [2, 2, 2]
x2 = [4, 4, 4]
distance = Edistance(x1, x2, 3)
print(distance)

#Returning K nearest neighbours based on distance
def Neighbors(train, test, k):
distances = []
length = len(test)-1
for x in range(len(train)):
dist = Edistance(test, train[x], length)
distances.append((train[x], dist))
distances.sort(key=operator.itemgetter(1))
neighbors = []
for x in range(k):
neighbors.append(distances[x][0])
return neighbors

#Checking this function
train = [[2, 2, 2,'a'], [4, 4, 4,'b']]
test = [5, 5, 5]
k = 1
print(Neighbors(train, test, 1))

#Voting the classes obtained from Neighbours
def Response(neighbors):
        Votes = {}
        for x in range(len(neighbors)):
                response = neighbors[x][-1]
                if response in Votes:
                        Votes[response] += 1
                else:
                        Votes[response] = 1
        sortedVotes = sorted(Votes.items(), key=operator.itemgetter(1), reverse=True)
        return sortedVotes[0][0]

#Checking response
neighbors = [[1,1,1,'a'], [2,2,2,'a'], [3,3,3,'b']]
response = Response(neighbors)
print(response)

#Calculating Accuracy
def Accuracy(testSet, predictions):
        correct = 0
        for x in range(len(testSet)):
                if testSet[x][-1] == predictions[x]:
                        correct += 1
        return (correct/float(len(testSet))) * 100.0
```

```
#Checking accuracy
testSet = [[1,1,1,'a'], [2,2,2,'a'], [3,3,3,'b']]
predictions = ['a', 'a', 'a']
accuracy = Accuracy(testSet, predictions)
print(accuracy)


#Combining all functions to get KNN

trainingSet=[]
test=[]
split = 0.67
dataload('iris.data', split, trainingSet, test)
print('Train set: ',len(trainingSet))
print('Test set: ',len(test))
predictions=[]
k = 1
for x in range(len(test)):
        neighbors = Neighbors(trainingSet, test[x], k)
        result = Response(neighbors)
        predictions.append(result)
        print('Predicted=',result,', Actual=',test[x][-1])
accuracy =Accuracy(test, predictions)
print(accuracy)
```

To Do for Students:

1. Identify the bug and fix the bug. Also find the accuracy
2.Modify the given code for various distance measures

a. Euclidean Distance
b. Minkowski Distance

3. Compare the accuracies for other distance measures

OUTCOME:

Given New Test Tuple Without Class Label The Algorithm Is Able To Classify To Which Class Label Tuple Belongs To I.E Either To Iris_Virginica,Iris_Versicolor,Iris_Setosa With Accuracy.