



UNIVERSITY OF TEHRAN

COLLEGE OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

INTELLIGENT SYSTEM

ASSIGNMENT#4

MOHAMMAD HEYDARI

810197494

UNDER SUPERVISION OF:

DR. RESHAD HOSSINI

ASSISTANT PROFESSOR

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

UNIVERSITY OF TEHRAN

Dec. 2021

1 CONTENTS

| | | |
|----------|--|-----------|
| 2 | Question #1 | 3 |
| 2.1 | K-Means Clustering(Theoretical) | 3 |
| 2.2 | Hierarchical Clustering(Theoretical) | 7 |
| | Creating a Proximity Matrix: | 8 |
| 3 | Question #2 | 10 |
| 3.1 | Implementing K-Means Algorithm (No Python Libraries) | 10 |
| 3.2 | Effect of Repetition in experiments | 15 |
| 4 | Question #3 | 17 |
| 4.1 | Logistic Regression..... | 17 |
| 4.2 | Pre Model Evaluation | 21 |
| 4.3 | Semi Supervised Learning (Self Training Approach)..... | 22 |
| 4.4 | Changing Threshold Value and Evaluate Model | 26 |
| 5 | Question #4 | 28 |
| 5.1 | Probability (Theoretical Based) | 28 |
| 5.1.1 | Un-Fair Coin Problem..... | 28 |
| 5.1.2 | Evaluation of Reaching Time in Bus & Metro Problem | 29 |
| 5.1.3 | Employee of Programming Company..... | 31 |
| 5.2 | Probability (Simulation Based)..... | 33 |
| 5.2.1 | Birthday Problem | 33 |
| 5.2.2 | Central Limit Theorem..... | 35 |
| 6 | Acknowledgement | 40 |
| 7 | References | 41 |

2 QUESTION #1

2.1 K-MEANS CLUSTERING(THEORETICAL)

In this part we intend to solve a simple example which cover K-Means Algorithm usage in Clustering.

According to question description I have only investigated this problem in a theoretical-based approach.

First of all, let's have a look on K-Means algorithm:

There are several steps must be issued when we are struggling with these type of problems:

Step 1:

Randomly initialize the cluster centers of each cluster from the data points.

Step 2:

For each data point, compute the Euclidian-Distance from all the centroids and assign the cluster based on the minimal distance to all the centroids.

Step 3:

Adjust the centroid of each cluster by taking the average of all the data points which belong to that cluster on the basis of the computations performed in step 2.

Step 4:

Repeat this process till clusters are well separated or convergence is achieved!

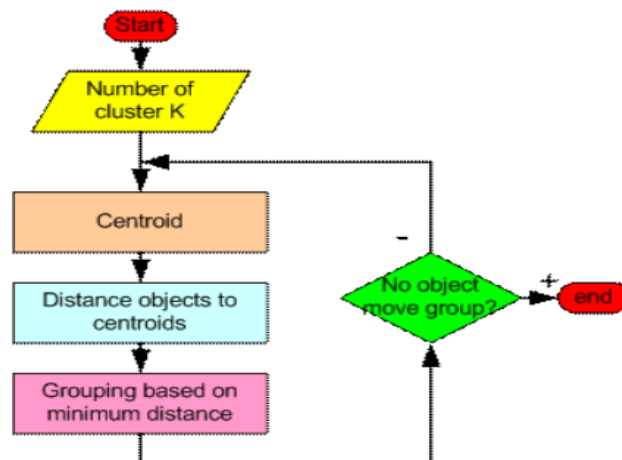
Now, let's dive into our problem.

Not that in this problem we have only **two** clusters so $K=2$.

| i | x_1 | x_2 |
|-----|-------|-------|
| A | 1 | 1 |
| B | 1 | 0 |
| C | 0 | 2 |
| D | 2 | 4 |
| E | 3 | 5 |

Table 1.1: Representation of Dataset

Please follow the steps based on below flowchart:



1. Initial value of Centroids:

We have five points and each point have two attribute or features as shown in **Table 1.1**. our goal is to group these points in to $K=2$ group of point based on the two features.

Each point represents one point with two attributes (X, Y) that we can represent it as coordinate in an attribute space as shown in the figure below:

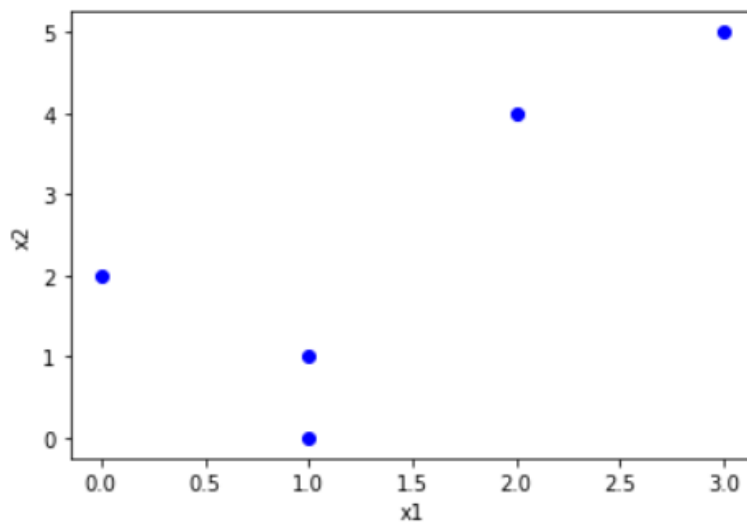


Figure 2: Representation of Data

1. initial value of centroids: Suppose we use point A and D as the first centroids. Let c_1, c_2 Denote the coordinate of the centroids, then $c_1 = (1,1)$ and $c_2 = (2,4)$

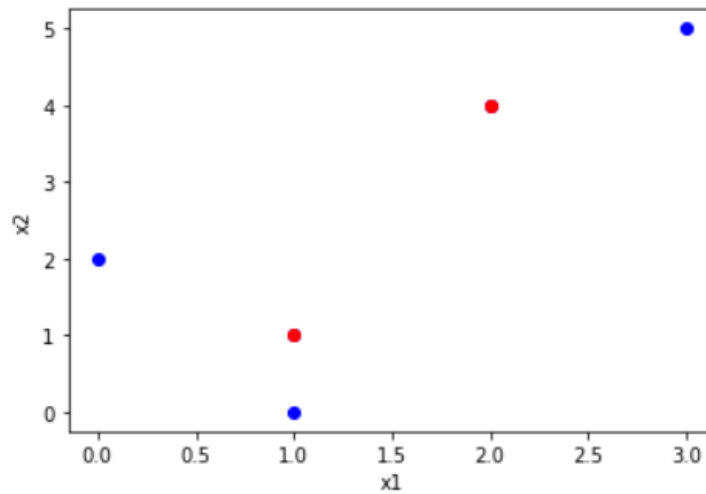


Figure 2: Representation of Data & initial Centroids

2. Object-Centroids distance: we calculate the distance between cluster centroid to each object. Let use Euclidean distance, then we have calculated distance matrix at iteration 0 is:

$$D^0 = \begin{bmatrix} 0 & 1 & 1.41 & 3.16 & 4.47 & c_1(1,1) \\ 3.16 & 4.12 & 2.83 & 0 & 1.41 & c_2(2,4) \end{bmatrix}$$

$$A = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, C = \begin{pmatrix} 0 \\ 2 \end{pmatrix}, D = \begin{pmatrix} 2 \\ 4 \end{pmatrix}, E = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

3. Object clustering: we assign each object based on the minimum distance. Thus, point A is assigned to group 1, point B to group 1, point C to group 1, point D to group 2 and finally point E to group 2. The element of group matrix is 1 if and only if the object is assigned to that group.

$$C^0 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & c_1(1,1) \\ 0 & 0 & 0 & 1 & 1 & c_2(2,4) \end{bmatrix}$$

$$A = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, C = \begin{pmatrix} 0 \\ 2 \end{pmatrix}, D = \begin{pmatrix} 2 \\ 4 \end{pmatrix}, E = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

4. iteration-1, determine centroids: Knowing the members of each group, now we compute the new centroid of each group based on these new memberships. Group 1 has three member thus the centroid obtain in this way so $c_1 = \left(\frac{1+1}{3}, \frac{1+0+2}{3}\right) = (0.67, 1)$. Group 2 now has two members; thus the centroid is the average coordinate among the two members: $c_2 = \left(\frac{2+3}{2}, \frac{4+5}{2}\right) = (2.5, 4.5)$.

5. iteration-1, Object-Centroids distances: The next step is to compute the distance of all objects to the new centroids. Similar to step 2, we have distance matrix at iteration 1 is:

$$D^1 = \begin{bmatrix} 0.33 & 1.05 & 1.21 & 3.28 & 4.63 & c_1(1,1) \\ 3.81 & 4.74 & 3.54 & 0.71 & 0.71 & c_2(2,4) \end{bmatrix}$$

$$A = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, C = \begin{pmatrix} 0 \\ 2 \end{pmatrix}, D = \begin{pmatrix} 2 \\ 4 \end{pmatrix}, E = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

6. iteration-1, Object clustering: similar to step 3, we assign each object based on minimum distance. Based on the new distance matrix, we move the point A to group 1, we move the point B to group 1, we move the point C to group 1, we move the point D to group 2, we move the point E to group 2 and as we can see all of the point have the same category!

$$C^1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & c_1(1,1) \\ 0 & 0 & 0 & 1 & 1 & c_2(2,4) \end{bmatrix}$$

$$A = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, C = \begin{pmatrix} 0 \\ 2 \end{pmatrix}, D = \begin{pmatrix} 2 \\ 4 \end{pmatrix}, E = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

Final: we obtain the result $C^1 = C^0$. Comparing the grouping of last iteration and this iteration reveals that the objects doesn't move group anymore. Thus, the computation of the k-mean clustering has reached stability and no more iteration is needed. We get the final grouping as the results.

So we have finally below figure for distribution of our clustering data:

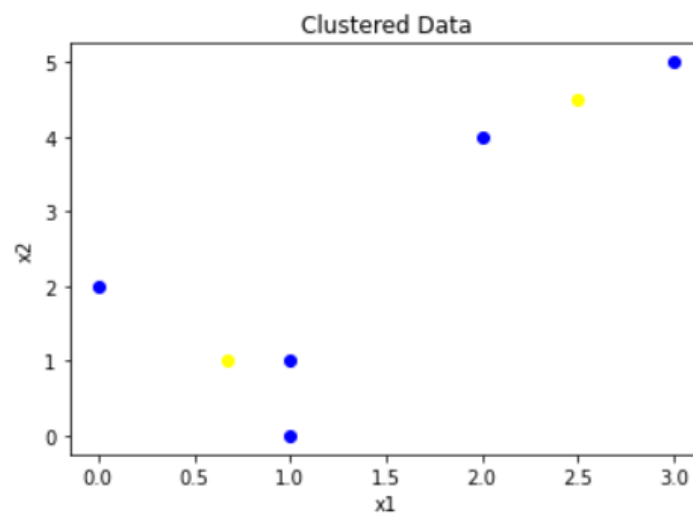


Figure 3: Representation of Clustered Data

2.2 HIERARCHICAL CLUSTERING(THEORETICAL)

In this part we intend to analyse Hierarchical clustering through a simple Theoretical-based example.

Note that in this problem we have used Single-Link method which let us to pick up the minimum score during merging different cells!

Furthermore, we have used L2-Norm for clustering the data and in other word most important metrics in this process related to L2-Norm during finding the distance matrix.

Algorithm:

At the glance:

First, we assign each of our points to a separate cluster, Then, based on the similarity of these clusters, we can combine the most similar clusters together and repeat this process until only a single cluster is left!

We are essentially building a hierarchy of clusters. That's why this algorithm is called hierarchical clustering.

In Practical cases:

We merge the most similar points or clusters in hierarchical clustering.

Similarity metric: Take the distance between the centroids of these clusters. The points having the least distance are referred to as similar points and we can merge them. We can refer to this as a distance-based algorithm as well (since we are calculating the distances between the clusters).

In hierarchical clustering, we have a concept called a proximity matrix. This stores the distances between each point. Let's dive into our problem to understand this matrix as well as the steps to perform hierarchical clustering.

| i | x | y |
|-------|------|------|
| P_1 | 0.22 | 0.38 |
| P_2 | 0.35 | 0.32 |
| P_3 | 0.26 | 0.19 |
| P_4 | 0.08 | 0.41 |
| P_5 | 0.45 | 0.30 |

Table 3.2: Representation of Dataset

Creating a Proximity Matrix:

First, we will create a proximity matrix which will tell us the distance between each of these points. Since we are calculating the distance of each point from each of the other points, we will get a square matrix of shape $n \times n$ (where n is the number of observations).

Let's make the 5×5 proximity matrix for our example:

According to **Table 4.2** we will have following cases:

Iteration #1:

$$\begin{bmatrix} i & p_1 & p_2 & p_3 & p_4 & p_5 \\ p_1 & 0 & 0.1432 & 0.19 & 0.1432 & 0.24 \\ p_2 & 0.1432 & 0 & 0.16 & 0.28 & \mathbf{0.1} \\ p_3 & 0.19 & 0.16 & 0 & 0.28 & 0.22 \\ p_4 & 0.14 & 0.28 & 0.28 & 0 & 0.39 \\ p_5 & 0.24 & \mathbf{0.1} & 0.22 & 0.39 & 0 \end{bmatrix} \quad \left\{ \begin{array}{l} \text{Single Link(minimum)} \\ |p_1| = |(0.22,0.38)| = 0.44 \\ |p_2| = |(0.35,0.32)| = 0.47 \\ |p_3| = |(0.26,0.19)| = 0.32 \\ |p_4| = |(0.08,0.41)| = 0.42 \\ |p_5| = |(0.45,0.30)| = 0.54 \end{array} \right.$$

Next, we will look at the smallest distance in the proximity matrix and merge the points with the smallest distance.

So I have merged P2 & P5 at the first step.

We then update the proximity matrix:

$$\begin{bmatrix} i & p_1 & p_2, p_5 & p_3 & p_4 \\ p_1 & 0 & \mathbf{0.1432} & 0.19 & \mathbf{0.1432} \\ p_2, p_5 & \mathbf{0.1432} & 0 & 0.16 & 0.28 \\ p_3 & 0.19 & 0.16 & 0 & 0.28 \\ p_4 & \mathbf{0.1432} & 0.28 & 0.28 & 0 \end{bmatrix} \quad \left\{ \begin{array}{l} \text{Single Link(minimum)} \\ |p_1| = |(0.22,0.38)| = 0.44 \\ |p_2, p_5| = |(0.35,0.32)| = 0.47 \\ |p_3| = |(0.26,0.19)| = 0.32 \\ |p_4| = |(0.08,0.41)| = 0.42 \end{array} \right.$$

Next, we will look at the smallest distance in the proximity matrix and merge the points with the smallest distance.

So I have merged P2 & P5 with P1 or we can merge P2 & P5 with P4 or P1 & P4 at the second step but it doesn't make a matter so we choose P1 with P4 and Then merge them together at the second step!

We then update the proximity matrix:

$$\begin{bmatrix} i & p_2, p_5 & p_3 & p_1, p_4 \\ p_2, p_5 & 0 & \mathbf{0.16} & 0.28 \\ p_3 & \mathbf{0.16} & 0 & 0.28 \\ p_1, p_4 & 0.28 & 0.28 & 0 \end{bmatrix} \quad \left\{ \begin{array}{l} \text{Single Link(minimum)} \\ |p_2, p_5| = |(0.35,0.32)| = 0.47 \\ |p_3| = |(0.26,0.19)| = 0.32 \\ |p_1, p_4| = |(0.08,0.41)| = 0.42 \end{array} \right.$$

Next, we will look at the smallest distance in the proximity matrix and merge the points with the smallest distance.

So I have merged P3 with P5 & P2 at the third step!

We then update the proximity matrix:

$$\begin{bmatrix} i & p_2, p_3, p_5 & p_1, p_4 \\ p_2, p_3, p_5 & 0 & \mathbf{0.28} \\ p_1, p_4 & \mathbf{0.28} & 0 \end{bmatrix} \quad \left\{ \begin{array}{l} \text{Single Link (minimum)} \\ |p_2, p_3, p_5| = |(0.26, 0.19)| = 0.32 \\ |p_1, p_4| = |(0.08, 0.41)| = 0.42 \end{array} \right.$$

Next, we will look at the smallest distance in the proximity matrix and merge the points with the smallest distance.

So I have merged P2 & P3 & P5 with P1 & P4 at the forth step!

And That's it all step have been done successfully!

And at the end we face with a single cell as our single cluster.

Further you can see the Dendrogram

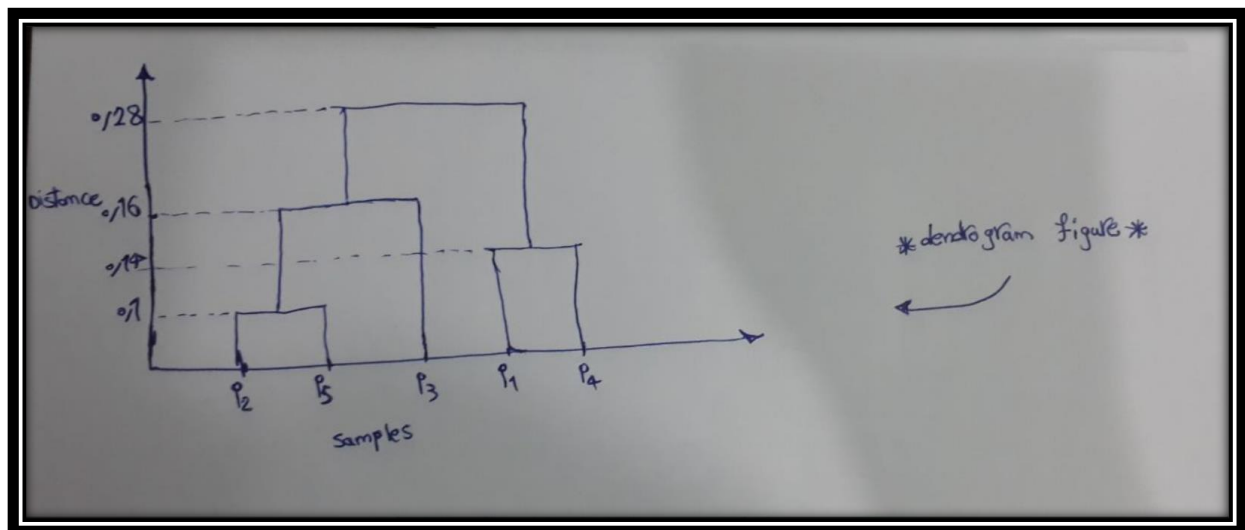


Figure 4: Representation of Clustered Data

If there is any issue with that my solution please let me know to provide more detailed information. (be in touch [here!](#))

3 QUESTION #2

3.1 IMPLEMENTING K-MEANS ALGORITHM (NO PYTHON LIBRARIES)

In this part we intend to implement K-Means Algorithm from scratch in python!

First of all, As I mentioned totally in Question #1 in this part I want to discuss about the algorithm and steps!

Hopefully K-Means algorithm was so straight forward and I have not any issues with its implementation!

So let's check the following steps:

Step 1:

Randomly initialize the cluster centers of each cluster from the data points.

Step 2:

For each data point, compute the Euclidian-Distance from all the centroids and assign the cluster based on the minimal distance to all the centroids.

Step 3:

Adjust the centroid of each cluster by taking the average of all the data points which belong to that cluster on the basis of the computations performed in step 2.

Step 4:

Repeat this process till clusters are well separated or convergence is achieved!

Describing my implementation:

Note that in this implementation I have written a function called `Kmeans_algorithm(K,X,iteration)`

It takes number of cluster and also number of iteration which is 150 in our case and also it takes our input data which is a data-frame type in whole of process in my case because I feel working with data-frame is very nice and comfort and it gives me a better insight during debug and also during the interpreting the results and so on!

In this function I have used from two for loop to calculate the Euclidean distance between every single centroids and every other 4-dimension x data!

The beautiful syntax which I have never used in python is `for in something.iterrows()` which pass over the rows and also gives us the index which highly helped me to code pretty much better and in an easier method and really loved that notation!

Also I have calculated the distance over 4 dimension and afterward I have summed four values and going through this way I have obtained the 4-dimension distance among two 4D list!

I have used a list called ED () which collected the Distance values during dealing with for loops and store data and in other word implemented the matrix which I have discussed and used in previous section

Also a considerable note about my code is that I have used the X data-frame during all of my code and obviously it changes in every stage of my code so at the first of loop I have saved the x vector in a separate vector called X-saver in my notation!

After these type of steps we are going to cluster data point and update the centroids for which iteration, I this part I have passed over X values and stored the correspond distance value in same rows with their X data.

And finally I have passed over the X and check according to distance value whether point is belonging to a specific cluster or not and after the end of loop I have grouped X by Cluster feature which declares in my data-frame and update centroids using. mean ()

Further you can see whole of my implementation in python:

K-Means Algorithm

```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
from operator import truediv
from sklearn import datasets
pd.options.mode.chained_assignment = None

iris= datasets.load_iris()
X=iris.data
y= iris.target
df = pd.DataFrame(data=iris.data, columns=["sepal_length", "sepal_width", "petal_length", "petal_width"])
df["class"] = iris.target

def data_cluster(K,X_saver,C):
    X_saver["Cluster"]=C
    df_new = X_saver[X_saver["Cluster"]==K]
    data_c=df_new[["sepal_length", "sepal_width", "petal_length", "petal_width"]]
    return data_c

def Distance_cluster(X_saver):
    cost=[]
    cost_data=[]
    for idx1,row_1 in X_saver.iterrows():
        ED=[]
        for idx2,row_2 in X_saver.iterrows():
            d1=(row_1["sepal_length"]-row_2["sepal_length"])**2
            d2=(row_1["sepal_width"]-row_2["sepal_width"])**2
            d3=(row_1["petal_length"]-row_2["petal_length"])**2
            d4=(row_1["petal_width"]-row_2["petal_width"])**2
            d=np.sqrt(d1+d2+d3+d4)
            ED.append(d)
        cost_data.append(sum(ED))
    cost=0.5*sum(cost_data)
    return cost

def Kmeans_algorithm(K,X,iteration):
    cluster_Number=K
    Centroids = (X.sample(n=cluster_Number))
    Centroids
    final_cost=[]
    for i in range(iteration):
        X_saver=X
        cnt=1
        for idx1,row_Centroids in Centroids.iterrows():
            ED=[]
            for idx2,row_data in X_saver.iterrows():
                d1=(row_Centroids["sepal_length"]-row_data["sepal_length"])**2
                d2=(row_Centroids["sepal_width"]-row_data["sepal_width"])**2
                d3=(row_Centroids["petal_length"]-row_data["petal_length"])**2
                d4=(row_Centroids["petal_width"]-row_data["petal_width"])**2
                d=np.sqrt(d1+d2+d3+d4)
                ED.append(d)
```

```

        X[cnt]=ED
        cnt=cnt+1
    C=[]
    for index,row in X.iterrows():
        min_dist=row[1]
        pos=1
        for j in range(cluster_Number):
            if row[j+1] < min_dist:
                min_dist = row[j+1]
                pos=j+1
        C.append(pos)
    X["Cluster"]=C
    data_collection=[]
    for p in range(cluster_Number):
        cluster_p=data_cluster(p+1,X_saver,C)
        cost_p=Distance_cluster(cluster_p)
        data_collection.append(cost_p)

    final_cost.append(sum(data_collection))
    Centroids_new = X.groupby(["Cluster"]).mean()[["sepal_length", "sepal_width", "petal_length", "petal_width"]]
    Centroids = X.groupby(["Cluster"]).mean()[["sepal_length", "sepal_width", "petal_length", "petal_width"]]

    W=final_cost[iteration-1]
    B=T-W

    return W,B,final_cost

```

PART A

```

X = df[["sepal_length", "sepal_width", "petal_length", "petal_width"]]
X
T=0.5*Distance_cluster(X)

```

```

T=0.5*Distance_cluster(X)
print("T:",T)

```

T: 14218.184189683327

```

W1,B1,final_cost1=Kmeans_algorithm(5,X,150)
W2,B2,final_cost2=Kmeans_algorithm(10,X,150)
W3,B3,final_cost3=Kmeans_algorithm(20,X,150)

```

```

print("K=5  W/B is:",W1/B1)
print("K=10  W/B is:",W2/B2)
print("K=20  W/B is:",W3/B3)

```

K=5 W/B is: 0.15014813674637276
K=10 W/B is: 0.08459557647328948
K=20 W/B is: 0.019854518772032006

```

plt.plot(final_cost1)
plt.xlabel('Iteration')
plt.ylabel('Cost Function')
plt.title('algorithm for K=5')
plt.show()

```

```

plt.plot(final_cost2)
plt.xlabel('Iteration')
plt.ylabel('Cost Function')
plt.title('algorithm for K=10')

plt.show()

```

```

plt.plot(final_cost3)
plt.xlabel('Iteration')

```

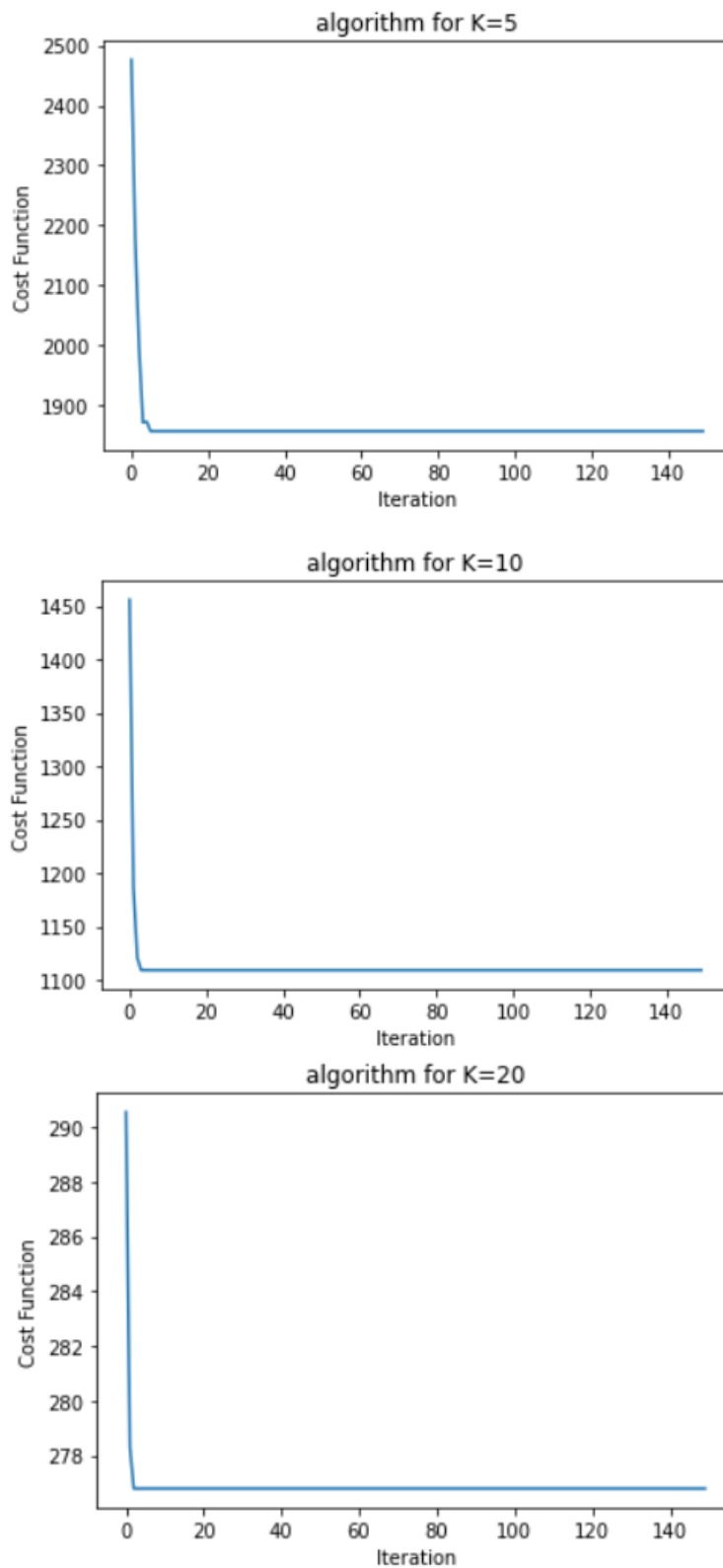
```

plt.ylabel('Cost Function')
plt.title('algorithm for K=20')

plt.show()

```

Further I have provided the results which is cost function during iteration for every K values:5,10,20:



As we can see the algorithm convergence speed is very high but it's reasonable because our dataset is about length and width of plants and the density of these data as of our first observation is so high So despite we have used a random-based approach for initializing the first

centroids but cause of mentioned statement the convergence speed is very high and as you can see under 20 iterations we meet the convergence!

Another considerable note is that I have used the Within similarity criteria as of our cost function and I have calculated this criterion in each iteration and according to class notation it's defined by capital W in my code and also calculated for all number of cluster!

Also as of question description I have calculated T which is the total distance between every pair points and obviously is a constant because during our algorithm the data points are constant and doesn't move at all!

Afterward we know that between criterion which is side similarity defined as B in class notation is also determined by a simple rule:

$$B = T - W$$

And a have used this note to calculate B and reduce our calculation for determining extra distances!

And as my implementation showed I have calculated W/B for all of K values after convergence when we have no long any changes between our centroids and also between our clusters!

And the result provided below:

PART A

```
x = df[["sepal_length", "sepal_width", "petal_length", "petal_width"]]
x
T=0.5*Distance_cluster(X)
print("T:",T)
```


T: 14218.184189683327

```
W1,B1,final_cost1=Kmeans_algorithm(5,X,150)
W2,B2,final_cost2=Kmeans_algorithm(10,X,150)
W3,B3,final_cost3=Kmeans_algorithm(20,X,150)
```

```
print("K=5 W/B is:",W1/B1)
print("K=10 W/B is:",W2/B2)
print("K=20 W/B is:",W3/B3)
```

K=5 W/B is: 0.15014813674637276
K=10 W/B is: 0.08459557647328948
K=20 W/B is: 0.019854518772032006

As we know we tend that this characteristic going to be small as much as it can be because it leads to this fact that the data which is belongs to a specific cluster must be near as much as it can be and in other side the other centroids much be distance as much as it can be which leads B must be big and overall we like to choose the smaller one which according to above figure I have picked up the **K=20** which have the smallest one!

For running this part please have an accurate look on related directory and There you can easily find all of you need and also I would appreciate it if you would consider them


3.2 EFFECT OF REPETITION IN EXPERIMENTS

As we know if we run the program several times we found out that the Cost function and the general form of our clustering has been changed because we choose centroids randomly.

So in this part I have repeated the algorithm 20 times and obtained the Cost function (W) for each of that, finally I have obtained a vector which is our case!

Afterward find the standard deviation and mean of this vector as you can see below:

PART B

```
metricw1=[]
metricB1=[]
metricw2=[]
metricB2=[]
metricw3=[]
metricB3=[]
for i in range(30):
    W1_data,B1_data,final_cost1_data=Kmeans_algorithm(5,X,20)
    W2_data,B2_data,final_cost2_data=Kmeans_algorithm(10,X,20)
    W3_data,B3_data,final_cost3_data=Kmeans_algorithm(20,X,20)
    metricw1.append(W1_data)
    metricw2.append(W2_data)
    metricw3.append(W3_data)
    metricB1.append(B1_data)
    metricB2.append(B2_data)
    metricB3.append(B3_data)
```

print result :Metric W

```
mean1=np.mean(metricw1)
standard_deviation1=np.std(metricw1)

mean2=np.mean(metricw2)
standard_deviation2=np.std(metricw2)

mean3=np.mean(metricw3)
standard_deviation3=np.std(metricw3)

print("K=5 mean is:",mean1)
print("K=5 std is:",standard_deviation1)

print("K=10 mean is:",mean2)
print("K=10 std is:",standard_deviation2)

print("K=20 mean is:",mean3)
print("K=20 std is:",standard_deviation3)

K=5 mean is: 1920.8022181235456
K=5 std is: 307.5212789954384
K=10 mean is: 888.198041151194
K=10 std is: 239.58866576480509
K=20 mean is: 327.5125900229687
K=20 std is: 40.12166610839808
```

print result Metric W/B

```
res1 = list(map(truediv,metricW1,metricB1))
res2 = list(map(truediv,metricW2,metricB2))
res3 = list(map(truediv,metricW3,metricB3))

mean1=np.mean(res1)
standard_deviation1=np.std(res1)

mean2=np.mean(res2)
standard_deviation2=np.std(res2)

mean3=np.mean(res3)
standard_deviation3=np.std(res3)


print("K=5  mean is:",mean1)
print("K=5  std is:",standard_deviation1)

print("K=10  mean is:",mean2)
print("K=10  std is:",standard_deviation2)

print("K=20  mean is:",mean3)
print("K=20  std is:",standard_deviation3)

K=5  mean is: 0.15696740811362997
K=5  std is: 0.03085031125028835
K=10  mean is: 0.06698247131484449
K=10  std is: 0.019525205533771777
K=20  mean is: 0.023586440300000303
K=20  std is: 0.002963662652091988
```

As you can see I have obtained the std and mean for both W as our Cost function metric and also W/B because in practical case we always determine the standard deviation and mean for a Ratio!

For running this part please have an accurate look on related directory and There you can easily find all of you need and also I would appreciate it if you would consider them


4 QUESTION #3

4.1 LOGISTIC REGRESSION

In this part we intend to implement a self-training module which works based on semi-supervised Learning approaches.

As the question description we consider complication column in Surgical-dataset as of our target and rest of that for data.

First of all, let's find out what a Logistic Regression do as a classifier:

Logistic regression is another technique borrowed by machine learning from the field of statistics.

It is the go-to method for binary classification problems (problems with two class values). In this post you will discover the logistic regression algorithm for machine learning.

Logistic Function:

Logistic regression is named for the function used at the core of the method, the logistic function.

The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

$$1 / (1 + e^{\text{-value}})$$

Where e is the base of the natural logarithms (Euler's number or the EXP() function in your spreadsheet) and value is the actual numerical value that you want to transform. Below is a plot of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function.

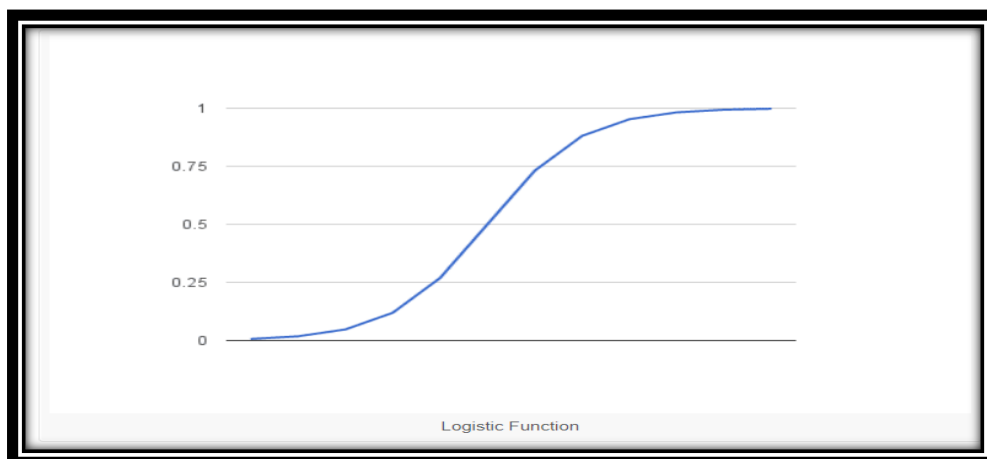


Figure 5: Representation of Logistic

Logistic regression uses an equation as the representation, very much like linear regression.

Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value.

Below is an example logistic regression equation:

$$y = e^{(b_0 + b_1 \cdot x)} / (1 + e^{(b_0 + b_1 \cdot x)})$$

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in our input data has an associated b coefficient (a constant real value) that must be learned from your training data.

The actual representation of the model that we would store in memory or in a file are the coefficients in the equation (the beta value or b's).

Now, let's dive into coding and implement a logisticRegression using Sklearn and use that to train a model which let us to predict unlabelled data!

Further you can see all of my implementation in python:

For running this part please have an accurate look on related directory and There you can easily find all of you need and also I would appreciate it if you would consider them



Semi-Supervised Learning

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn.metrics import plot_confusion_matrix
import random
pd.options.mode.chained_assignment = None
```

```
df = pd.read_csv('Surgical.csv')
df=df.sample(frac=1)
size=len(df)
```

```
def splitting_data(df,tr_percent,te_percent,un_percent):
    train = df.iloc[0:round(size*(tr_percent/100))]
    test = df.iloc[round(size*(tr_percent/100)):round(size*(tr_percent/100))+round(size*(te_percent/100))]
    unlabeled = df.iloc[round(size*(tr_percent/100))+round(size*(te_percent/100)):round(size*(tr_percent/100))+round(size*(te_per

    return train,test,unlabeled
```

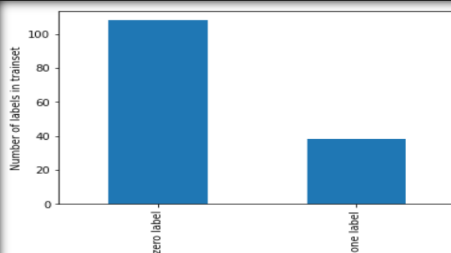
```
def making_label_data(train,test,unlabeled):
    X_train=train[['bmi','Age','asa_status','baseline_cancer' , 'baseline_charlson' , 'baseline_cvd', 'baseline_dementia', 'baseline_electrocardiogram', 'baseline_hemoglobin', 'baseline_hypertension', 'baseline_insulin', 'baseline_lipid', 'baseline_neutrophils', 'baseline_platelets', 'baseline_pulmonary', 'baseline_sodium', 'baseline_surgery', 'baseline_temperature', 'baseline_wbc', 'baseline_wound', 'baseline_xray']]
    X_test=test[['bmi','Age','asa_status','baseline_cancer' , 'baseline_charlson' , 'baseline_cvd', 'baseline_dementia', 'baseline_electrocardiogram', 'baseline_hemoglobin', 'baseline_hypertension', 'baseline_insulin', 'baseline_lipid', 'baseline_neutrophils', 'baseline_platelets', 'baseline_pulmonary', 'baseline_sodium', 'baseline_surgery', 'baseline_temperature', 'baseline_wbc', 'baseline_wound', 'baseline_xray']]
    X_unlabeled=unlabeled[['bmi','Age','asa_status','baseline_cancer' , 'baseline_charlson' , 'baseline_cvd', 'baseline_dementia', 'baseline_electrocardiogram', 'baseline_hemoglobin', 'baseline_hypertension', 'baseline_insulin', 'baseline_lipid', 'baseline_neutrophils', 'baseline_platelets', 'baseline_pulmonary', 'baseline_sodium', 'baseline_surgery', 'baseline_temperature', 'baseline_wbc', 'baseline_wound', 'baseline_xray']]
    y_train=train['complication']
    y_test=test['complication']

    return X_train,X_test,X_unlabeled,y_train,y_test
```

Logistic Regression

```
tr_percent=1
te_percent=25
un_percent=74
train,test,unlabeled=splitting_data(df,tr_percent,te_percent,un_percent)
X_train,X_test,X_unlabeled,y_train,y_test=making_label_data(train,test,unlabeled)
```

```
y_train.value_counts().plot(kind='bar')
plt.xticks([0,1], ['zero label', 'one label'])
plt.ylabel('Number of labels in trainset');
```



Pre Model Evaluation

```
model = LogisticRegression()
model.fit(X_train, y_train)
y_predict= model.predict(X_test)
pre_f1_score = f1_score(y_test,y_predict)
accuracy=model.score(X_test,y_test)
plot_confusion_matrix(model,X_test,y_test,cmap='Blues',normalize='true',display_labels=['zero label','one label']);

print("Pre f1 Score:",pre_f1_score)
print("Pre accuracy:",accuracy)

Pre f1 Score: 0.3657142857142857
Pre accuracy: 0.757310740639519
```

As of question description I have provided results there:

Histogram of Complication and Non-Complication which means 0 labels and 1 labels:

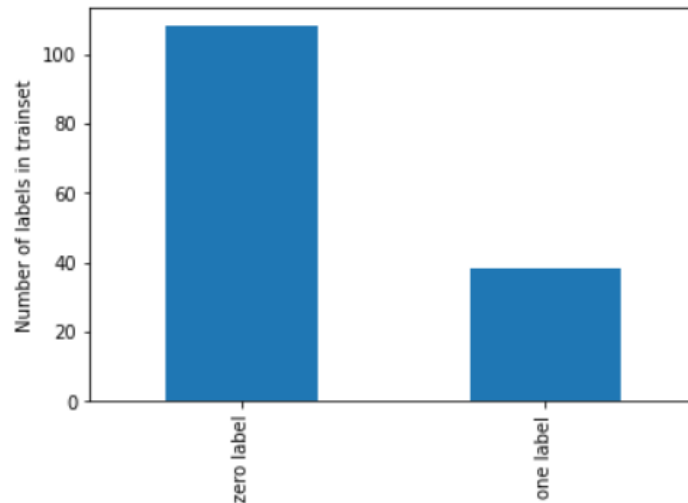


Figure 6: Histogram of Train classes

As we can see it's very unbalanced and the numbers of zero label in comparison to one label is very high!

Note that in this part I split the data frame-work according to below percentage:

1% Train (Labeled)

25% Test (Labeled)

74% Unlabeled

Important notes:

As we said in previous part the majority of zero labels is high in comparison to 1 labels in this case the accuracy metric from sklearn may not be a good one for evaluating the classifier. So as you see in my code I have used F1-score as of my metrics to evaluate the model but for don't missing any important evaluation data I have reported the accuracy along the F1-score to have a deeper insight!

4.2 PRE MODEL EVALUATION

In this part we only declare LogisticRegression and then train that to be prepared for predicting the test_data.

The result of evaluation has been attached below:



Figure 7: Histogram of Train classes

As we can see the F1-score is 0.36 which is low but it's reasonable because we have only used from one percent of train labels to train model so we expect to get a better accuracy and also better F1-score where we will use the semi-supervised approaches for training the model.

Further you can see the confusion matrix which is a good match with our result in previous part.

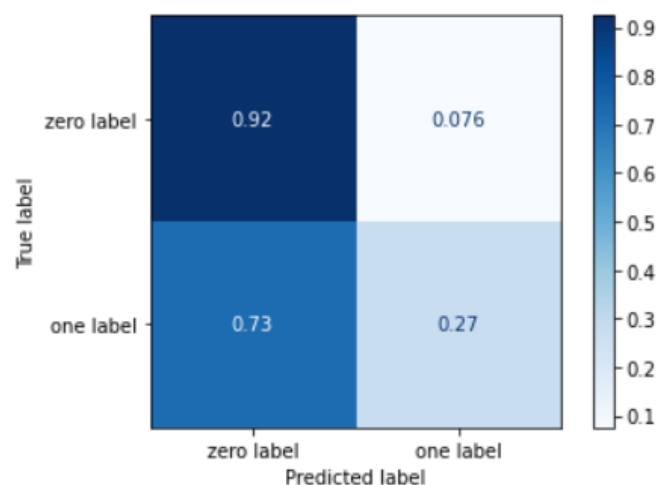


Figure 8: Confusion matrix

4.3 SEMI SUPERVISED LEARNING (SELF TRAINING APPROACH)

Self-training Algorithm:

Self-training can be abbreviated in several Steps which is provided below:

Step 1:

First, we must train a Logistic Regression classifier on the labeled training data.

Step 2:

Next, we use the classifier to predict labels for all unlabeled data, as well as probabilities for those predictions. In this case, we will only accept labels which have a greater value in comparison with our probability threshold for predictions.

This approach is called Threshold-based self training there is another one which is called K-best but in this project we don't use that.

Step 3:

Concatenate the accepted unlabeled data with the labeled training data, and re-train the classifier on the concatenated data.

Step 4:

Use trained classifier to make predictions for the labeled test data, and evaluate the classifier.

Repeat steps 1 through 4 until no more predictions have greater than Threshold probability, or no unlabeled data remains!

Further I have provided whole of my implementation in python:

For running this part please have an accurate look on related directory and There you can easily find all of you need and also I would appreciate it if you would consider them



Self Training Algorithm

```
def Updating_train_data_and_Dropping_unlabeled_data(X_train,y_train,X_unlabeled,y_unlabeled,best_predicted):
    X_train_new = pd.concat([X_train,X_unlabeled.loc[best_predicted.index]], axis=0)
    y_train_new = pd.concat([y_train,best_predicted.y_unlabeled])
    X_unlabeled_new = X_unlabeled.drop(index=best_predicted.index)
    Num_remaining_unlabel=len(X_unlabeled_new)

    return X_train_new,y_train_new,X_unlabeled_new,Num_remaining_unlabel
```

```
def Checking_threshold_and_adding_unlabel_to_train(storing_frame,threshold):

    best_pick_zero=storing_frame.loc[storing_frame["p0"] > threshold]
    best_pick_one=storing_frame.loc[storing_frame["p1"] > threshold]
    best_predicted = pd.concat([best_pick_zero,best_pick_one],axis=0)
    Num_adding_unlabel=len(best_predicted)

    return Num_adding_unlabel,best_predicted
```

```
def making_correspond_unlabels_with_correspond_probabilities(y_unlabeled,Probabilities,X_unlabeled):
    storing_frame = pd.DataFrame([])
    storing_frame["y_unlabeled"] = y_unlabeled
    storing_frame["p0"] = Probabilities[:,0]
    storing_frame["p1"] = Probabilities[:,1]
    storing_frame.index = X_unlabeled.index

    return storing_frame
```

```
def unlabeled_prediction_each_step_logistic_regression(X_train,y_train,X_test,y_test,X_unlabeled):
    model = LogisticRegression(max_iter=1000)
    model.fit(X_train, y_train)
    y_predict = model.predict(X_test)
    Post_f1_score =f1_score(y_test,y_predict)
    Post_accuracy=model.score(X_test,y_test)
    Probabilities = model.predict_proba(X_unlabeled)
    y_unlabeled = model.predict(X_unlabeled)

    return Post_f1_score,Post_accuracy,Probabilities,y_unlabeled
```

```
def self_Training_Algorithm(X_train,y_train,X_test,y_test,X_unlabeled,threshold):
    cnt=0
    best_predicted_values=[]
    f1_score_values=[]
    accuracy_values=[]
    best_predicted=random.randint(1, 20)
    while len(best_predicted) > 0:
        Post_f1_score,Post_accuracy,Probabilities,y_unlabeled=unlabeled_prediction_each_step_logistic_regression(X_train,y_train,
        storing_frame=making_correspond_unlabels_with_correspond_probabilities(y_unlabeled,Probabilities,X_unlabeled)
        Num_adding_unlabel,best_predicted=Checking_threshold_and_adding_unlabel_to_train(storing_frame,threshold)
        X_train_new,y_train_new,X_unlabeled_new,Num_remaining_unlabel=Updating_train_data_and_Dropping_unlabeled_data(X_train,y_train,

        if(cnt !=0):
            f1_score_values.append(Post_f1_score)
            accuracy_values.append(Post_accuracy)
            best_predicted_values.append(Num_adding_unlabel)
            print(f"End of Iteration {cnt}")
            print(f"added {Num_adding_unlabel} new labels")
            print("f1 Score:",Post_f1_score)
            print("accuracy:",Post_accuracy)
            print(f"remaining {Num_remaining_unlabel} unlabeled")
```

```
X_train=X_train_new
y_train=y_train_new
X_unlabeled=X_unlabeled_new
cnt=cnt+1
```

```
return f1_score_values,accuracy_values,best_predicted_values
```

```
threshold=0.99
f1_score_values,accuracy_values,best_predicted_values=self_Training_Algorithm(X_train,y_train,X_test,y_test,X_unlabeled,threshold)
```

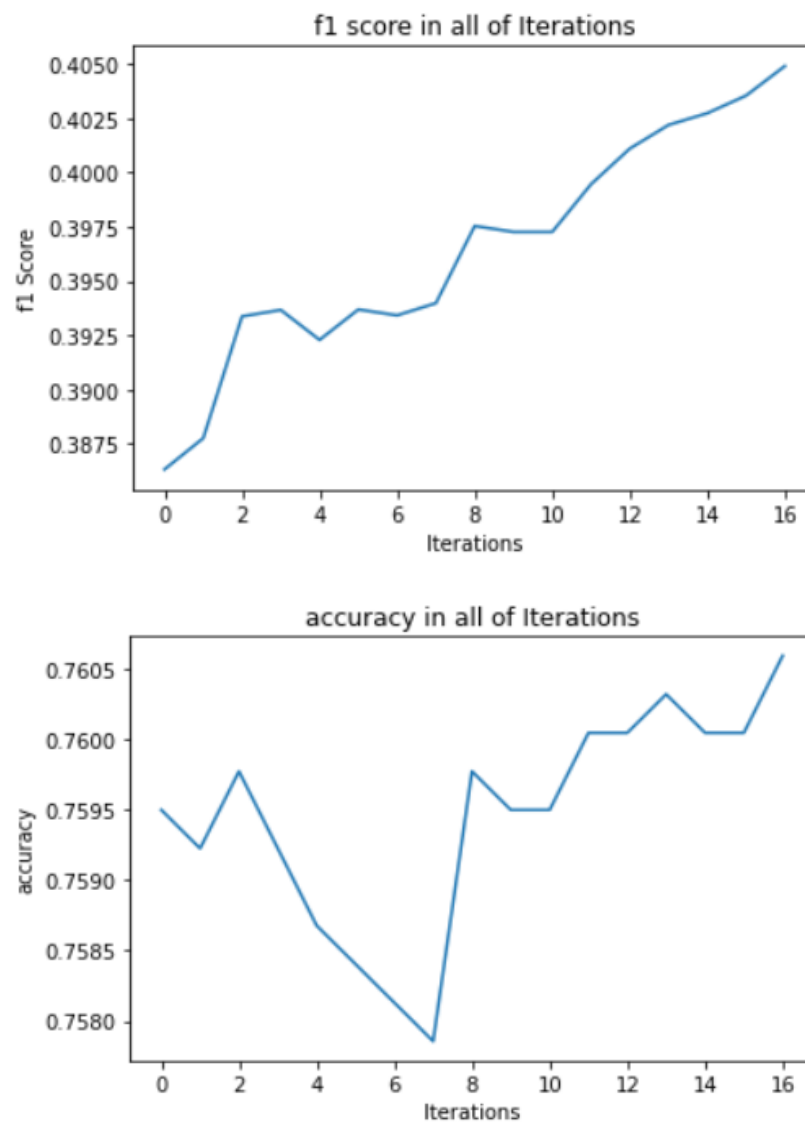
Post Self-training Model Evaluation

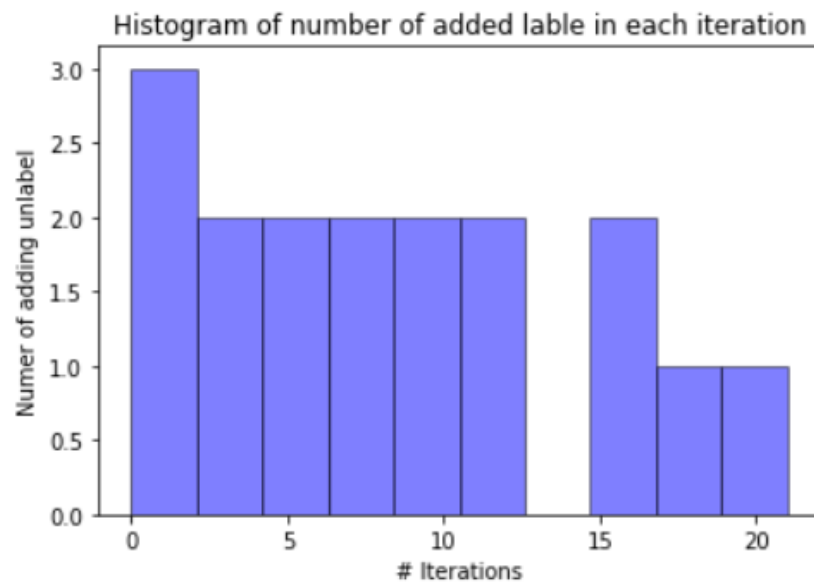
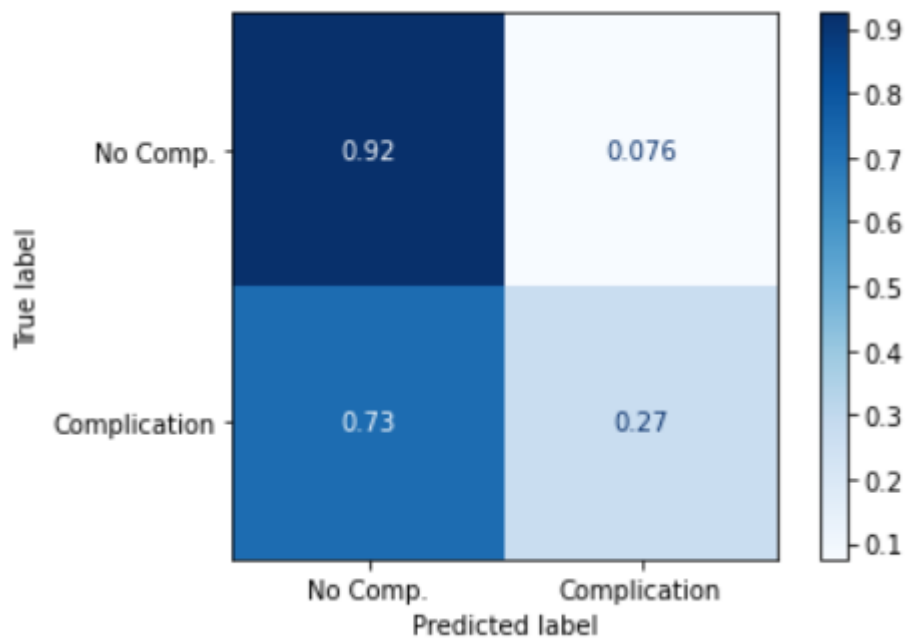
```
plt.plot(f1_score_values)
plt.xlabel('Iterations')
plt.ylabel('f1 Score')
plt.title('f1 score in all of Iterations')
plt.show()

plt.plot(accuracy_values)
plt.xlabel('Iterations')
plt.ylabel('accuracy')
plt.title('accuracy in all of Iterations')
plt.show()

plt.hist(best_predicted_values,color='blue', alpha=0.5, edgecolor='black')
plt.ylabel('Number of adding unlabel')
plt.xlabel('# Iterations')
plt.title('Histogram of number of added lable in each iteration')
plt.show()

plot_confusion_matrix(model, x_test, y_test, cmap='Blues', normalize='true',display_labels=['No Comp.', 'Complication']);
```

**Figure 9: Representing Performance**

**Figure 10: Histogram****Figure 11: Confusion matrix**

Over the 17 iterations, the F1 score and accuracy improved despite it's only a very small increase, it looks like self-training has improved the classifier's performance on the test data set so we satisfy the expectation of semi-supervised algorithm.

Final-notes:

But we must note that despite we are including our accepted-unlabelled data with labeled training data, some of the predictions for unlabelled are going to be false! Because despite the model say us that the probability for these cases is high but in these case we face with lots of unlabelled data in comparison to train label date so always we may be in a danger of missing performance and gets us a worse result in some cases!

4.4 CHANGING THRESHOLD VALUE AND EVALUATE MODEL

In this part I have used a for loop and pass over an interval of 0.7 to 0.99 as of our threshold and for each of them calculated the F1-score and accuracy-one and afterward plot the results.

As you can see all of F1-score and also accuracy values are in a same region and there is not any high deviation between that so in this case we can choose 0.99 threshold which seems to be better cause of using high probabilities!

Determining The Best Threshold values

```
# from 0.7 to 0.99 threshold
f1_score_array=[]
accuracy_array=[]
T_interval=np.arange(70,100,1)/100
for threshold in T_interval:
    f1_score_values,accuracy_values,best_predicted_values=self_Training_Algorithm(X_train,y_train,X_test,y_test,X_unlabeled,threshold)
    f1_score_array.append(f1_score_values[-1])
    accuracy_array.append(accuracy_values[-1])
```

```
plt.plot(T_interval,f1_score_array)
plt.xlabel('Threshold')
plt.ylabel('f1 Score')
plt.title('f1 Score during Threshold')
plt.show()
```

```
plt.plot(T_interval,accuracy_array)
plt.xlabel('Iterations')
plt.ylabel('accuracy')
plt.title('Accuracy during Threshold')
plt.show()
```

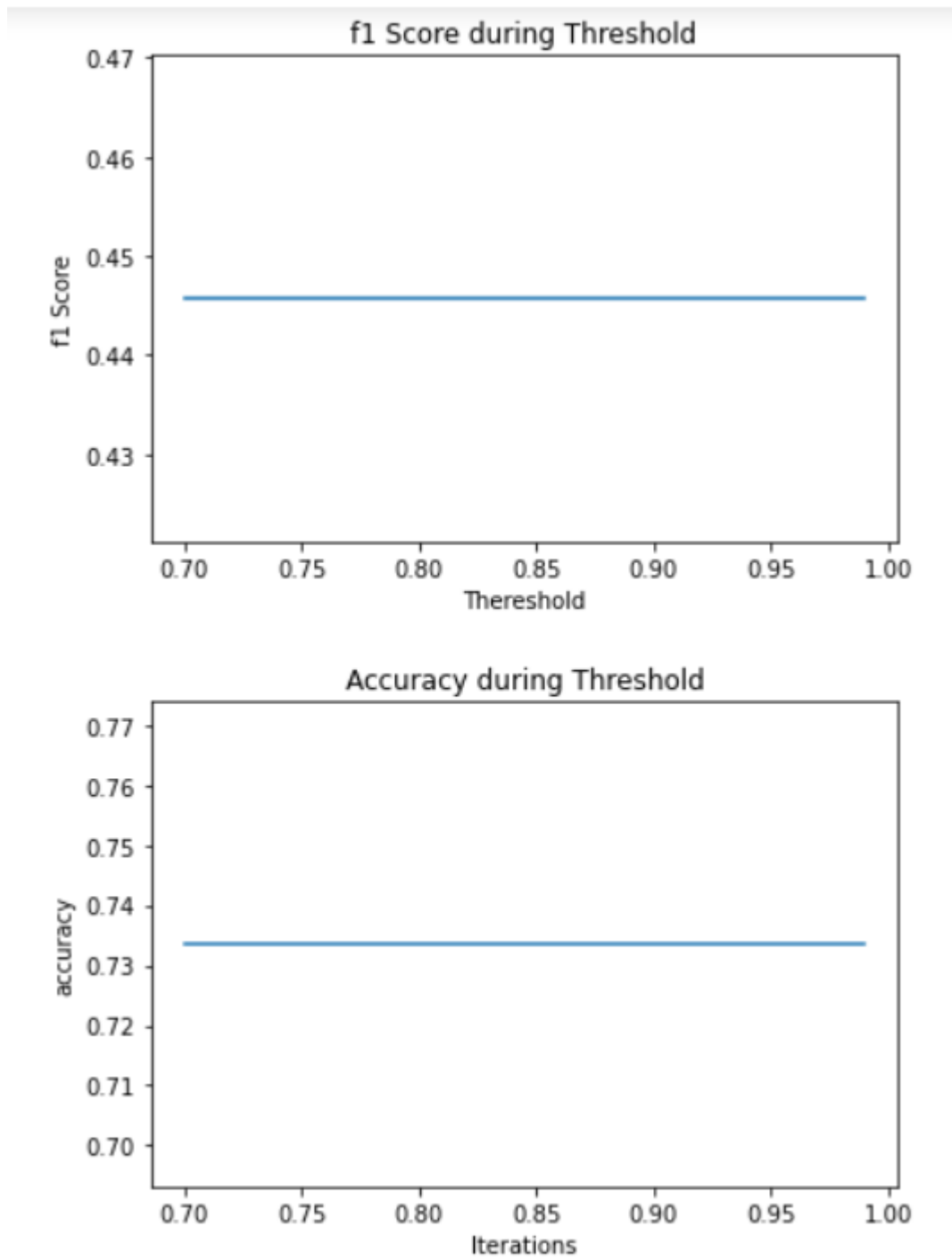


Figure 12: Accuracy during Threshold

5 QUESTION #4

5.1 PROBABILITY (THEORETICAL BASED)

In this part we investigate some common problems about statistics further I have provide my solutions!

I would appreciate it if you would consider them.

5.1.1 Un-Fair Coin Problem

Q1) we know that in this case we have a un-Fair coin with probability of p

so if we want to make a un-biased decision

we have to \Rightarrow two-iteration

if we get $p(1-p) = pq$ or $(1-p)p = qp$ in 2 iteration

we can easily have a same and Fair probabilities For

choosing between two people.

in Fact we have a key Rule $\Rightarrow pq = qp$

so in this way we can do that.

5.1.2 Evaluation of Reaching Time in Bus & Metro Problem

Q2) $f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & o.w \end{cases}$ $E\{x_1\} = \frac{1}{\lambda_1}$ $E\{x_2\} = \frac{1}{\lambda_2}$

$\begin{cases} U = \min(x, Y) \\ V = \max(x, Y) \end{cases} \Rightarrow UV = XY \xrightarrow{*} E\{UV\} = E\{XY\} = E\{x\}E\{y\} = \frac{1}{\lambda_1 \lambda_2}$

$F_U(u) = 1 - [1 - F_1(u)][1 - F_2(u)] = 1 - (1 - F_1(u) + F_1(u) - F_1(u)F_2(u)) = 1 - 1 + F_2(u) + F_1(u) - F_1(u)F_2(u)$

$\frac{d}{du} \Rightarrow f_U(u) = f_2(u) + f_1(u) - f_1(u)F_2(u) - F_1(u)f_2(u)$, $F_U(u) = 1 - e^{-\lambda_1 u}$

$\Rightarrow f_U(u) = \lambda_1 e^{-\lambda_1 u} (1 - e^{-\lambda_2 u}) + \lambda_2 e^{-\lambda_2 u} (1 - e^{-\lambda_1 u}) = \lambda_1 e^{-\lambda_1 u} - \lambda_1 e^{-(\lambda_1 + \lambda_2)u} + \lambda_2 e^{-(\lambda_1 + \lambda_2)u} - \lambda_2 e^{-\lambda_2 u}$

$\Rightarrow f_U(u) = \lambda_1 e^{-\lambda_1 u} + \lambda_2 e^{-\lambda_2 u} - (\lambda_1 + \lambda_2) e^{-(\lambda_1 + \lambda_2)u}$

$\int \Rightarrow E\{U\} = \frac{1}{\lambda_1} + \frac{1}{\lambda_2} - \frac{1}{\lambda_1 + \lambda_2} = \frac{\lambda_1^2 + \lambda_2^2 + \lambda_1 \lambda_2}{\lambda_1 \lambda_2 (\lambda_1 + \lambda_2)}$

now for $F_V(v) = F_1(v) \times F_2(v) = F_1(v)F_2(v) + F_1(v)f_2(v)$

$\Rightarrow f_V(v) = \lambda_1 e^{-\lambda_1 v} + \lambda_2 e^{-\lambda_2 v} - \lambda_1 e^{-\lambda_1 v} - \lambda_2 e^{-\lambda_2 v} + (\lambda_1 + \lambda_2) e^{-(\lambda_1 + \lambda_2)v}$

$\Rightarrow E\{V\} = \frac{1}{\lambda_1 + \lambda_2}$

$Cov\{U, V\} = E\{XY\} - E\{U\}E\{V\} = \frac{1}{\lambda_1 \lambda_2} - \frac{(\lambda_1 + \lambda_2)^2 - \lambda_1 \lambda_2}{(\lambda_1 + \lambda_2)^2 \lambda_1 \lambda_2}$

$= \frac{\lambda_1 \lambda_2}{\lambda_1 \lambda_2 (\lambda_1 + \lambda_2)^2} = \frac{1}{(\lambda_1 + \lambda_2)^2} = Cov(u, v)$

Q2-2)

I have used the math. stack exchange to understand the problem and get an idea to solve that through below approaches $\rightarrow X, Y \sim \mathcal{N}(0, 1)$

$$\begin{cases} V = \max(X, Y) \\ W = \min(X, Y) \end{cases} \rightarrow \text{Cov}(X, V) = E\{XV\} - E\{X\}E\{V\} = E\{XV\} \quad (3)$$

$$\text{simply } \rightarrow V = X \mathbb{1}_{Y < X} + Y \mathbb{1}_{X < Y}$$

$$E\{XV\} = E\{X^2; Y < X\} + E\{XY; X < Y\}$$

$$\text{by symmetry we have: } E\{XY; X < Y\} = E\{XY; Y < X\} \quad (*)$$

$$\rightarrow X, Y \sim \mathcal{N}(0, 1) \xrightarrow{\text{i.i.d}} E\{X\}E\{Y\} = E\{XY\} = 0 \quad (**)$$

$$\begin{matrix} (*) \\ \oplus \\ (*) \end{matrix} \rightarrow E\{XY; X < Y\} = 0 \xrightarrow{\text{yield}} E\{XV\} = E\{X^2 F(X)\} \quad (1)$$

CDF

$$\text{we know that } F(-X) = 1 - F(X)$$

$$\rightarrow E\{X^2 F(X)\} = E\{(-X)^2 F(-X)\} = E\{X^2 (1 - F(X))\} = E\{X^2\} - E\{X^2 F(X)\}$$

$$\Rightarrow 2E\{X^2 F(X)\} = E\{X^2\} \Rightarrow E\{X^2 F(X)\} = \frac{E\{X^2\}}{2} \quad (2)$$

$$\textcircled{1}, \textcircled{2} \xrightarrow{\textcircled{3}} E\{XV\} = \text{Cov}(X, V) = \frac{1}{2} E\{X^2\}$$

$$\Rightarrow \text{Cov}(X, \max(X, Y)) = \frac{1}{2} \sigma^2$$

$$\min(-X, -Y) = -\max(X, Y) \xrightarrow{\text{symmetry}} \text{Cov}(X, \min(X, Y)) = \frac{1}{2} \text{var}(X)$$

$$\Rightarrow \text{Cov}(X, \min(X, Y)) = \frac{1}{2} \sigma^2$$

5.1.3 Employee of Programming Company

$$Q3) \quad f(x,y) = c[1-x-y] \Rightarrow \iint f(x,y) = 1$$



$$\int_0^1 \int_0^{1-x} c(1-x-y) dy dx$$

$$= \int_0^1 \left[y - xy - \frac{y^2}{2} \right]_0^{1-x} dx$$

$$= \int_0^1 1-x-x(1-x) - \frac{(1-x)^2}{2} dx = c \left[x - \frac{x^2}{2} - \frac{x^2}{2} + \frac{x^3}{3} + \frac{(1-x)^3}{6} \right]_0^1$$

$$= \left[\frac{1}{3} - \frac{1}{6} \right] c = \frac{c}{6} = 1 \rightarrow \boxed{c = 6}$$

$$f(x,y) = 6[1-x-y], \quad f_x(x) = \int_0^{1-x} (6-6x-6y) dy$$

$$= 6(1-x) - 6x(1-x) - 3(1-x)^2 = 6-6x-6x+6x^2-3-3x^2+6x$$

$$\Rightarrow \boxed{f_x(x) = 3x^2 - 6x + 3}$$

$$\Rightarrow F_x(x) = \int_0^x 3x^2 - 6x + 3 = x^3 - 3x^2 + 3x \Big|_0^x$$

$$= x^3 - 3x^2 + 3x = F(x) \rightarrow \Pr\{x < 0.5\} = F_x(0.5) = \frac{1}{8} - \frac{3}{4} + \frac{3}{2}$$

$$= \frac{1-6+12}{8} = \boxed{\frac{7}{8}}$$

it represents the probability that the cumulative time works
for one of employee tend to be lower half of work

and it's so probable because we expect every employee

do the half of work so the value $\frac{7}{8}$ is matched

with our expectations.

$$\begin{aligned}
E\{X+Y\} &= \int_0^1 \int_0^{1-x} 6(x+y)(1-x-y) \, dy \, dx = 6x - 6x^2 - 6xy + 6x - 6x^2 - 6xy \\
&= \int_0^1 \int_0^{1-x} -12x^2 - 12xy + 12x \, dy \, dx \\
&= \int_0^1 -12x^2(1-x) - 6x(1-x)^2 + 12(1-x)x \, dx = -12x^2 + 12x^3 - 6x - 6x^3 + 12x^2 + 12x \\
&\Rightarrow = -4x^3 + 3x^4 - 3x^2 - \frac{6}{4}x^4 + 4x^3 + 6x^2 - 4x \Big|_0^1 \\
&= -4 + 3 - 3 - \frac{6}{4} + 4 + 6 - 4 = 2 - 1.5 = \boxed{0.5}
\end{aligned}$$

$E\{X+Y\}$ represents that the mean of sum of time work for both person and as we guess at the

First This value 0.5 is matched well with our expectation.

5.2 PROBABILITY (SIMULATION BASED)

In This part we are going to analyse and implement some common and simple Probability problem which give us a deeper insight over different stochastic and statistical concepts!

5.2.1 Birthday Problem

In this part we intend to investigate and implement very common problem sometimes known as birthday paradox problem!

Our approach in this problem is a such a way that we make a random vector which represent the data of birth for every single person who considered as of our subjective!

And afterward in a number of $n=10000$ experiments we go through the vector and count the same cases.

Note that I have used a very fantastic syntax (in my opinion!) to count same birthday.

We know that in python there is a syntax called `set()` and the note is that if we detect any same birth in our vector then the set function will have several cells which represent the same pair of birth element so if we calculate length of `set()` we found that it's not equal with the first vector and all the point is checking following condition, And going through this way we can easily found that we have two same birth or not because if we haven't any match case the length of random vector is equal which length of set of that vector and I think it's so tricky and beautiful one to implement our problem.

```
len(random_people_birthday) == len(set(random_people_birthday))
```

further you can easily find the full implementation of my birthday problem:

Birthday Problem

```
import random
import numpy as np
import matplotlib.pyplot as plt

def Birthday_problem(n):
    flag_same_birthday=1
    flag_unsame_birthday=0
    prob_data=[]
    experiment_number=10000
    for i in range(experiment_number):
        random_people_birthday = []
        for j in range(n):
            rnd_data = random.randint(0, 365)
            random_people_birthday.append(rnd_data)

        if len(random_people_birthday) == len(set(random_people_birthday)):
            prob_data.append(flag_unsame_birthday)
        else:
            prob_data.append(flag_same_birthday)

    probability=np.sum(prob_data)/(np.size(prob_data))

    return probability
```

Plotting Results

```

: data_saver=[]
  people=[]
  for i in range(100):
    prob=Birthday_problem(i)
    people.append(i)
    data_saver.append(prob)

plt.plot(people,data_saver,'red')
plt.xlabel("Number of people")
plt.ylabel("Probability of Prodox birthday")
plt.title('Probability during Number of people')
plt.show()

```

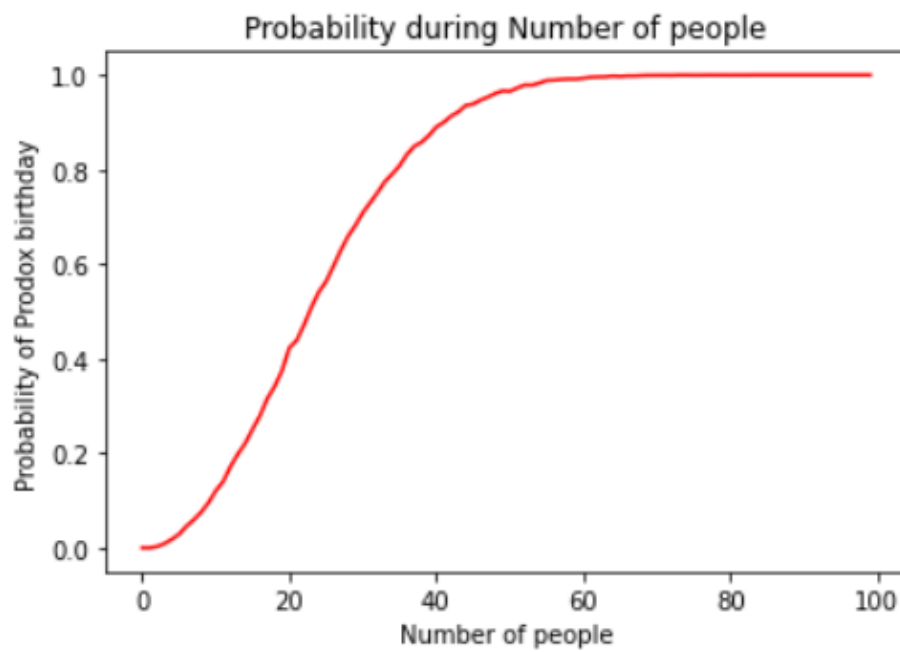
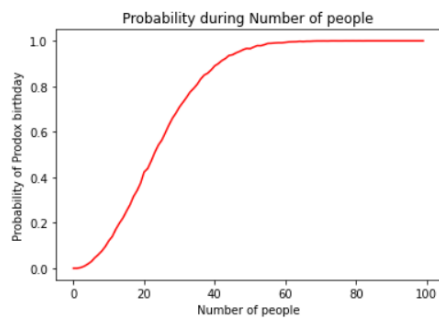


Figure 13: our Desired Curve for Exponential One

5.2.2 Central Limit Theorem

In this part we intend to investigate central Limit Theorem using two different distributions which are binomial and exponential distribution and see how these two random vectors with sampling tend to have same behaviour with normal distribution.

First let's find out the theory which is hidden behind this fact:

The Central Limit Theorem (CLT) is often referred to as one of the most important theorems, not only in statistics but also in the sciences as a whole, we will try to understand the essence of the Central Limit Theorem with simulations in Python.

The core point here is that the sample mean itself is a random variable, which is dependent on the sample observations!

Like any other random variable in statistics, the sample mean (\bar{x}) also has a probability distribution, which shows the probability densities for different values of the sample mean.

This distribution is often referred to as the sampling distribution.

We Suppose X is a random variable(not necessarily normal) representing the population data, And, the distribution of X , has a mean of μ and standard deviation σ . Suppose we are taking repeated samples of size 'n' from the above population.

Then, the Central Limit Theorem states that given a high enough sample size, the following properties hold true:

Sampling distribution's mean = Population mean (μ)

Sampling distribution's standard deviation (standard error) = $\frac{\sigma}{\sqrt{n}}$

such that for $n \geq 30$, the sampling distribution tends to a normal distribution for all practical purposes.

In other words, for a large n:

$$\bar{x} \rightarrow \mathbb{N}\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$$

Above statement has a key role in our analyses!

Let's moving forward and have a deeper insight on this theorem!

We prove this state in case of two different distributions but results have proven that if n tend to be large so with a high probability and with a considerable error we reach this fact in both simulation and theoretical approaches.

5.2.2.1 Exponentially distributed population:

First we have implemented exponential distribution in python using `np.random.exponential()` beautiful function.

Note that the exponential distribution has a considerable parameter called rate, such that both the mean and the standard deviation of the distribution are given by $(1/\lambda)$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Simulation Algorithm:

As we mentioned before we have used $\lambda=2$ in our case and actually the algorithm is such a way that we sample from the mentioned distribution for several times and in this way we will have different random vectors which all of that satisfy our expectation which is having normal distribution.

Note that in our experiment we have repeated the whole process for $s=100000$ In my code and also we have chosen $n=1200$ in both cases whether we face with binomial or whether we face with exponential distribution!

And please care that being to large is a necessary condition for our s parameter to have a smooth curve as of our results for an ideal normal distribution.

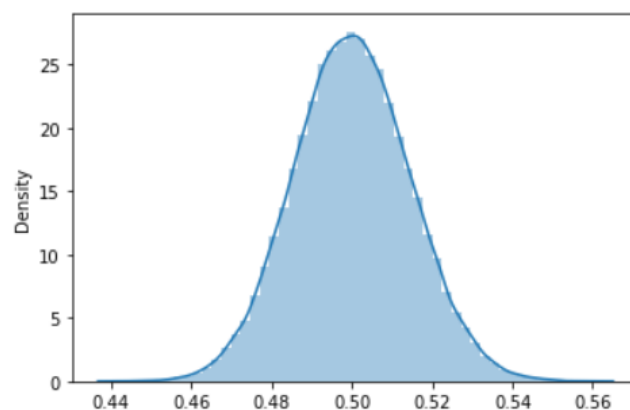


Figure 14: our Desired Curve for Exponential One

As of question description to show that all of thing has a good match with our theoretical concept kindly discussed in previous part further I have provided mean and std which is obtained in both theoretical approach or in a simulation-based approach:

Mean and standard deviation of our exponential distribution:

Central Limit Theorem

Exponentially Distributed Population

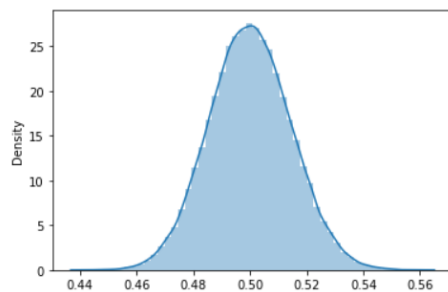
```
: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

: landa=2
n=1200
s=100000
data_collection=np.zeros(s)
for i in range(s):
    sampling = np.random.exponential((1/landa),n)
    data_collection[i]=sampling.mean()

print('mean:',1/landa)
print('standard deviation:',np.sqrt(1/(landa**2)))

mean: 0.5
standard deviation: 0.5
```

```
sns.distplot(data_collection);
```



```
data_collection
```

```
array([0.48982496, 0.49533005, 0.50128748, ..., 0.50583853, 0.5008476 ,
       0.50293664])
```

```
print(data_collection.mean())
print(1/landa)
```

```
0.5000101775894811
0.5
```

```
print((1/(landa))/ np.sqrt(n))
data_collection.std()
```

```
0.014433756729740642
0.014465507419749677
```

As you can see we have a good match between theoretical and simulation analyses.

Note that I have obtained the mean and std for new random vector with have the same behaviour with normal distribution and also I have used from `sns.distplot()` function which let us know the density of vector points around x domain, in other word this function provide us density of random vector point around every single point and in other way it gives us the PDF of our vector, as you can see I have used that to estimate the PDF value of each random point and then plot the curve and investigate whether it has normal behaviour or not and hopefully and theoretically it satisfy all of our expectation from a normal distribution!

Similarly, we can observe that the standard deviation of the 1200 sample means is quite close to the value stated by the CLT, i.e., $(\sigma/\sqrt{n}) = 0.014$ and in the same case for mean!

5.2.2.2 BINOMIALLY DISTRIBUTED POPULATION:

First we have implemented BINOMIALLY distribution in python using `np.random.binomial()` beautiful function.

Note that the BINOMIALLY distribution has a considerable parameter called n & p , such that both the mean and the standard deviation of the distribution are given by:

$$E(X) = kp = \mu$$

$$SD(X) = \sqrt{kp(1-p)} = \sigma$$

$$P(x) = \begin{cases} \binom{k}{x} (p)^x (1-p)^{1-x} & \text{if } x = 0, 1, 2, \dots, k \\ 0 & \text{otherwise} \end{cases}$$

Simulation Algorithm:

As we mentioned before we have used $n=20, p=0.8$ in our case and actually the algorithm is such a way that we sample from the mentioned distribution for several times and in this way we will have different random vectors which all of that satisfy our expectation which is having normal distribution.

Note that in our experiment we have repeated the whole process for $s=100000$ In my code and also we have chosen $n=1200$ in both cases whether we face with binomial or whether we face with exponential distribution!

And please care that being to large is a necessary condition for our s parameter to have a smooth curve as of our results for an ideal normal distribution.

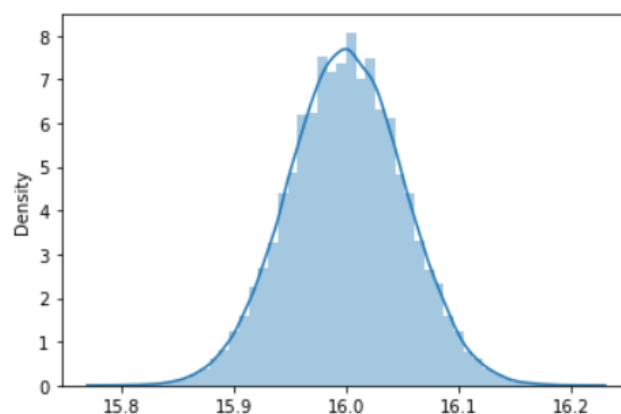
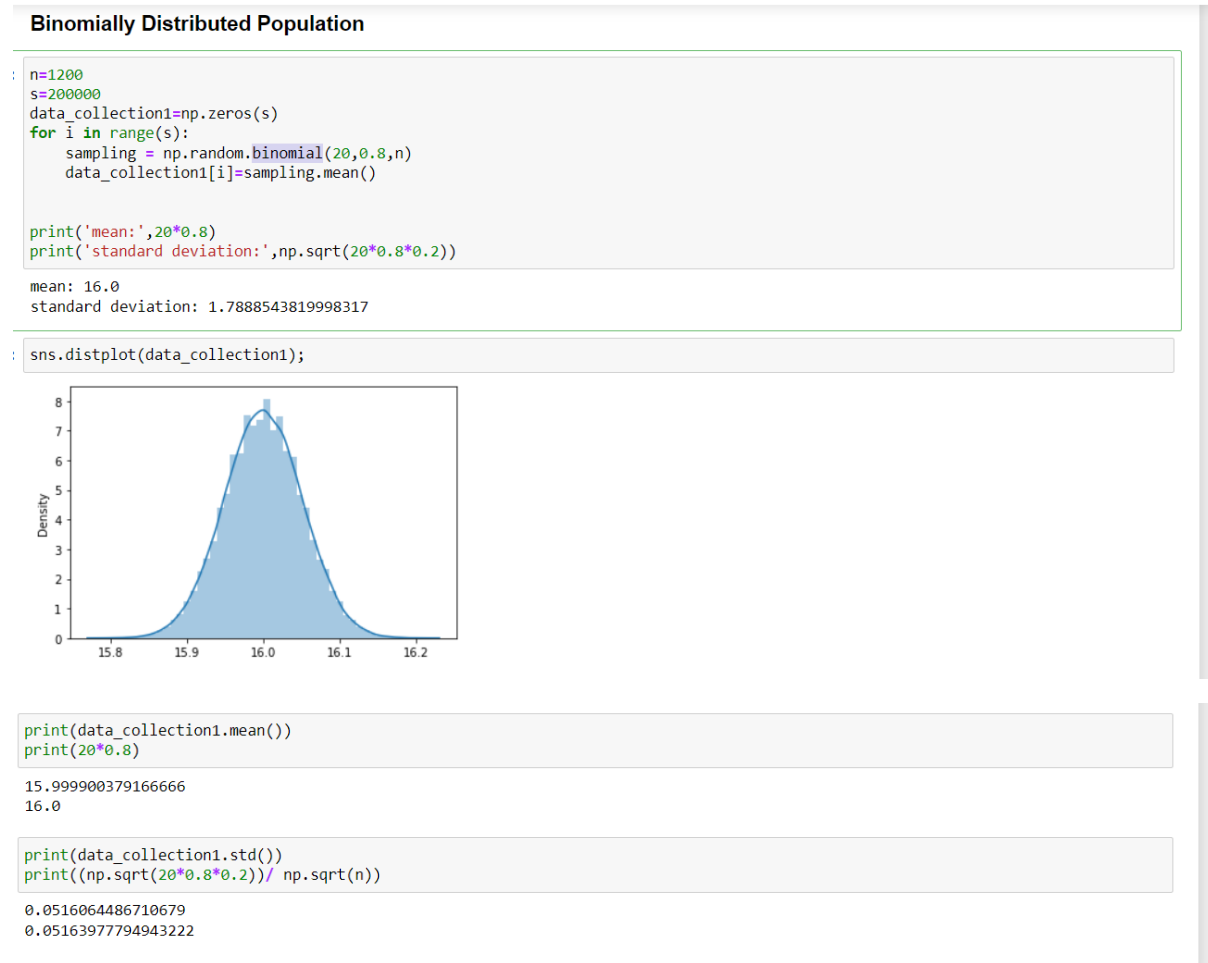


Figure 15: our Desired Curve for Binomial One

As of question description to show that all of thing has a good match with our theoretical concept kindly discussed in previous part further I have provided mean and std which is obtained in both theoretical approach or in a simulation-based approach:

Mean and standard deviation of our exponential distribution:



As you can see we have a good match between theoretical and simulation analyses.

Note that I have obtained the mean and std for new random vector with have the same behaviour with normal distribution and also I have used from `sns.distplot()` function which let us know the density of vector points around x domain, in other word this function provide us density of random vector point around every single point and in other way it gives us the PDF of our vector, as you can see I have used that to estimate the PDF value of each random point and then plot the curve and investigate whether it has normal behaviour or not and hopefully and theoretically it satisfy all of our expectation from a normal distribution!

Similarly, we can observe that the standard deviation of the 1200 sample means is quite close to the value stated by the CLT, i.e., $(\sigma/\sqrt{n}) = 0.0516$ and in the same case for mean!

6 ACKNOWLEDGEMENT

I am really grateful for Mr MohammadHossein Vaeedi (810197605) because in some part of this project we had a nice and effective discussion which highly helped us to have a better analyse and also have more adequate results! (Note that we only talked about ideas behind different problems.)

Afterwards, I am thankful to all of course teaching assistants: NarjesNoorzad(njnoorzad@gmail.com) and Amirhossein Rokni (a.rokni@ut.ac.ir) and RezaTalakoob (rezatalakoob@yahoo.com) and SalarNoori(salar.nouri@ut.ac.ir) who designed this project with high quality.

7 REFERENCES

- [1] <https://people.revoledu.com/kardi/tutorial/kMean/NumericalExample.htm>
- [2] <https://www.analyticsvidhya.com/blog/2019/05/beginners-guide-hierarchical-clustering/>
- [3] https://www.tutorialspoint.com/matplotlib/matplotlib_bar_plot.htm
- [4] <https://www.machinelearningplus.com/plots/matplotlib-histogram-python-examples/>
- [5] Testing Birthday Paradox in Faker Library (Python) | by Elena Kosourova | Towards Data Science
- [6] https://en.wikipedia.org/wiki/Exponential_distribution
- [7] <https://blog.quantinsti.com/central-limit-theorem/>
- [8] Creating Histograms of Well Log Data Using Matplotlib in Python | by Andy McDonald | Towards Data Science
- [9] <https://towardsdatascience.com/using-histograms-to-visualise-well-log-data-16142e2c7f81>
- [10] <https://math.stackexchange.com/questions/307650/compute-operatornamecovx-maxx-y-and-operatornamecovx-minx-y>
- [11] <https://stats.stackexchange.com/questions/25965/what-is-covx-y-where-x-minu-v-and-y-maxu-v-for-independent-uniform0-1-v>
- [12] <https://medium.com/data-folks-indonesia/step-by-step-to-understanding-k-means-clustering-and-implementation-with-sklearn-b55803f519d6>