Ahsanullah University of Science and Technology

Department of Computer Science and Engineering

Program: Bachelor of Science in Computer Science and Engineering

Course No: CSE 4108

Course Title: Artificial Intelligence Lab

Assignment No: 04

Date of Submission: 12 / 02 / 2022

Submitted to:

Mr. Faisal Muhammad Shah
Associate Professor, Department of CSE, AUST.

Mr. Md. Siam Ansary
Lecturer, Department of CSE, AUST.

Submitted by,

Name: Nusrat Jahan

Student ID: 180104020

Question 1: Implement K Nearest Neighbor classifier in Python. For implementing the algorithms, Scikit-learn library cannot be used.

Solution:

Python Code:

```
import pandas as pd
import numpy as np
import operator

dataset = pd.read_csv("Customer.csv")
print(dataset)

def E_Distance(x1, x2, length):
    distance = 0

    for x in range(length):
        distance += np.square(x1[x] - x2[x])
    return np.sqrt(distance)



def knn(trainingSet, testInstance, k):
    distances = {}
    length = testInstance.shape[1]

    for x in range(len(trainingSet)):
```

```
        dist = E_Distance(testInstance, trainingSet.iloc[x], length)
        distances[x] = dist[0]
    sortdist = sorted(distances.items(), key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(sortdist[x][0])


    Count = {}
    for x in range(len(neighbors)):
        response = trainingSet.iloc[neighbors[x]][3]
        if response in Count:
            Count[response] += 1
        else:
            Count[response] = 1
    sortcount = sorted(Count.items(), key=operator.itemgetter(1),
reverse=True)
    print(sortcount)
    return (sortcount[0][0], neighbors)


testSet = [[54, 25, 3],[34, 45, 2],[23, 90, 3]]
test = pd.DataFrame(testSet)
k1 = 3
result1, neigh1 = knn(dataset, test, k1)
print(neigh1)
print(result1)
```

Offline4_1.py

```python
import pandas as pd
import numpy as np
import operator

dataset = pd.read_csv("Customer.csv")
print(dataset)


def E_Distance(x1, x2, length):
    distance = 0

    for x in range(length):
        distance += np.square(x1[x] - x2[x])
    return np.sqrt(distance)


def knn(trainingSet, testInstance, k):
    distances = {}
    length = testInstance.shape[1]

    for x in range(len(trainingSet)):
        dist = E_Distance(testInstance, trainingSet.iloc[x], length)
        distances[x] = dist[0]
    sortdist = sorted(distances.items(), key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(sortdist[x][0])


    Count = {}
    for x in range(len(neighbors)):
        response = trainingSet.iloc[neighbors[x]][3]
        if response in Count:
            Count[response] += 1
        else:
            Count[response] = 1
    sortcount = sorted(Count.items(), key=operator.itemgetter(1), reverse=True)
    print(sortcount)
    return (sortcount[0][0], neighbors)


testSet = [[54, 25, 3],[34, 45, 2],[23, 90, 3]]
test = pd.DataFrame(testSet)
k1 = 3
result1, neigh1 = knn(dataset, test, k1)
print(neigh1)
print(result1)
```

```
C:\Users\HP\AppData\Local\Microsoft\WindowsApps\python3.9.exe
Connected to pydev debugger (build 213.6461.77)
   Age  Income(K)  No_Of CC Class
0   35         50         3    No
1   22         40         2    No
2   34         45         3   Yes
3   23         78         2    No
4   43         90         3   Yes
5   35         12         2   Yes
6   54         25         3    No
7   34         45         2   Yes
8   23         90         3    No
[('Yes', 2), ('No', 1)]
[6, 5, 2]
Yes

Process finished with exit code 0
```

Question 2: Implement K Means Clustering algorithm in Python.For implementing the algorithms, Scikit-learn library cannot be used.

Python Code:

```
import numpy as np


X = np.array([
    [1, 4],
    [2, 3],
    [4, 2],
    [5, 3]]
)
```

```
colors = ["cluster1", "cluster2"]


class K_Means:
    def __init__(self, k=2, tol=0.001, max_iter=300):
        self.k = k
        self.tol = tol
        self.max_iter = max_iter


    def fit(self, data):


        self.centroids = {}


        for i in range(self.k):
            self.centroids[i] = data[i]


        for i in range(self.max_iter):
            self.classifications = {}


            for i in range(self.k):
                self.classifications[i] = []


            for featureset in data:
                distances = [np.linalg.norm(featureset -
self.centroids[centroid]) for centroid in self.centroids]
```

```
                classification = distances.index(min(distances))

self.classifications[classification].append(featureset)

            prev_centroids = dict(self.centroids)

            for classification in self.classifications:
                self.centroids[classification] =
np.average(self.classifications[classification], axis=0)

            optimized = True

            for c in self.centroids:
                original_centroid = prev_centroids[c]
                current_centroid = self.centroids[c]
                if np.sum((current_centroid - original_centroid) /
original_centroid * 100.0) > self.tol:
                    np.sum((current_centroid - original_centroid) /
original_centroid * 100.0)
                    optimized = False

            if optimized:
                break


    def predict(self, data):
```

```
        distances = [np.linalg.norm(data - self.centroids[centroid])
for centroid in self.centroids]
        classification = distances.index(min(distances))
        return classification



clf = K_Means()
clf.fit(X)


for classification in clf.classifications:
    color = colors[classification]
    for featureset in clf.classifications[classification]:
        print(featureset[0], featureset[1], color)


for centroid in clf.centroids:
    print(clf.centroids[centroid][0], clf.centroids[centroid][1])
```

Offline4_2.py

```python
import numpy as np

X = np.array([
    [1, 4],
    [2, 3],
    [4, 2],
    [5, 3]]
)

colors = ["cluster1", "cluster2"]

class K_Means:
    def __init__(self, k=2, tol=0.001, max_iter=300):
        self.k = k
        self.tol = tol
        self.max_iter = max_iter

    def fit(self, data):

        self.centroids = {}

        for i in range(self.k):
            self.centroids[i] = data[i]

        for i in range(self.max_iter):
            self.classifications = {}

            for i in range(self.k):
                self.classifications[i] = []

            for featureset in data:
                distances = [np.linalg.norm(featureset - self.centroids[centroid]) for centroid in self.centroids]
                classification = distances.index(min(distances))
                self.classifications[classification].append(featureset)

            prev_centroids = dict(self.centroids)

            for classification in self.classifications:
                self.centroids[classification] = np.average(self.classifications[classification], axis=0)

            optimized = True

            for c in self.centroids:
                original_centroid = prev_centroids[c]
                current_centroid = self.centroids[c]
                if np.sum((current_centroid - original_centroid) / original_centroid * 100.0) > self.tol:
                    np.sum((current_centroid - original_centroid) / original_centroid * 100.0)
                    optimized = False

            if optimized:
                break
```

```
59
60  clf = K_Means()
61  clf.fit(X)
62
63  for classification in clf.classifications:
64      color = colors[classification]
65      for featureset in clf.classifications[classification]:
66          print(featureset[0], featureset[1], color)
67
68  for centroid in clf.centroids:
69      print(clf.centroids[centroid][0], clf.centroids[centroid][1])
70
```

```
C:\Users\HP\AppData\Local\Microsoft\WindowsApps\python3.9.exe
Connected to pydev debugger (build 213.6461.77)
1 4 cluster1
2 3 cluster1
4 2 cluster2
5 3 cluster2
1.5 3.5
4.5 2.5

Process finished with exit code 0
```