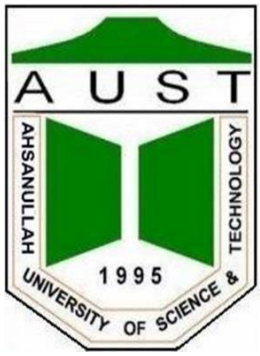
Ahsanullah University of Science and Technology



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: 03

Date of Submission: 08/08/2022

Submitted to:

Mr. Md. Siam Ansary
Lecturer, Department of CSE, AUST.
Ms. Tamanna Tabassum
Lecturer, Department of CSE, AUST.

Submitted by,

Name: S. M. Tasnimul Hasan

Student ID: 180204142

Question 1: Implement K Nearest Neighbor classifier in Python.

Solution:

Python Code:

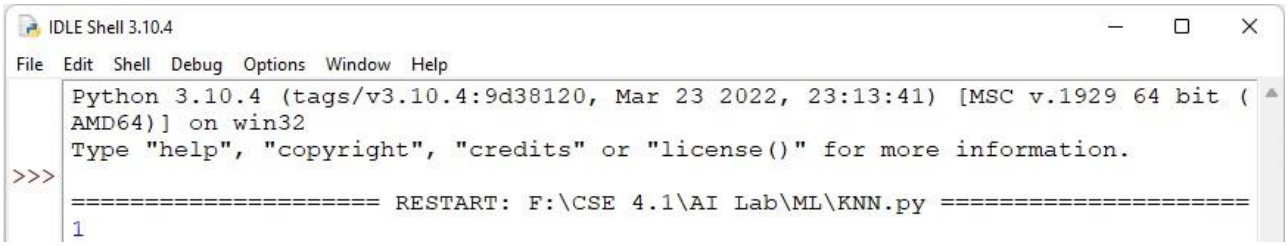
```
import math
from collections import Counter
import csv

def read_data():
file = open('Social_Network_Ads.csv')
data = csv.reader(file)
header = next(data)
x_train = []
y_train = []
for row in data:
# the index no 2 is Age and 3 is Salary which are two features I
used to predict
x_train.append([int(row[2]), float(row[3])])
# index 4 is class
y_train.append(int(row[4]))
file.close()
return x_train, y_train

def euclidian_distance(x1, y1, x2, y2):
return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

def argsort(x):
return sorted(range(len(x)), key=x.__getitem__)

class KNN:
def __init__(self, k=2):
self.Y_train = None
self.X_train = None
self.k = k
def fit(self, x, y):
self.X_train = x
self.Y_train = y
def predict(self, x):
# compute distance
distances = []
for feature in self.X_train:
x1 = feature[0]
y1 = feature[1]
x2 = x[0]
y2 = x[1]
distances.append(euclidian_distance(x1, y1, x2, y2))
# k nearest labels
k_indices = argsort(distances)[:self.k]
# find labels
k_nearest_labels = [self.Y_train[ind] for ind in k_indices]
most_common = Counter(k_nearest_labels).most_common(1)
return most_common[0][0]
```

```
x_train, y_train = read_data()
clf = KNN(k=3)
clf.fit(x_train, y_train)
test = [48, 29000]
print(clf.predict(test))
```

KNN.py - F:\CSE 4.1\AI Lab\ML\KNN.py (3.10.4)

File Edit Format Run Options Window Help

```
import math
from collections import Counter
import csv


def read_data():
    file = open('Social_Network_Ads.csv')
    data = csv.reader(file)

    header = next(data)

    x_train = []
    y_train = []

    for row in data:
        # the index no 2 is Age and 3 is Salary which are two features I used to predict
        x_train.append([int(row[2]), float(row[3])])
        # index 4 is class
        y_train.append(int(row[4]))

    file.close()
    return x_train, y_train


def euclidian_distance(x1, y1, x2, y2):
    return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)


def argsort(x):
    return sorted(range(len(x)), key=x.__getitem__)


class KNN:

    def __init__(self, k=2):
        self.Y_train = None
        self.X_train = None
        self.k = k

    def fit(self, x, y):
        self.X_train = x
        self.Y_train = y

    def predict(self, x):
        # compute distance
        distances = []
        for feature in self.X_train:
            x1 = feature[0]
            y1 = feature[1]
            x2 = x[0]
            y2 = x[1]
            distances.append(euclidian_distance(x1, y1, x2, y2))

        # k nearest labels
        k_indices = argsort(distances)[:self.k]

        # find labels
        k_nearest_labels = [self.Y_train[ind] for ind in k_indices]
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]


x_train, y_train = read_data()
clf = KNN(k=3)
clf.fit(x_train, y_train)
test = [48, 29000]
print(clf.predict(test))
```



Question 2: Implement K Means Clustering algorithm in Python.

Solution:

Python Code:

```
import csv
import random
import math
import matplotlib.pyplot as plt

def euclidian_distance(x1, y1, x2, y2):
return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

def calc_mean(data_points):
sumx = 0
sumy = 0
n = len(data_points)
for x in data_points:
sumx = sumx + x[0]
sumy = sumy + x[1]

return [sumx / n, sumy / n]

class Kmean:
def __init__(self, k=1, max_itr=1000):
self.k = k
self.centroid = None
self.max_itr = max_itr

def fit(self, x):
rand_ind = random.sample(range(0, len(x)), self.k)
self.centroid = [x[ind] for ind in rand_ind]

for i in range(0, self.max_itr):
clusters = self.assign_cluster(x)
prev_centroid = self.centroid
self.centroid = self.move_centroid(x, clusters)
if self.centroid == prev_centroid:
```

```
break
return clusters

def assign_cluster(self, x):
distance = []
clusters = []
for row in x:
for centroid in self.centroid:
x1 = row[0]
y1 = row[1]

x2 = centroid[0]
y2 = centroid[1]
distance.append(euclidian_distance(x1, y1, x2, y2))
min_distance = min(distance)
min_ind = distance.index(min_distance)
clusters.append(min_ind)
distance.clear()
return clusters

def move_centroid(self, x, clusters):
unique_clusters = list(set(clusters))
new_centroid = []
for cluster in unique_clusters:
temp = []
for ind in range(0, len(clusters)):
if cluster == clusters[ind]:
temp.append(x[ind])
new_centroid.append(calc_mean(temp))
return new_centroid

def read_csv():
file = open('custering.csv')
data = csv.reader(file)

header = next(data)
x_train = []
for row in data:

x_train.append([float(row[0]), float(row[1])])

file.close()
return x_train

x_train = read_csv()
kmean = Kmean(k=3)
y_mean = kmean.fit(x_train)
# print(y_mean)

t1 = []
t2 = []
t3 = []

for x in range(0, 2):
```

```
for ind in range(0, len(x_train)):
if y_mean[ind] == x:
if x == 0:
t1.append(x_train[ind])
elif x == 1:
t2.append(x_train[ind])
else:
t3.append(x_train[ind])

# plt.scatter([p[0] for p in t1], [p[1] for p in t1], color='green')
# plt.scatter([p[0] for p in t2], [p[1] for p in t2], color='red')
# plt.scatter([p[0] for p in t3], [p[1] for p in t3], color='blue')

# plt.show()
print("Cluster: 0")
print(t1)
print("Cluster: 1")
print(t2)
print("Cluster: 2");
print(t3)
```

k_means_cluster.py - F:\CSE 4.1\AI Lab\ML\k_means_cluster.py (3.10.4)

File Edit Format Run Options Window Help

```
import csv
import random
import math
import matplotlib.pyplot as plt


def euclidian_distance(x1, y1, x2, y2):
    return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)


def calc_mean(data_points):
    sumx = 0
    sumy = 0
    n = len(data_points)
    for x in data_points:
        sumx = sumx + x[0]
        sumy = sumy + x[1]
    return [sumx / n, sumy / n]


class Kmean:
    def __init__(self, k=1, max_itr=1000):
        self.k = k
        self.centroid = None
        self.max_itr = max_itr

    def fit(self, x):
        rand_ind = random.sample(range(0, len(x)), self.k)
        self.centroid = [x[ind] for ind in rand_ind]

        for i in range(0, self.max_itr):
            clusters = self.assign_cluster(x)
            prev_centroid = self.centroid
            self.centroid = self.move_centroid(x, clusters)
            if self.centroid == prev_centroid:
                break
        return clusters
```

```
    def assign_cluster(self, x):
        distance = []
        clusters = []
        for row in x:
            for centroid in self.centroid:
                x1 = row[0]
                y1 = row[1]
                x2 = centroid[0]
                y2 = centroid[1]
                distance.append(euclidian_distance(x1, y1, x2, y2))
            min_distance = min(distance)
            min_ind = distance.index(min_distance)
            clusters.append(min_ind)
            distance.clear()
        return clusters

    def move_centroid(self, x, clusters):
        unique_clusters = list(set(clusters))
        new_centroid = []
        for cluster in unique_clusters:
            temp = []
            for ind in range(0, len(clusters)):
                if cluster == clusters[ind]:
                    temp.append(x[ind])
            new_centroid.append(calc_mean(temp))
        return new_centroid


def read_csv():
    file = open('custering.csv')
    data = csv.reader(file)

    header = next(data)
    x_train = []
    for row in data:
        x_train.append([float(row[0]), float(row[1])])

    file.close()
    return x_train


x_train = read_csv()
kmean = Kmean(k=3)
y_mean = kmean.fit(x_train)
# print(y_mean)

t1 = []
t2 = []
t3 = []
```

```
for x in range(0, 2):
    for ind in range(0, len(x_train)):
        if y_mean[ind] == x:
            if x == 0:
                t1.append(x_train[ind])
            elif x == 1:
                t2.append(x_train[ind])
            else:
                t3.append(x_train[ind])

# plt.scatter([p[0] for p in t1], [p[1] for p in t1], color='green')
# plt.scatter([p[0] for p in t2], [p[1] for p in t2], color='red')
# plt.scatter([p[0] for p in t3], [p[1] for p in t3], color='blue')
# plt.show()
print("Cluster: 0")
print(t1)
print("Cluster: 1")
print(t2)
print("Cluster: 2");
print(t3)
```