LUMS
A Not-for-Profit University

Department of Electrical Engineering
School of Science and Engineering

EE212 Mathematical Foundations for Machine Learning and Data Science

ASSIGNMENT 4 – SOLUTIONS

Due Date: 23:55, Saturday. August 8, 2020 (Submit online on LMS)
Format: ?? problems, for a total of 100 marks
Instructions:

- You are not allowed to submit a group assignment. Each student must submit his/her own hand-written assignment, scanned in a single PDF document.
- You are allowed to collaborate with your peers but copying your colleague's solution is strictly prohibited. Anybody found guilty would be subjected to disciplinary action in accordance with the university rules and regulations.

Problem 1 (15 marks)

Alessandra designed an experiment where subjects tasted water from four different cups and attempted to identify which cup contained bottled water. Each subject was given three cups that contained regular tap water and one cup that contained bottled water (the order was randomized). She wanted to test if the subjects could do better than simply guessing when identifying the bottled water.
Her hypotheses were $H_0 : p = 0.25$ vs $H_a : p > 0.25H$ (where p is the true likelihood of these subjects identifying the bottled water).
The experiment showed that 20 of the 60 subjects correctly identified the bottle water. Alessandra calculated that the statistic $\hat{p} = \dfrac{20}{60} = \dfrac{1}{3}$ had an associated P-value of approximately 0.068.

(a) [**5 marks**] What conclusion should be made using a significance level of $\alpha = 0.05$?
   i. Fail to reject H_0
   ii. Reject H_0 and accept H_a
   iii. Accept H_0

(b) [**5 marks**] What does this conclusion imply in this context?

(c) [**5 marks**] How would the conclusion have changed if Alessandra had instead used a significance level of $\alpha = 0.10$?

Solution:

(a) Fail to reject H_0. Since the p-value of 0.068 is greater than $\alpha = 0.05$, we should fail to reject H_0.

$$\text{P-value} < \alpha \Rightarrow \text{reject } H_0 \Rightarrow \text{accept } H_\text{a}$$
$$\text{P-value} \geq \alpha \Rightarrow \text{fail to reject } H_0$$

Since the P-value of 0.068 is larger than $\alpha = 0.05$, sample results as high or higher than $\hat{p} = \frac{20}{60}$ can plausibly happen by random chance alone when H_0 is true. In other words, if all 60 subjects were just guessing, there's about a 6.8% chance that 20 or more of them would correctly identify the bottled water. This random chance probability exceeds the significance level $\alpha = 0.05$ so the results aren't unusual enough for us to reject H_0.

(b) We don't have enough evidence to say that these subjects can do better than guessing when identifying the bottled water. The null hypothesis $H_0 : p = 0.25$ says their likelihood is no better than guessing, and we failed to reject the null hypothesis.

(c) She would have rejected H_0 and accepted H_a Changing the significance level would not change the results of the experiment or the P-value. Since 0.068 is less than $\alpha = 0.10$ this significance level would have led Alessandra to reject H_0 and accept H_a.

Problem 2 (15 marks)

A test for a certain rare disease has 90% accuracy: if a person has the disease, the test results are positive with probability 0.9, and if the person does not have the disease, the test results are negative with probability 0.9. A random person drawn from a certain population has probability 0.001 of having the disease. Given that the person just tested positive, what is the probability of having the disease?

Solution: Let A be the event that the person has the disease. Let B be the event that the test resukts are positive. The desired probability, $\mathbf{P}(A|B)$, is found by Bayes' rule:

$$\mathbf{P}(A|B) = \frac{\mathbf{P}(A)\mathbf{P}(B|A)}{\mathbf{P}(A)\mathbf{P}(B|A) + \mathbf{P}(A^c)P(B|A^c)} = \frac{0.001 \cdot 0.90}{0.001 \cdot 0.90 + 0.999 \cdot 0.10} = 0.0089$$

Note that even though the test was assumed to be fairly accurate, a person who has tested positive is still very unlikely to have the disease.

Problem 3 (15 marks)

Manufactured items have a strength that has a normal distribution with a standard deviation of 4.2. The mean strength can be altered by the operator. At what value should the mean strength be set so that exactly 95% of the items have a strength less than 100?

Solution: From the question statement, we have standard deviation $\sigma = 4.2$. Let X be the strength of individual items. Then:

$$P(X \leq 100) = 0.95$$
$$P(Z \leq \frac{100 - \mu}{4.2}) = 0.95$$

The z value corresponding to the CDF of 0.95 is 1.645. Therefore, solving:

$$\frac{100 - \mu}{4.2} = 1.645$$
$$\mu = 93.09$$

Hence, the mean strength should be set at 93.09.

Problem 4 (15 marks)

The lifetime of particles inside a chemical reactor can be modelled as an exponential random variable. In a chemical reactor, 10% of the particles stay inside the vessel no longer than about 0.42 minutes. What percentage of the particles stay longer than 2 minutes inside the vessel?

Solution: Let X be the lifetime of the particles; X can be modelled as an exponential random variable.

$$\begin{aligned} P(X < 0.42) &= 0.10 \\ 1 - e^{-0.42\lambda} &= 0.10 \\ \lambda &= 0.25 \\ P(X > 2) &= e^{-2\lambda} \\ &= e^{-0.50} \\ &= 0.60 \end{aligned}$$

Hence around 60% of the particles stay longer than 2 minutes inside the vessel.

Problem 5 (20 marks)

K-means Clustering Algorithm

Another application of unsupervised learning is K-means clustering. The algorithm operates on data clusters. Let's say we have a data which can be divided into 5 classes. This means that a feature of data that categorize the class can be plotted on a 2D plane and it will form 5 clusters. Each cluster belonging to 1 class. Sometimes the clusters are easily separable and far apart, sometimes they are jumbled up.

Now, we have unlabeled data available, so we don't know which data point belongs to which class. What this algorithm does is take the unlabeled data and tries to separate it into K different clusters. This way we can classify which data point belongs to which cluster.The algorithm has the following steps:

1. Randomly initialize K centroids. These are data points that will indicate the centre of each cluster. Think of them naively as a label of sorts.
2. Take the euclidean distance of each data point from each centroid. For each data point, the centroid which is closest to the it, is assigned to it. Hence after this step, every data point has a centroid assigned to it.
3. For each centroid, find the mean of the data points attached to it and shift the centroid to that mean.
4. Repeat part 2 and 3 until the centroids eventually stop shifting in every iteration and converge.

Implementation

We will now implement this algorithm using Python. Submit your files along with your assignment PDF solution on LMS. We are going to use the IRIS data set which has data that corresponds to a class of iris plant.We wish to separate the data of each class.

(a) Load the following libraries and load and visualize the iris data set using the following commands:

```
from sklearn import datasets
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans

# Load data
iris = datasets.load_iris()

# Define data and labels
X = iris.data[:, :2]
y = iris.target


# Visualize data
plt.scatter(X[:,0], X[:,1], c=y, cmap='gist_rainbow')
plt.xlabel('Spea1 Length', fontsize=18)
plt.ylabel('Sepal Width', fontsize=18)
plt.show()
```

This should show you the data clusters and the original classification of the data into 3 different classes.

(b) Perform K-means algorithm on this data set using the following commands:

```
kmeans = KMeans(init="random", n_clusters=3, n_init=a,
                max_iter=b, random_state=c)
kmeans.fit(X)
y_test=kmeans.labels_
```

Don't run this code yet because this requires values in arguments and not a,b and c.

(c) Find the lowest error this classifier gives and also plot it to see how you classifier performed, using the following commands:

```
 # Lowest error value
print("Lowest error value:", kmeans.inertia_)
print("Centroid location:", kmeans.cluster_centers_)
print("Iterations till convergence:", kmeans.n_iter_)


# Visualize
# Plot the identified clusters and compare with the answers
fig, axes = plt.subplots(1, 2, figsize=(16,8))
axes[0].scatter(X[:, 0], X[:, 1], c=y, cmap='gist_rainbow',
edgecolor='k', s=150)
axes[1].scatter(X[:, 0], X[:, 1], c=y_test, cmap='jet',
edgecolor='k', s=150)
axes[0].set_xlabel('Sepal length', fontsize=18)
axes[0].set_ylabel('Sepal width', fontsize=18)
axes[1].set_xlabel('Sepal length', fontsize=18)
axes[1].set_ylabel('Sepal width', fontsize=18)
axes[0].tick_params(direction='in', length=10, width=5, colors='k',
```

```
                    labelsize=20)
axes[1].tick_params(direction='in', length=10, width=5, colors='k',
                    labelsize=20)
axes[0].set_title('Actual', fontsize=18)
axes[1].set_title('Predicted', fontsize=18)
plt.show()
```

Parameters

init indicates that we have randomly initialized the centroid location.

n_clusters indicates that in how many classes do we wish to separate the clusters into. This means we need to know the number of classes the data has. If we don't know then there are separate methods of figuring that out, The Elbow Method is one example, if you are curious. For our case, we will assume we know the number of classes, which is 3.

n_init refers to how many different random initializations we wish to try out. How you initialize the centroid can have a huge affect on where they finally converge. Since initializations are random, we wish to try different ones to see which ones gives us the lowest error. We can't try too many since that increases the computation cost.

max_iter is the maximum number of iterations you wish to run to ensure your algorithm results in the centroid converging. Again, if it's too small then your algorithm won't converge and if it's too large then it's computationally heavy.

random_state corresponds how you randomly initialize the centroid.

Task

Tweak the values of a, b and c to see which combination gives you minimum error and takes as few iterations to converge as possible. For c, try different values between 1 and 42.

Problem 6 (20 marks)

The following code uses some pre-determined weights in a two-input and one-output perceptron for separating points on a graph. The dataset variable contains four inputs, each of which has three entries. The first two correspond to the input while the third is the output. Run the code to verify that the results using these weights generate incorrect outputs or predictions for each input. The code then trains the perceptron in order to minimize the error against the current output, computing new weights. Using these new weights, an updated prediction is made about the given inputs. Modify the code below by re-configuring initial weights, learning rate and/or number of iterations/epochs. Briefly comment why the perceptron using the current parameters is unable to correctly classify the given inputs.

```
import matplotlib.pyplot as plt

def predict(row, weights):
activation = weights[0]
for i in range(len(row)-1):
 activation += weights[i + 1] * row[i]
return 1.0 if activation >= 0.0 else 0.0
```

```
def train_weights(train, l_rate, n_epoch,old_weights):
weights = old_weights
for epoch in range(n_epoch):
sum_error = 0.0
for row in train:
 prediction = predict(row, weights)
 error = row[-1] - prediction
 sum_error += error**2
weights[0] = weights[0] + l_rate * error
 for i in range(len(row)-1):
 weights[i + 1] = weights[i + 1] + l_rate *     error * row[i]
print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
return weights

dataset = [[0,0,0],[0,1,1],[1,0,0],[1,1,0]]

print("[info] plotting the dataset...")
print("[info] 0 -> Red, 1 -> Blue...")
for k in dataset:
 if k[2] == 0:
 plt.scatter(k[0], k[1], c='r', label='data')
 else:
 plt.scatter(k[0], k[1], c='b', label='data')


print("[info] setting predetermined weights...")
weights = [-1, 1, 1]
print("[info] weights: ",weights)
print("[info] making a prediction...")
for row in dataset:
 prediction = predict(row, weights)
 print("[data] Expected=%d, Predicted=%d" % (row[-1], prediction))

print("[info] trainging weights...")
l_rate = 0.1
n_epoch = 4
weights = train_weights(dataset, l_rate, n_epoch,weights)
print("[info] weights: ",weights)

print("[info] making a prediction...")
for row in dataset:
 prediction = predict(row, weights)
 print("[data] Expected=%d, Predicted=%d" % (row[-1], prediction))

plt.show()
```

Solution: Increased learning rate (0.4) and higher number of epochs (8) allow for considerable iterations to update the weights and thus define the line appropriately for classification.

— End of Assignment —