COEN 240 MACHINE LEARNING

HOMEWROK TWO

NAME: BOSEN YANG **STUDENT ID: 1589880**

PROBLEM ONE

PROBLEM ONE

This is the cluster center update step.

Our goal is to derive $\vec{m}_k$ such that

$$\vec{m}_k = \arg\min_{\vec{m}_k^*} \sum_{n=1}^{N} \sum_{k=1}^{K} r_{kn} \left\| \vec{m}_k^* - \vec{x}_n \right\|_2^2$$

What we want is: $\frac{\partial J(\cdot)}{\partial \vec{m}_k}$

$$= \frac{\partial \sum_{n=1}^{N} r_{kn} (\vec{m}_k - \vec{x}_n)^T (\vec{m}_k - \vec{x}_n)}{\partial \vec{m}_k} \qquad \boxed{J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{kn} \left\| \vec{m}_k - \vec{x}_n \right\|_2^2}$$

$$= \frac{\partial}{\partial \vec{m}_k} \sum_{n=1}^{N} \left(\vec{m}_k^T \cdot \vec{m}_k - \vec{m}_k^T \cdot \vec{x}_n - \vec{x}_n^T \cdot \vec{m}_k + \vec{x}_n^T \cdot \vec{x}_n \right) \cdot r_{kn}$$

$$= \sum_{n=1}^{N} (2\vec{m}_k - 2\vec{x}_n) \cdot r_{kn} = \vec{0}$$

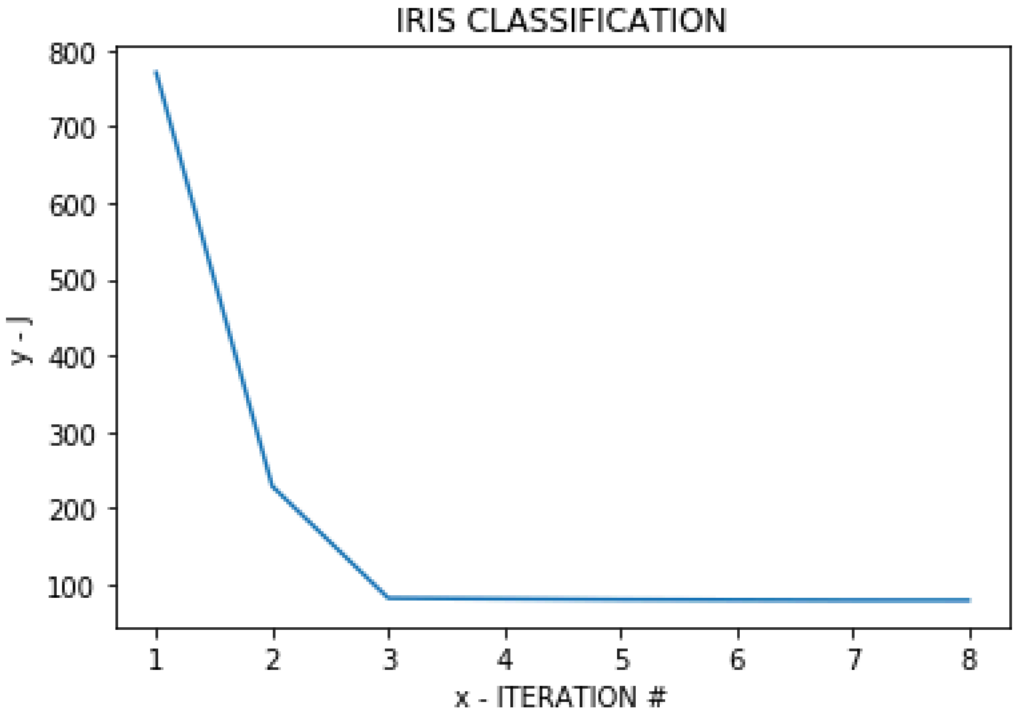
$$\Rightarrow \sum_{n=1}^{N} r_{kn} \cdot \vec{m}_k = \sum_{n=1}^{N} r_{kn} \cdot \vec{x}_n$$

$$\Rightarrow \vec{m}_k = \frac{\sum_{n=1}^{N} r_{kn} \cdot \vec{x}_n}{\sum_{n=1}^{N} r_{kn}}$$

The result tells us, if we view each data point as a vector, then the cluster center is the average vector of all points from that cluster.

PROBLEM TWO:

```
ITERATION #1    770.3261
ITERATION #2    228.4977
ITERATION #3    81.6952
ITERATION #4    80.8064
ITERATION #5    79.8736
ITERATION #6    79.3444
ITERATION #7    78.9213
ITERATION #8    78.8557
```

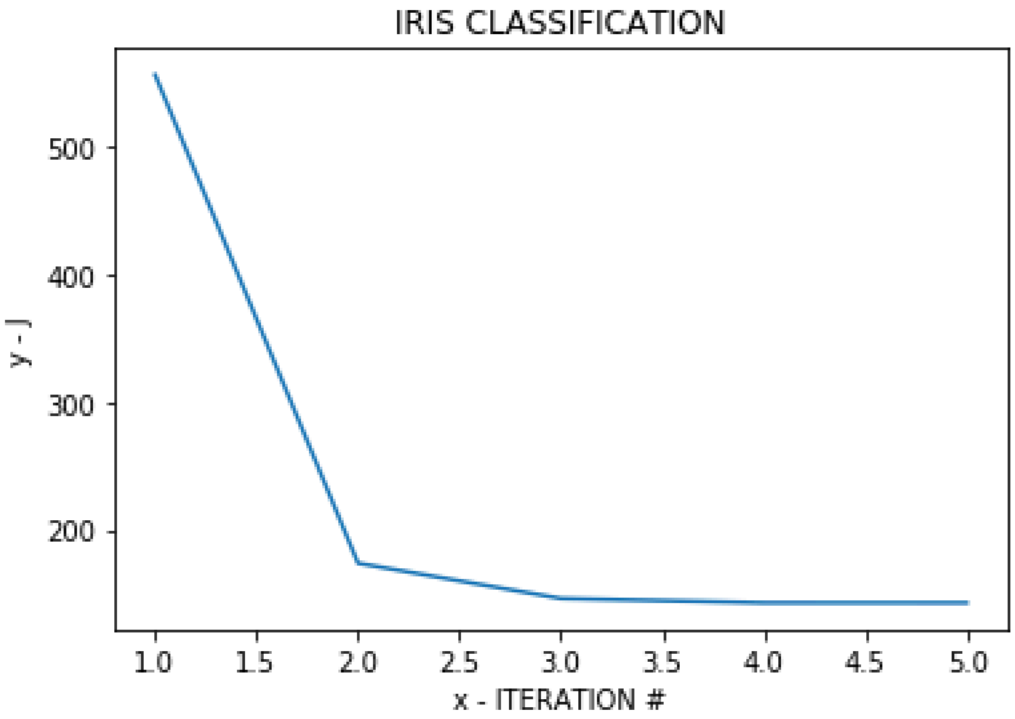


DONE

This is a typical result of our algorithm. After several repeats, it is found that the best our algorithm can do is to minimize the error function to some figure around 78.85. We can see our model converges really fast, normally taking 6 to 14 iterations, which means the model can quickly approximate the real cluster centers We can also find that for a majority of iteration, the algorithms is just adjusting the cluster center around its final position for better accuracy.

There is one more thing we should be aware of. The random initialization of cluster center can possibly lead to different convergence results. Sometimes we might not get result as good as the graph show above. For example, the following graphs are examples of convergence to a not so good local optimum.

```
ITERATION #1    556.7213
ITERATION #2    173.8028
ITERATION #3    146.1369
ITERATION #4    142.7846
ITERATION #5    142.7541
```



DONE

PROBLEM THREE & FOUR

PROBLEM THREE

a. We use y to denote the logistic regression output.

$y = \sigma(a) = \frac{1}{1+\exp(a)}$, where $a = \vec{w}^T \cdot \vec{x}_n$, and we have to know a bias term is already included in $\vec{w}^T$.

The criterion: if $y > 0.5$, classify it as C_1

otherwise, classify it as C_2

b. Only ONE weight, $\vec{w}$, needs to be calculated in this method.

PROBLEM FOUR

a. We also us y to denote the output

$y = \max_{C_k} P(C_k|\vec{x})$, where $P(C_k|\vec{x}) = \frac{P(C_k) \cdot P(\vec{x}|C_k)}{\sum_{j=1}^{K} P(C_j) \cdot P(\vec{x}|C_j)}$

The criterion is to choose the class with the highest probability.

b. K weights, the same number as total classes, need to be calculated in this method.

ATTACHMENTS

PROBLEM TWO CODE

```
import numpy as np
import pandas as pd
import random
import sys
import matplotlib.pyplot as plt

def initializeCenter(X, centers):
    attr_max = np.zeros((NUM_COLUMN, 1))
    attr_min = np.zeros((NUM_COLUMN, 1))
    for i in range(NUM_COLUMN):
        attr_val = X[:, i]
        attr_max[i] = np.amax(attr_val)
        attr_min[i] = np.amin(attr_val)
    for center in centers:
        for i in range(X.shape[1]):
            center[i][0] = random.uniform(attr_min[i], attr_max[i])

# X is a list of lists, where each component list represents a cluster
def assignCenter(X, centers):
    new_X = []
    for i in range(NUM_CENTER):
        new_X.append([])
    M = 0
    for cluster in X:
        for point in cluster:
            point = point.reshape(NUM_COLUMN,1)
            index, min_distance = findCorresCenter(point, centers)
            new_X[index].append(point)
            M = M + min_distance
    return new_X, M

def findCorresCenter(point, centers):
    index = 0
    min_distance = np.linalg.norm(point-centers[0])**2
    for i in range(len(centers)):
        temp = np.linalg.norm(point-centers[i])**2
        if (temp < min_distance):
            min_distance = temp
            index = i
    return index, min_distance
```

```
def calculateCenter(X, centers):
    new_centers = []
    for i in range(NUM_CENTER):
        cluster = X[i]
        cluster_center = np.zeros((NUM_COLUMN, 1))
        if (len(cluster) == 0):
            new_centers.append(centers[i])
            continue
        for i in range(NUM_COLUMN):
            cluster = np.array(cluster)
            cluster = cluster.reshape(cluster.shape[0], NUM_COLUMN)
            attr_val = cluster[:, i]
            cluster_center[i][0] = np.mean(attr_val)
        new_centers.append(cluster_center)
    return new_centers

# READ FROM ORIGINAL XLS FILE INTO NUMPY ARRAY
file_path_xls = "/Users/bosen/Library/Mobile Documents/com~apple~CloudDocs/Portal/COEN
240/Assignment/HW02/Iris.xls"
file_path_csv = "/Users/bosen/Library/Mobile Documents/com~apple~CloudDocs/Portal/COEN
240/Assignment/HW02/Iris.csv"
iris_xls = pd.read_excel(file_path_xls)
iris_xls.to_csv(file_path_csv, index = None, header=False)
iris_raw = np.genfromtxt(file_path_csv, delimiter=',')[:, 1:]
del(iris_xls)
NUM_ROW     = iris_raw.shape[0]
NUM_COLUMN  = iris_raw.shape[1]-1
X = iris_raw[:, :NUM_COLUMN]
t = iris_raw[:, NUM_COLUMN].reshape(NUM_ROW,1)

# DEFINE HYPERPARAMETERS AND INITIALIZE CENTERS
M = sys.float_info.max
Ms = []
EPSILON = 10**(-5)
NUM_CENTER = 3
NUM_ITERATION = 0
centers = []
for i in range(NUM_CENTER):
    center = np.zeros((NUM_COLUMN, 1))
    centers.append(center)
initializeCenter(X, centers)
X = np.array([X])
```

```
# ALTERNATES BETWEEN ASSIGNMENT AND CLUSTER-CENTER UPDATE
while(True):
   NUM_ITERATION = NUM_ITERATION+1
   X, new_M = assignCenter(X, centers) # THE FORM OF X
   if (M - new_M < EPSILON):
      break
   M = new_M
   Ms.append(M)
   print("ITERATION #%d\t%.4f" % (NUM_ITERATION, M))
   centers = calculateCenter(X, centers)

# PLOTTING
plt.plot(range(1, len(Ms)+1, 1), Ms)
plt.xlabel('x - ITERATION #')
plt.ylabel('y - J')
plt.title('IRIS CLASSIFICATION')
plt.show()

print("DONE")
```