Assignment 2

anare@purdue.edu

1. Problem 1

   (a) What is the principal assumption in the Naive Bayes' model, and when is this assumption useful?
   Solution.
   The existence of a given feature in a class is assumed to be independent to the presence of any other feature by the Naive Bayes classifier.
   For example, if a fruit is red, round, and roughly 3 inches in diameter, it is termed an apple. Even if these characteristics are reliant on one another or on the presence of other characteristics, they all add to the likelihood that this fruit is an apple, which is why it is called "Naive."

   (b) When do we expect k-NN to be better than logistic regression?
   Solution.
   k-NN allows for non-linear solutions, whereas Logistic Regression only allows for linear solutions. In other words, when there is no dependence between the features we can expect k-NN to perform better than Logistic regression.

   (c) Suppose we have two classes. 150 examples in the + class and 50 examples in the - class. What is the entropy of the class variable (you can leave this in terms of logs)?
   Solution.
   Given: Examples in +ve = 150 and in −ve class = 50
   We know that, $Entropy = -\sum p \log p$ where p = probability of each class

   $\mathbb{P}\left[+ve\right] = \frac{150}{200} = \frac{3}{4}$
   $\mathbb{P}\left[-ve\right] = \frac{50}{200} = \frac{1}{4}$

   Therefore,

$$Entropy = -\frac{3}{4}\log_2(\frac{3}{4}) - \frac{1}{4}\log_2(\frac{1}{4})$$

$$= -(-0.31125 - 0.5)$$

$$= 0.81125$$

(d) Assume that you observed the following number of requests to a web server from two domains, Domain A and Domain B (Domain A: Class1, Domain B: Class2).

Number of Requests from Domain A $= \{2, 3, 4, 3, 4, 3, 3, 4, 5, 3, 2, 3, 4, 3, 2, 2, 3, 4, 5, 6\}$
Number of Requests from Domain B$= \{22, 23, 24, 23, 24, 23, 23, 24, 25, 23\}$

Using the data, estimate the mean, std. dev. and prior (that is, P(A)/P(A + B) and P(B)/P(A + B)) for each class.
Solution.

$$\mu_A = \frac{\sum A}{\mathbb{N}} = \frac{2+3+4+3+4+3+3+4+5+3+2+3+4+3+2+2+3+4+5+6}{20}$$

$$\mu_A = \frac{68}{20} = 3.4$$

Similarly,

$$\mu_B = \frac{\sum B}{\mathbb{N}} = \frac{22+23+24+23+24+23+23+24+25+23}{10}$$

$$\mu_B = \frac{234}{10} = 23.4$$

Now for σ_A we know that $\sigma = \sqrt{\frac{\sum(A-\mu)^2}{N}} \implies \sigma_A = \sqrt{\frac{\sum_{a\in A}(a-3.4)^2}{20}}$

A	$A - \mu_a$	$(A - \mu_a)^2$
2	-1.4	1.96
3	-0.4	0.16
4	0.6	0.36
3	-0.4	0.16
4	0.6	0.36
3	-0.4	0.16
3	-0.4	0.16
4	0.6	0.36
5	1.6	2.56
3	-0.4	0.16
2	-1.4	1.96
3	-0.4	0.16
4	0.6	0.36
3	-0.4	0.16
2	-1.4	1.96
2	-1.4	1.96
3	-0.4	0.16
4	0.6	0.36
5	1.6	2.56
6	2.6	5.76
		$\sum = 22.8$

$\sigma_A = \sqrt{\frac{22.8}{20}} = 1.067$

$$prior_A = \frac{\mathbb{P}[A]}{\mathbb{P}[A+B]} = \frac{68}{302} = \frac{34}{151}$$

Now for σ_B we know that $\sigma = \sqrt{\frac{\sum(B-\mu_b)^2}{N}} \implies \sigma_B = \sqrt{\frac{\sum_{b\in B}(b-23.4)^2}{10}}$

B	$B-\mu_b$	$(B-\mu_b)^2$
22	-1.4	1.96
23	-0.4	0.16
24	0.6	0.36
23	-0.4	0.16
24	0.6	0.36
23	-0.4	0.16
23	-0.4	0.16
24	0.6	0.36
25	1.6	2.56
23	-0.4	2.56
		$\sum = 6.4$

$\sigma_B = \sqrt{\frac{6.4}{10}} = 0.8$

$$prior_B = \frac{\mathbb{P}[B]}{\mathbb{P}[A+B]} = \frac{234}{302} = \frac{117}{151}$$

(e) We'd like to model each conditional probability of the form $\mathbb{P}\left[x(i)|y\right]$ as a normal distribution for the training data below. We then use the naive Bayes assumption to write an expression for: $\mathbb{P}\left[y=0|x\right]\,\mathbb{P}\left[y=1|x\right]$. Please show you answer clearly.

Solution.

Solving for $x(1)$ we get,

Class			μ	σ
0	4	-4	0	5.65
1	2	10	6	5.65

Using Gaussian equation to get the probabilities we get,

$$\mathbb{P}\left[x(1)|y=0\right] = \frac{1}{5.65\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-0}{5.65}\right)^2\right) \ldots eq(1)$$

$$\mathbb{P}\left[x(1)|y=1\right] = \frac{1}{5.65\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-6}{5.65}\right)^2\right) \ldots eq(2)$$

Similarly, solving for $x(2)$ we get,

Class			μ	σ
0	7	5	6	1.414
1	10	4	7	4.24

$$\mathbb{P}\left[x(2)|y=0\right] = \frac{1}{1.414\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-6}{1.414}\right)^2\right) \ldots eq(3)$$

$$\mathbb{P}\left[x(2)|y=1\right] = \frac{1}{4.24\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-7}{4.24}\right)^2\right) \ldots eq(4)$$

Assuming the features are independent we can write the following,

$$\mathbb{P}\left[y=0|X\right] = eq(3)\times eq(1) = \frac{1}{1.414\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-6}{1.414}\right)^2\right)\times\frac{1}{5.65\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-0}{5.65}\right)^2\right)$$

$$\mathbb{P}\left[y=1|X\right] = eq(4)\times eq(2) = \frac{1}{4.24\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-7}{4.24}\right)^2\right)\times\frac{1}{5.65\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-6}{5.65}\right)^2\right)$$

Therefore,

$$\mathbb{P}\left[y=0|X\right]\times\mathbb{P}\left[y=1|X\right] = \frac{1}{1.414\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-6}{1.414}\right)^2\right)\times\frac{1}{5.65\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-0}{5.65}\right)^2\right)$$

$$\times\frac{1}{4.24\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-7}{4.24}\right)^2\right)\times\frac{1}{5.65\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-6}{5.65}\right)^2\right)$$

2. Problem 2

   (a) Which attribute would the algorithm choose to use for the root of the tree? Show the details of your calculations. Recall from lectures that if we let S denote the data set at current node, A denote the feature with values $v \in V$, H denote the entropy function, and S_v denote the subset of S for which the feature A has the value v, the gain of a split along the feature A, denoted $InfoGain(S, A)$ is computed as:

$$InfoGain(S, A) = H(S) - \sum_{v \in V} \left(\frac{|S_v|}{|S|} \right) H(S_v)$$

That is, we are taking the difference of the entropy before the split, and subtracting the entropy of each new node after splitting, with an appropriate weight depending on the size of each node.

Solution.

$$H(Y) = -\left(\frac{9}{16}\right) \log_2 \left(\frac{9}{16}\right) - \left(\frac{7}{16}\right) \log_2 \left(\frac{7}{16}\right)$$

$$H(Y) = 0.9886$$

We now split by **Color**:
Yellow = 13; Green = 3

$$H(Y|Yellow) = -\left(\frac{8}{13}\right) \log_2 \left(\frac{8}{13}\right) - \left(\frac{5}{13}\right) \log_2 \left(\frac{5}{13}\right)$$

$$H(Y|Yellow) = 0.9611$$

$$H(Y|Green) = -\frac{1}{3} \log_2 \left(\frac{1}{3}\right) - \frac{2}{3} \log_2 \left(\frac{2}{3}\right))$$

$$H(Y|Green) = 0.9182$$

Information Gain (Color) = $H(Y)\breve{}H(Y|Color)$
Information Gain (Color) = $0.9886 - \left(\frac{13}{16} \times 0.9611 + \frac{3}{16} \times 0.9182\right)$
Information Gain (Color) = 0.0357

We now split by **Size**:
Small = 8; Large = 8

$$H(Y|Small) = -\frac{6}{8} \log_2 \frac{6}{8} - \frac{2}{8} \log_2 \frac{2}{8}$$

$$H(Y|Small) = 0.8112$$

$$H(Y|Large) = -\frac{3}{8}\log_2\frac{3}{8} - \frac{5}{8}\log_2\frac{5}{8}$$

$$H(Y|Large) = 0.9544$$

Information Gain (Size) $= H(Y)\check{}H(Y|Size)$
Information Gain (Size) $= 0.9886 - \left(\frac{8}{16} \times 0.8112 + \frac{8}{16} \times 0.9544\right)$
Information Gain (Size) $= 0.1058$

We now split by **Shape**:
Regular = 12; Irregular = 4

$$H(Y|Regular) = -\frac{6}{12}\log_2\frac{6}{12} - \frac{6}{12}\log_2\frac{6}{12}$$

$$H(Y|Regular) = 1$$

$$H(Y|Irregular) = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4}$$

$$H(Y|Irregular) = 0.8112$$

Information Gain (Shape) $= H(Y)\check{}H(Y|Size)$
Information Gain (Shape) $= 0.9886 - \left(\frac{12}{16} \times 1 + \frac{4}{16} \times 0.8112\right)$
Information Gain (Shape) $= 0.0358$

Therefore, the Information Gain is highest for "Size" and hence it is the root node of the decision tree.

Now, let us consider only the **Small** branch of the tree.

$$H(Y) = -\frac{6}{8}\log_2\frac{6}{8} - \frac{2}{8}\log_2\frac{2}{8}$$

$$H(Y) = 0.8112$$

We now split by **Color**:
Yellow = 6; Green = 2

$$H(Y|Yellow) = -\frac{5}{6}\log_2\frac{5}{6} - \frac{1}{6}\log_2\frac{1}{6}$$

$$H(Y|Yellow) = 0.6500$$

$$H(Y|Green) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2}$$

$$H(Y|Green) = 1$$

Information Gain (Color) = $H(Y)\breve{\ }H(Y|Color)$
Information Gain (Color) = $0.8112 - [\frac{6}{8} \times 0.6500 + \frac{2}{8} \times 1]$
Information Gain (Color) = 0.0737

We now split by **Shape**:
Round = 6; Irregular = 2

$$H(Y|Round) = -\frac{4}{6}\log_2\frac{4}{6} - \frac{2}{6}\log_2\frac{2}{6}$$

$$H(Y|Round) = 0.9182$$

$$H(Y|Irregular) = -\frac{2}{2}\log_2\frac{2}{2} - 0$$

$$H(Y|Irregular) = 0$$

Information Gain (Shape) = $H(Y)\breve{\ }H(Y|Shape)$
Information Gain (Shape) = $0.8112 - [\frac{6}{8} \times 0.9182 + 0]$
Information Gain (Shape) = 0.1227

Therefore, the Information Gain is highest for **Shape** and hence it is the next node of the decision tree under the "Small" branch.
Now, let us consider only the "Large" branch of the tree.

$$H(Y) = -\frac{3}{8}\log_2\frac{3}{8} - \frac{5}{8}\log_2\frac{5}{8}$$

$$H(Y) = 0.9544$$

We now split by **Color**:
Yellow = 7; Green = 1

$$H(Y|Yellow) = -\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7}$$

$$H(Y|Yellow) = 0.9852$$

$$H(Y|Green) = -\frac{1}{1}\log_2\frac{1}{1}$$

$$H(Y|Green) = 0$$

Information Gain (Color) = $H(Y)\breve{\ }H(Y|Color)$
Information Gain (Color) = $0.9544 - [\frac{7}{8} \times 0.9852 + 0]$

Information Gain (Color) = 0.0924

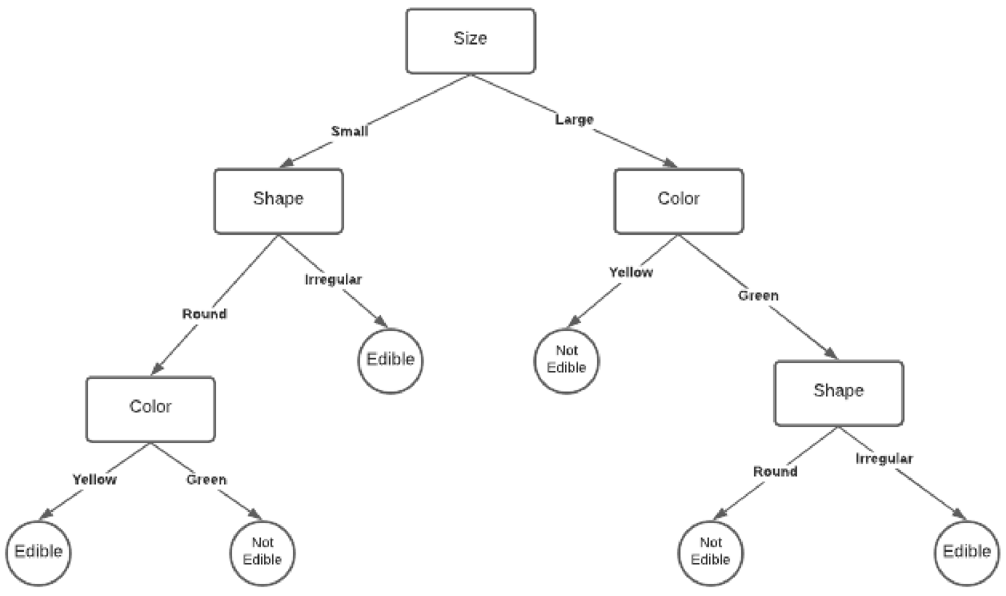
We now split by **Shape**:
Round = 6; Irregular = 2

$$H(Y|Round) = -\frac{4}{6}\log_2\frac{4}{6} - \frac{2}{6}\log_2\frac{2}{6}$$
$$H(Y|Round) = 0.9182$$
$$H(Y|Irregular) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2}$$
$$H(Y|Irregular) = 1$$

Information Gain (Shape) = $H(Y)$ˇ$H(Y|Shape)$
Information Gain (Shape) = $0.9544 - [\frac{6}{8} \times 0.9182 + \frac{2}{8} \times 1]$
Information Gain (Shape) = 0.0158

Therefore, the Information Gain is highest for "Color" and hence it is the next node of the decision tree under the "Large" branch.

(b) Draw the full decision tree that would be learned for this data (assume no pruning and you stop splitting a leaf node when all samples in the node belong to the same class, i.e., there is no information gain in splitting the node).
Solution.



(c) Table 1 includes only categorical features. However, some datasets such as the ones in security often include real valued (numerical) features. Handling real valued (numerical) features is totally different from categorical features in splitting nodes. In this problem, we look for a simple solution to decide good thresholds for splitting based on numerical features. Specifically, when there is a numerical feature in data, an option would be treating all numeric values of feature as discrete, i.e., proceeding exactly as we do with categorical data. Do any problems arise when we use a tree derived this way to classify an unseen example? Please support your claims

Solution.

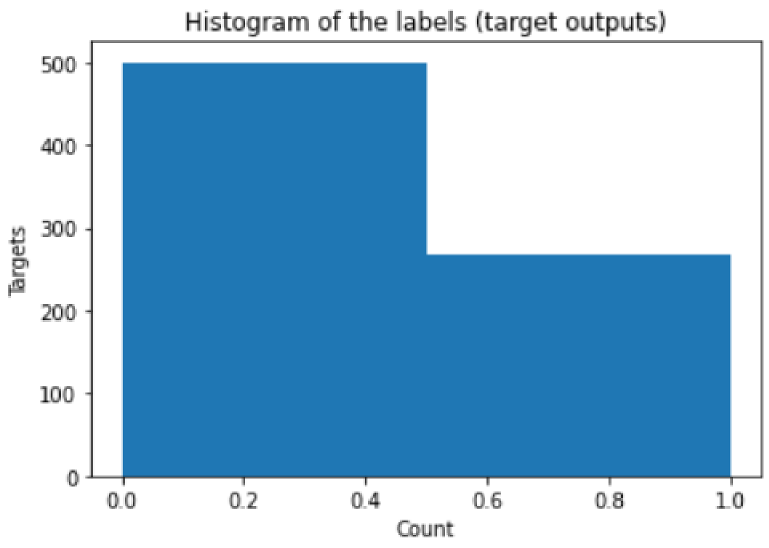
When there is a numerical feature in the data, one alternative is to treat all numeric values of the feature as discrete, similar to how we do with categorical data.

When dealing with categorical data and treating it as discrete, we must choose the splitting criteria that will yield the most information. If our data regularly flips the output, we won't be able to find a solid and adequate dividing criterion. In this scenario, the decision tree that was constructed may not have provided us with correct results.

3. Problem 3

   (a) Report the statistics of each feature (min, max, average, standard deviation) and the histogram of the labels (target outputs).

   Solution. Refer to Problem3.ipynb

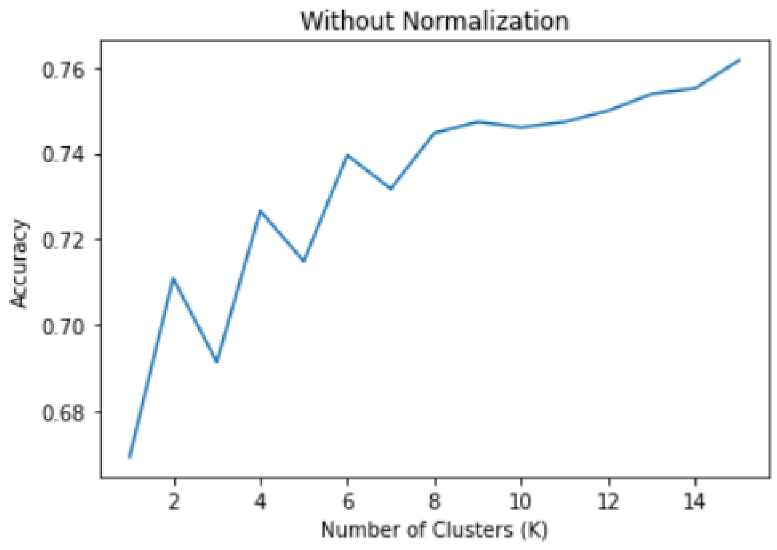


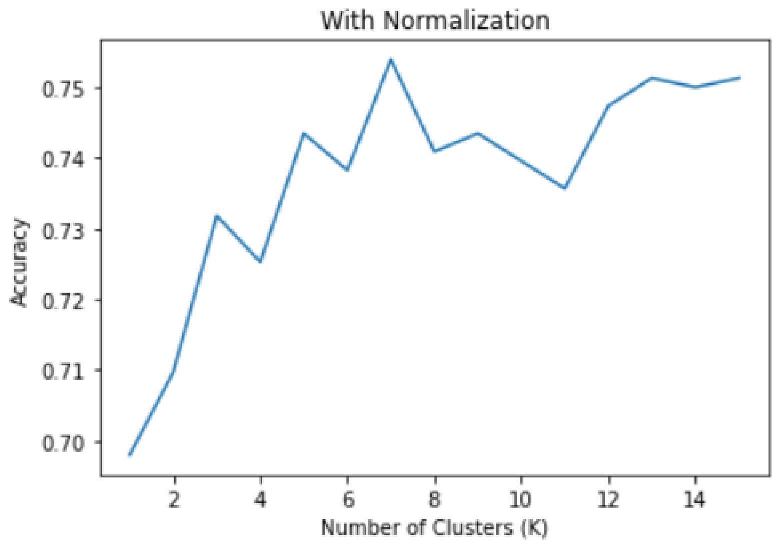
   (b) Include the plot in your report and justify your decision for picking a particular number of neighbors k.

   Solution.

   The value of K that I chose is 15 as it gives us a highest accuracy when training without regularization.

   The value of K that I chose is 7 as it gives us a highest accuracy when training with regularization.





(c) Evaluate the k-NN algorithm on test data with the optimal number of neighbours you obtained in previous step and report the test error (you will use the same value of k you have found in 2.).

Solution. Refer to Problem3.ipynb

(d) Process the input data by subtracting the mean (a.k.a. centralization) and dividing by the standard deviation (a.k.a. standardization) over each dimension (feature), repeat the previous part and report the accuracy. Do centralization and standardization impact the accuracy? Why?

Solution.

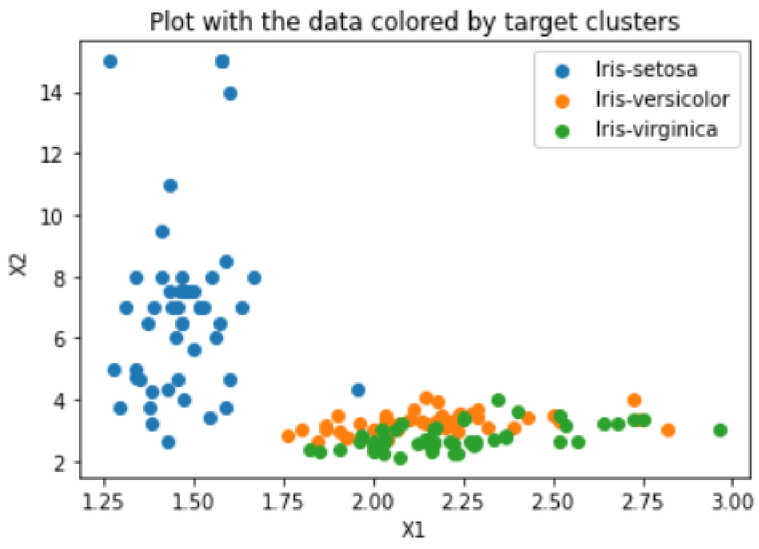
Yes, standardization and centralization (that is normalization) enhances the accuracy but not always. Normalization has eliminated the outliers from the dataset, which were causing multiple extra classes earlier.

Sometimes, the accuracy might not increase after normalization, it is due to the random initialization of points.

4. Problem 4

(a) Create a new data set with two features by computing the ratio of raw features x = (x1, x2) where x1 = (sepal length/sepal width) and x2 = (petal length/petal width)) and plot the data to observe the clusters in data by yourself (use class label to color the data points for better illustration of clusters).

Solution. Refer to Problem4.ipynb



(b) Implement the k-means++ algorithm. Submit the source code (documented!) of your implementation.

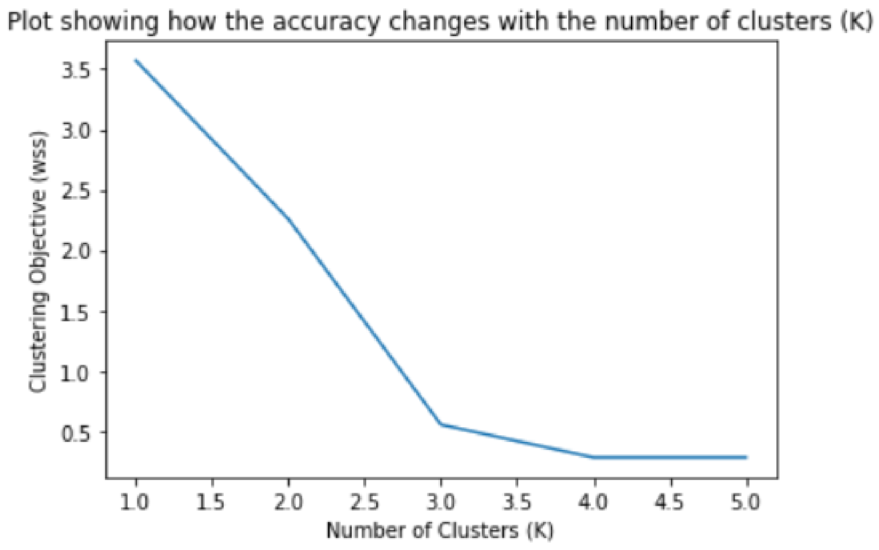
Solution.

K-Means ++ Algorithm (implemented from scratch)

```
def euclidean_dist(x1,y1,x2,y2):
    return np.sqrt((x1-x2)**2 + (y1-y2)**2)

#Function for Smart Initializing centroids using K-means ++ algorithm.
#Takes the data(x = feature1 (X1 in this case),
#y = feature2 (X2 in this case)) and number of clusters as input
#Outputs the centroids selected for initialization of K-means

def initialize_kmeans(x,y,k):
    c=[]

    # Choose a random point from the dataset

    r=random.randrange(len(x))
    random_point=[]
    random_point.append(x[r])
    random_point.append(y[r])
    c.append(random_point)

    for i in range(k-1):
        index = 0
        maximum = 0
        point=[]

        # Identify the farthest point (index) from the selected centroids
        # 1. Calculate Euclidean distance of all points from the
        #nearest selected centroids.
        # 2. Select the point with the maximum (farthest) euclidean distance.
        for j in range(len(x)):
            dist=[]
            for l in range(len(c)):
                dist.append(euclidean_dist(x[j],y[j],c[l][0],c[l][1]))
            min_dist = min(dist)
            if (min_dist>maximum):
                maximum = min_dist
                index = j
        point.append(x[index])
        point.append(y[index])
        c.append(point)
    return c

#Function to update centroids. I takes the data (points),
#old centroids and number of clusters as input
#and returns intermediate labels, clustered points and the updated centroids.

def calculate_next_centroids(x,y,old_centroids,k):

    #Cluster the points to its nearest centroid

    nearest_centroid=[]
    for i in range(len(x)):
        dist=[]
        for j in range(len(old_centroids)):
            dist.append(euclidean_dist(x[i],
            y[i],
            old_centroids[j][0],
            old_centroids[j][1]))
        dist = np.array(dist, dtype=np.float64)
        nearest_centroid.append(np.argmin(dist))

    nearest_centroid = np.array(nearest_centroid, dtype=np.int8)

    #Based on above clusters, compute the centroids of the points that belong
    #to a cluster. This will act as new centroids for our clustering.

    next_centroids=[]
    cluster=[]
    for i in range(k):
        cluster_x=[]
        cluster_y=[]
        cluster_x_y=[]
        cluster_centroid=[]
        for j in range(len(x)):
            if(nearest_centroid[j]==i):
                cluster_x.append(x[j])
                cluster_y.append(y[j])
            cluster_x_y.append(cluster_x)
            cluster_x_y.append(cluster_y)
            cluster.append(cluster_x_y)

            cluster_centroid.append(np.mean(cluster_x))
            cluster_centroid.append(np.mean(cluster_y))
            next_centroids.append(cluster_centroid)

        return nearest_centroid, cluster, next_centroids

    def kmeans_plus_plus(x,y,targets,k):
        centroids = initialize_kmeans(x, y, k)
        delta_max = []

        for j in range(50):

            #old_centroids
            old_centroids=centroids

            #update centroids
            labels, cluster, centroids = calculate_next_centroids(x, y, centroids, k)

            #new_centroids
            new_centroids=centroids

            delta=[]
            for i in range(len(centroids)):
                #Calculate the Within the cluster Square Sum to compare
                wss = (np.mean((cluster[i][0] - np.mean(cluster[i][0]))**2)
                +
                np.mean((cluster[i][1] - np.mean(cluster[i][1]))**2))/2


                delta.append(wss)

            #Calculate the clustering objective
            delta_max.append(np.mean(delta))

        return cluster, centroids, delta_max, wss
```

(c) Cluster the modified Iris dataset with the two features explained above. Run your algorithm 50 times over the data with different values of clusters k = 1, 2, . . . , 5 and plot the accuracies (x and y axes should be the number of clusters and the clustering objective.

Solution.



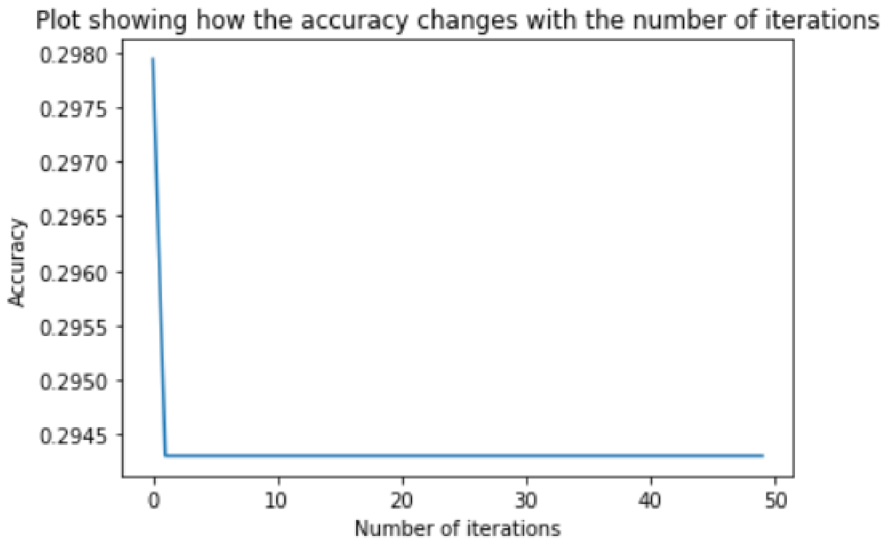
(d) Based on the above plot, decide the number of final clusters and justify your answer. For the chosen number of clusters,

Solution.

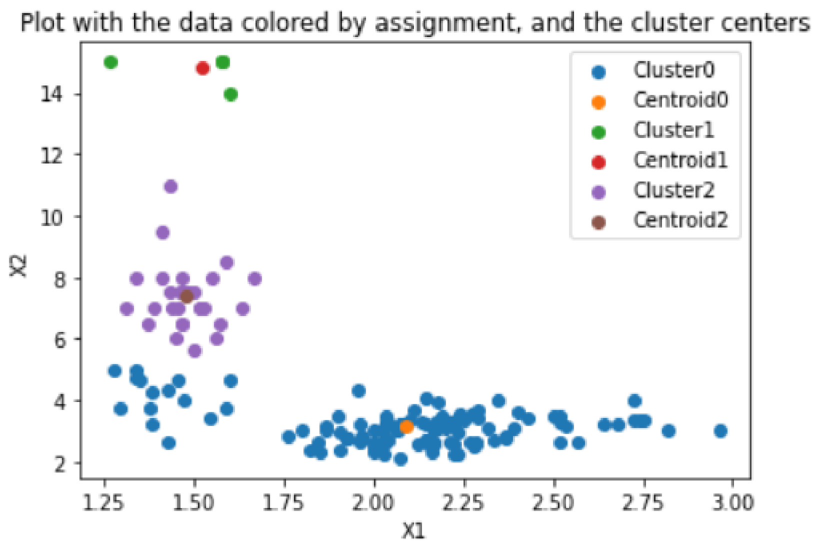
According to the above plot, the clustering objective for 1 depicts underfitting and clustering objective for 5 depicts overfitting as there are some unwanted clusters.

The optimal values is for 3 clusters where the clustering objective graph depicts optimal clustering

- Create a plot showing how the accuracy changes with the number of iterations.



- Create a plot with the data colored by assignment, and the cluster centers.



5. Problem 5

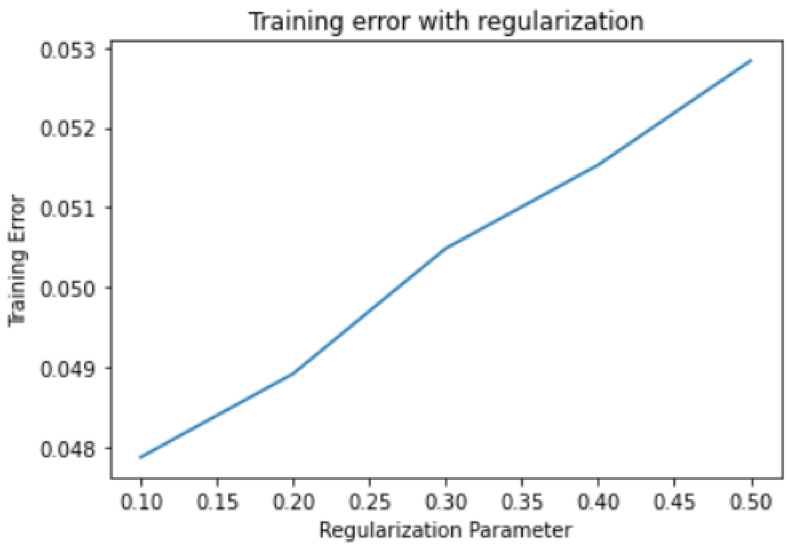
   (a) Use the logistic regression (multi class; implemented from scratch) training algorithm and plot the training and test errors.
   Solution.
   Training error: 0.03635888046037139

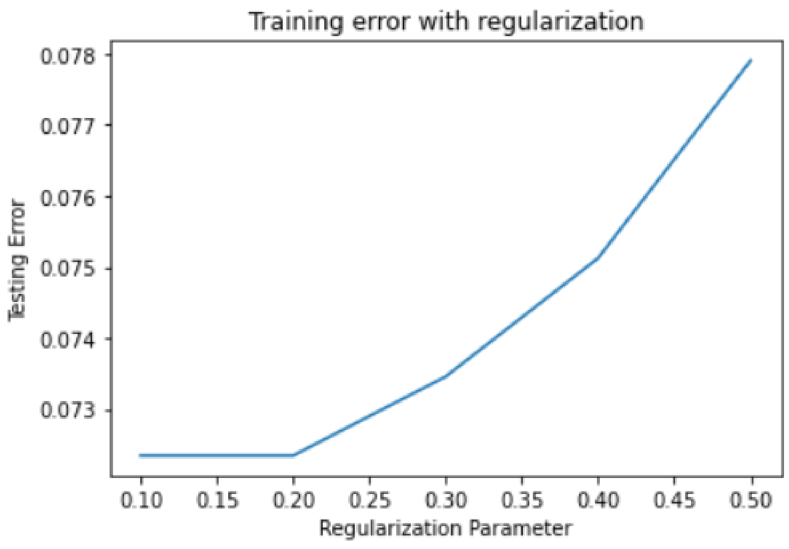
   Testing error: 0.06510851419031716

   (b) Use the logistic regression classifier with regularization so that you also penalize large weights($\lambda||w||_2^2$). Plot the average training and test errors for at least 5 different values of regularization parameter λ.
   Solution.
   Lambda = [0.1, 0.2, 0.3, 0.4, 0.5]

   Training error:
   0.1 = 0.047868166361496156
   0.2 = 0.048914465079780256
   0.3 = 0.05048391315720635
   0.4 = 0.05153021187549045
   0.5 = 0.05283808527334555

   Testing error:
   0.1 = 0.07234279354479689
   0.2 = 0.07234279354479689
   0.3 = 0.07345575959933226
   0.4 = 0.07512520868113526
   0.5 = 0.07790762381747351


Training error with regularization
Training Error
Regularization Parameter
0.048
0.049
0.050
0.051
0.052
0.053
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50


Training error with regularization
Testing Error
Regularization Parameter
0.073
0.074
0.075
0.076
0.077
0.078
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50