EXPERIMENT -1

Aim: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

Description:

- The concept learning approach in machine learning, can be formulated as "Problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples".
- Find-S algorithm for concept learning is one of the most basic algorithms of machine learning.

Algorithm:

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
   For each attribute constraint a i in h :
   If the constraint a i in h is satisfied by x then do nothing
   Else replace a i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

Source code:

```
import io
import csv
import pandas as pd
num_attributes = 6
a = []
print("The given dataset is")
with open('E:/MLDS/First.csv', 'r') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        a.append(row)
        print(row)
print("The initial hypothesis is")
hypothesis = ['0'] * num_attributes
```

```
print (hypothesis)
# Take the first 'yes' raw attributes into hypothesis
for j in range (0, num_attributes):
    hypothesis [j] = a [1] [j]
print (hypothesis)
print (" Finds: Finding maximal specific hypothesis")
for i in range ( 1, len (a)):
    if a[i] [num_attributes] == 'yes' or a[i]
                [num-attributes] == 'yes':
        for j in range (0, num_attributes):
            if a[i][j] != hypothesis[j]:
                hypothesis [j] = '?'
            else:
                hypothesis [j] = a[i][j]
print (" For Training instance NO: {0} the
    hypothesis is". format (i), hypothesis).
```

Output: The given dataset is:

['sky', 'Air temp', 'Humidity', 'Wind', 'water', 'Forecast', 'Enjoy sport']

['sunny', 'warm', 'Normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'High', 'strong', 'warm', 'same', 'yes']
['Sunny', 'cold', 'High', 'strong', 'warm', 'change', 'No']
['sunny', 'warm', 'High', 'Strong', 'cool', 'change', 'yes']

The initial hypothesis is

['0', '0', '0', '0', '0', '0']

['Sunny', 'warm', 'Normal', 'strong', 'warm', 'same']

Find S: Find maximal specific hypothesis.

For Training instance NO: 1 the hypothesis is

['sunny', 'warm', 'Normal', 'strong', 'warm', 'same']

For Training instance NO: 2 the hypothesis is

['sunny', 'warm', '?', 'strong', 'warm', 'same']

For Training instance NO: 3 the hypothesis is

['sunny', 'warm', '?', 'strong', 'warm', 'same']

For Training instance No: 4 the hypothesis is

['sunny', 'warm', '?', 'strong', '?', '?']

Viva Voce Questions

1) What is machine learning?

machine learning is an application of AI. machine learning uses data to train and find accurate results. The capability of a machine to imitate intelligent human behaviour

2) Differentiate between Supervised and Unsupervised learning.

* supervised learning allows you to collect data (or) produce a data output from the previous experience
* unsupervised learning helps you to find all kind of unknown patterns in data.

3) What is hypothesis in Find-s algorithm?

The find-s algorithm finds the most specific hypothesis that fits all the positive examples.

4) Which algorithm can be used for identifying suitable hypothesis?

The find-s algorithm can be used for identifying suitable hypothesis.

5) Why find-s algorithm does not consider negative example?

Find-s algorithm ignore the negative examples as it will give negative end result.

Source code:

```
import os
import csv  # upload
import pandas as pd
df = pd.read.csv("week2.csv")
print(df)
def entropy(probs):
    import math
    return sum(-prob * math.log(prob,2) for prob in probs)
def entropy-of-list(a-list):
    from collections import Counter
    cnt= Counter(x for x in a-list)
    num_instances = len(a.list)
    probs = [x/num_instances for x in cnt-values()]
    return entropy(probs)
total entropy = entropy-of-list(df['play Tennis'])
print('total entropy)
def information_gain(df, split_attribute_name, target_attribute_name,
                     trace=0):
df_split = df.groupby(split-attribute_name)
for name, group in df.split:
    nobs = len(df.index) * 1.0
    df_agg_ent = df_split.agg({target_attribute_name:
                 [entropy_of_list, lambda x: len(x)/nobs]})[target_attribute_name]
    avg_info = sum(df_agg_ent['entropy_of_list'] * df_agg_ent['<lambda_0>'])
    old_entropy = entropy_of_list(df[target_attribute_name])
    return old-entropy-avg_info.
def id3DT(df, target_attribute_name, attribute_names, default_class=None):
    from collections import Counter
    cnt = Counter(x for x in df[target_attribute_name])
```

```
    if len(cnt) == 1:
        return next(iter(cnt))
    elif df.empty or (not attribute_names):
        return default_class
    else:
        default_class = max(cnt.keys())
        gainz = [information_gain(df, attr, target_attribute_name) for attr in
                                                    attribute_names]
        index_of_max = gainz.index(max(gainz))
        best_attr = attribute_names[index_of_max]
        tree = {best_attr: {}}
        remaining_attributes_names = [i for i in attribute_names
                                        if i != best_attr]
        for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3DT(data_subset, target_attribute_name,
                        remaining_attributes_names, default_class)
            tree[best_attr][attr_val] = subtree
        return tree
attribute_names = list(df.columns)
attribute_names.remove('playTennis')

from pprint import pprint
tree = id3DT(df, 'playTennis', attribute_names)
print("The Resultant Decision Tree is")
pprint(tree)
attribute = next(iter(tree))
print("Best Attribute: \n", attribute)
print("Tree keys\n", tree[attribute].keys())

def classify(instance, tree, default=None):
    attribute = next(iter(tree))
    print("key: ", tree.keys())
```

```
print("Attribute", attribute)
if instance[attribute] in tree[attribute].keys():
    result = tree[attribute][instance[attribute]]
    print("Instance Attribute", instance[attribute], "Treekeys:", tree[attribute].keys())
    if isinstance(result,dict):
        return classify(instance, result)
    else:
        return result.
else:
    return default.
tree1={'outlook': ['Rainy', 'Sunny'], 'Temperature':['mild', 'Hot'], 'Humidity': ['Normal', 'high'], 'wind': ['weak', 'strong']}
df2 = pd.DataFrame(tree1)
df2['predicted'] = df2.apply(classify, axis = 1, args=(tree, 'No'))
print(df2)  output:-
```

Day	Outlook	Temp	Humidity	wind	play Tennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	weak	Yes
4	Rain	cool	Normal	weak	Yes
5	Rain	cool	Normal	Strong	NO
6	overcast	cool	Normal	Strong	Yes
7	sunny	mild	High	weak	NO
8	sunny	cool	Normal	weak	Yes
9	Rain	mild	Normal	weak	Yes
10	sunny	mild	Normal	strong	yes
11	overcast	mild	High	strong	Yes
12	overcast	Hot	Normal	weak	yes
13	Rain	mild	High	Strong	No

Entropy:- $\text{Entropy}(S) = \sum_{i=1} p_i \log_2 p_i$

→ $\text{Entropy}(S) = -(p+)\log_2(p+) - (p-)\log_2(p-)$

→ For our look we have 9 "yes" and 5 "No"

$\text{Entropy}(S) = -(9/14)+\log_2(9/14) - (5/14)\log_2(5/14) = 0.940.$

Output:

Temp: S = [9,5] = 0.94 ⇒ -(9/14)·log2(9/14) - (5/14) log2 (5/14)
Shot = [2,2] = 1.0 ⇒ -(2/4) × log2 (2/4) - (2/4) log2 (2/4)
Smild = [4,2] = 0.98 ⇒ -(4/6) log2 (4/6) - (2/6) log2 (2/6)
Scool = [3,1] = 0.811 ⇒ (3/4) log2 (3/4) - (1/4) log2 (1/4)

Humidity:
S = [9,5] = 0.94 ⇒ -(9/14) log2 (9/14) - (5/14) log2 (5/14)
SHigh = [3,4] = ⇒ -(3/7) log2 (3/7) - (4/7) log2 (4/7)
SNormal [6,1] = ⇒ -(6/7) log2 (6/7) - (1/7) log2 (1/7)

Wind:
S = [9,5] = 0.94 ⇒ -(9/14) log2 (9/14) - (5/14) log2 (5/14)
Sstrong [3,3] = 1.0 ⇒ -(3/6) log2 (3/6) - (3/6) log2 (3/6)
Sweak [6,2] = 0.8 ⇒ -(6/8) log2 (6/8) - (6/8) log2 (6/8)

Information Gain:-

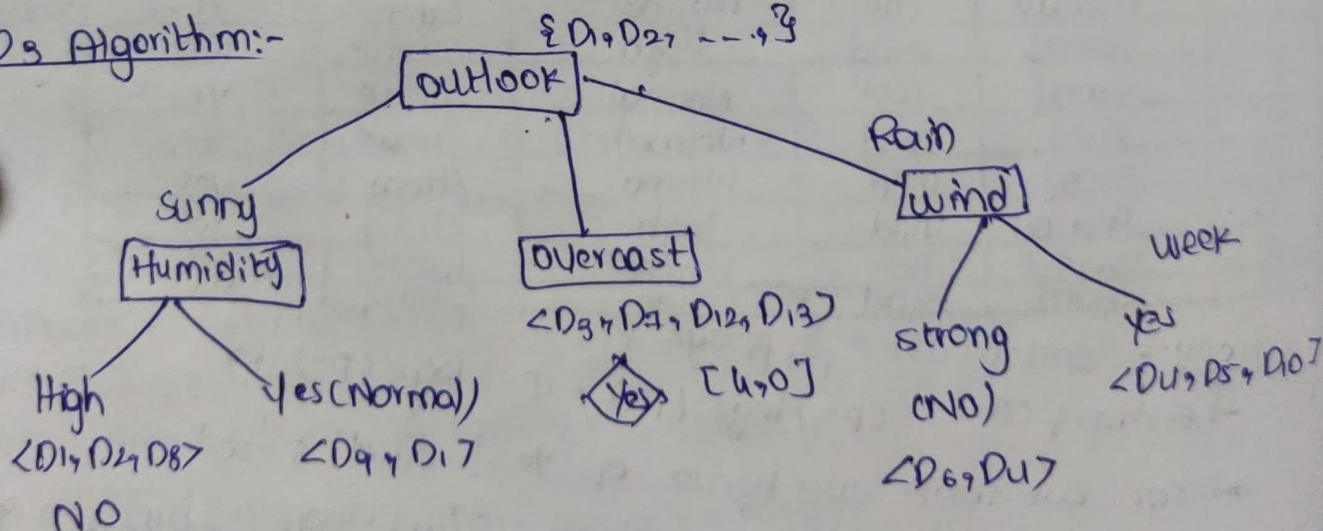
→ Gain (S, outlook) = Entropy (S) - $\sum_{v \in \{sunny, overcast, Rain\}}$ (Sv/S) Entropy.

Gain (S, outlook) =
entropy (S) - 5/14 entropy (S, sunny) - 4/14 Entropy (S, overcast) - 5/14 Entropy (S, rain)

⇒ Gain (S, outlook) = 0.94 - 4/14 (1.0) - 6/14 (0.9183) - 4/14 (0.94

⇒ 0.0289.

ID3 Algorithm:-



Viva Voce Questions

1) Define decision tree.

A decision tree is a type of supervised machine learning used to categorize (or) make predictions based on how a previous set of questions were answered.

2) What is the ID3 algorithm and how do we use it in decision tree regression?

* ID3 used a top-down greedy approach to build a decision tree.
* It can be used to construct decision tree for regression by replacing information gain with standard deviation Reduction.

3) How decision tree is constructed using ID3 algorithm?

The ID3 algorithm is a classic machine learning algorithm used to build a decision tree.
The id3 algorithm uses entropy and information gain to construct tree.

4) What are the advantages of ID3 algorithm?

Simple and easy to understand.
* Fast training time.
* Handles categorical data well.

5) What are the different types of decision trees?

* ID3
* CART (classification and regression tree)
* Random forest
* CHAID.

EXPERIMENT -3

Aim: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Description:

- Given a dataset X, y, we attempt to find a linear model h(x) that minimizes residual sum of squared errors. The solution is given by Normal equations.
- Linear model can only fit a straight line, however, it can be empowered by polynomial features to get more powerful models. Still, we have to decide and fix the number and types of features ahead.
- Alternate approach is given by locally weighted regression.
- Given a dataset X, y, we attempt to find a model h(x) that minimizes residual sum of weighted squared errors.
- The weights are given by a kernel function which can be chosen arbitrarily and in my case I chose a Gaussian kernel.
- The solution is very similar to Normal equations, we only need to insert diagonal weight matrix W.

Algorithm:

```
def local_regression(x0, X, Y, tau):
    # add bias term
    x0 = np.r_[1, x0]
    X = np.c_[np.ones(len(X)), X]
    # fit model: normal equations with kernel
    xw = X.T * radial_kernel(x0, X, tau)
    beta = np.linalg.pinv(xw @ X) @ xw @ Y
    # predict value
    return x0 @ beta
    def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
```

Source code:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
def kernal(point, xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye(m))
    print("WEIGHTS", weights)
    for j in range(m):
```

```
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
        print("WEIGHTS", weights)
    return weights
from numpy.core.memap import hdarray
def localweight(point, xmat, ymat, k):
    print("XMAT", xmat)
    print("YMAT", ymat)
    print("K", k)
    wei = kernal(point, xmat, k)
    w = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    print("W", w)
    return w
def localweightRegression(xmat, ymat, k):
    m,n = np.shape(xmat)
    ypred = np.zeroes(m)
    print("m", m)
    for i in range(m):
        ypred[i] = xmat[i]*localweight(xmat[i], xmat, ymat, k)
        print("ypred", ypred[i])
    return ypred
data = pd.read_csv("hotel-bill.csv")
bill = np.array(data.total_bill)
print("Bill", bill)
tip = np.array(data.tip)
print("Tip", tip)
```

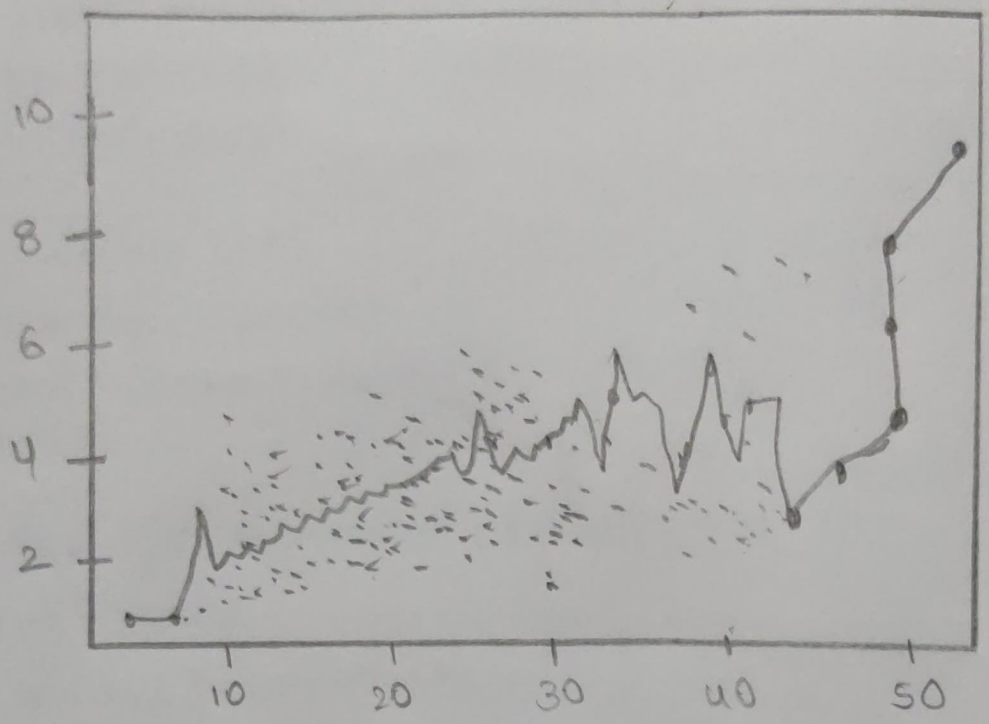
```
mbill = np.mat(bill)
print("mbill", mbill)
mtip = np.mat(tip)
print("mtip", mtip)

m = np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T, mbill.T))
print("X", X)

ypred = localWeightRegression(X, mtip, 0.5)
print("ypred", ypred)
SortIndex = X[:,1].argsort(0)
print("sortindex", SortIndex)
xsort = X[SortIndex][:,0]
print("XSORT", xsort)

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill, tip, color='green')
ax.plot(xsort[:,1], ypred[SortIndex], color='red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

Output:



Viva Voce Questions

1) What is locally weighted regression algorithm?

It is a non-parametric algorithm. The model will not learn a fixed set of parametric as is done in orderly linear regression.

2) What are the advantages of locally weighted regression?

It allows to improve the over all performance of regression methods.

3) What type of learning is used in locally weighted regression?

It is a Supervised learning algorithm is used in locally weighted regression. & it is a non-parametric algorithm.

4) What is logistic regression?

logistic regression is used to calculate (or) predict the prob of binary (Y/N) event occuring.

5) What is weighted regression analysis?

It is a method that you can use when the least square assumption of constant variance are in residual is voilated.

Find the mean closest to the item
Assign item to mean
Update mean

Source code:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
iris = datasets.load_iris()
X=pd.DataFrame(iris.data)
print(X.shape)
X.columns = ['sepal_length','sepal_width','petal_length',
                                    'petal_width']
y= pd.DataFrame(iris.target)
y.columns = ['target']
plt.figure(figsize = (14,7))
colormap= np.array(['red', 'lime', 'black'])
plt.subplot(1,2,1)
plt.scatter(X.sepal_Length,X.sepal_Width,c=colormap[y.target],s=
plt.title('Sepal')
plt.subplot(1,2,2)
plt.scatter(X.petal_length,X.petal_width,c=colormap[y.target],s=40
plt.title('petal')
```

```
model = kmeans (n_clusters = 3)
model . fit(X)
Print (model. labels_)
 plt. subplot (1,2,1)
plt .Scatter ( X. sepal_Length , X. sepal_Width, C=colormap[y. target], s=40)
plt . tittle ('Real classification')
plt . subplot (1,2,2)
plt . scatter (X. Sepal_Length, X. sepal_width, C=colormap[model. labels
                                                    s=40)
plt. tittle ('kMEANS classification')

Print (Sm. accuracy_score (y, model .labels_))
 sm. confusion_matrix (y, model . labels_)

from sklearn. naive_bayes import GaussianNB
clf = GaussianNB()
clf . fit (X, y)
y_cluster_gmm = clf. predict (X)
print (y_cluster_gmm)
plt. subplot (1,2,1)
plt. scatter (X. petal_Length, X. petal_width, C =colormap[y. target], s=40)
plt. title('Real classification)
 plt . subplot (1,2,2)
 plt . scatter (X. petal_Length, X. petal_ width,  C =Color map
                                                [y_cluster_gmm], s =40)
 plt. tittle ('kMEANS classification')
```

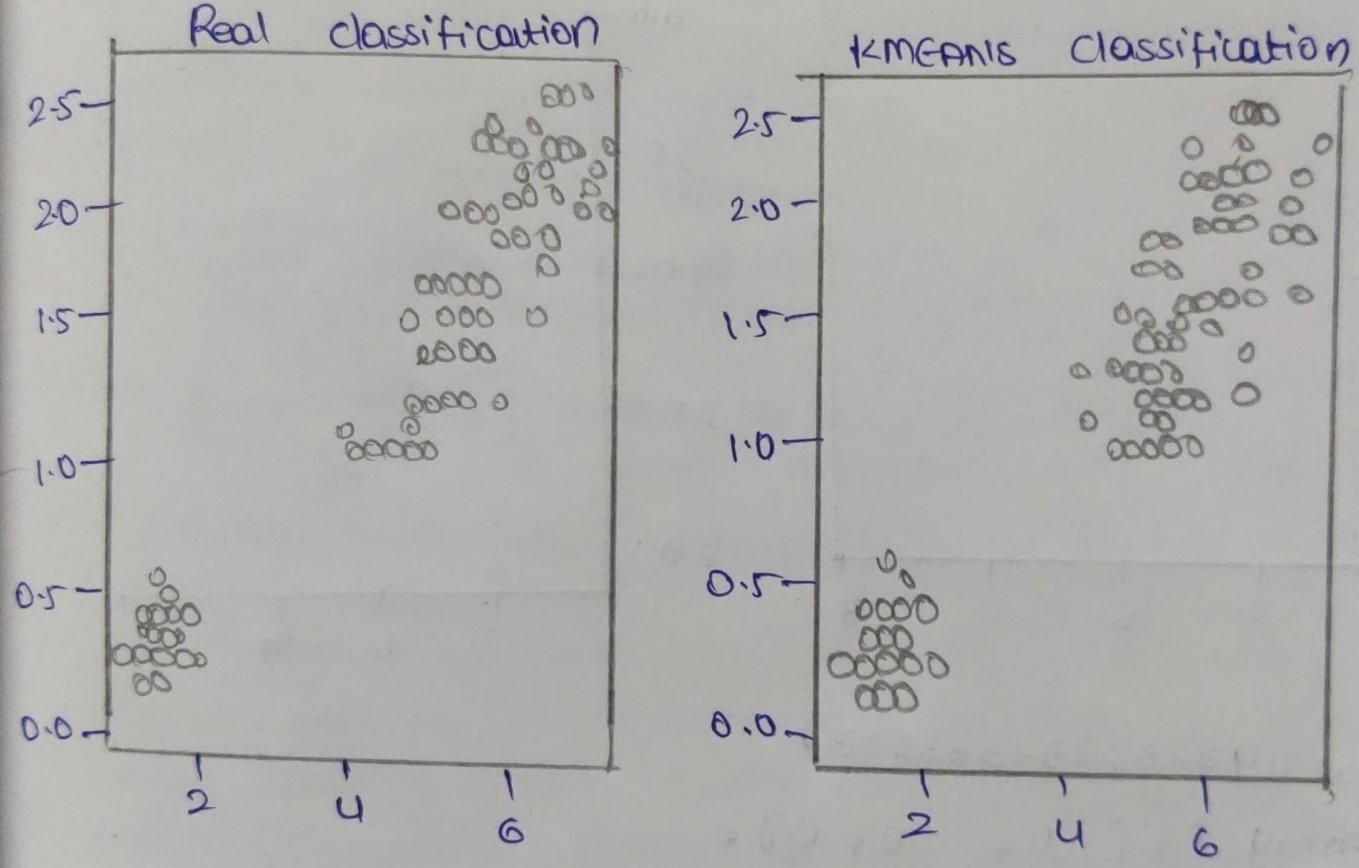
print (sm.accuracy_score (y, y_cluster_gmm))

sm. confusion_matrix (y, y_cluster_gmm)

output:

Gaussian matrix (n_components = 3)

Text (0.5, 1.0, 'KMEANS classification')



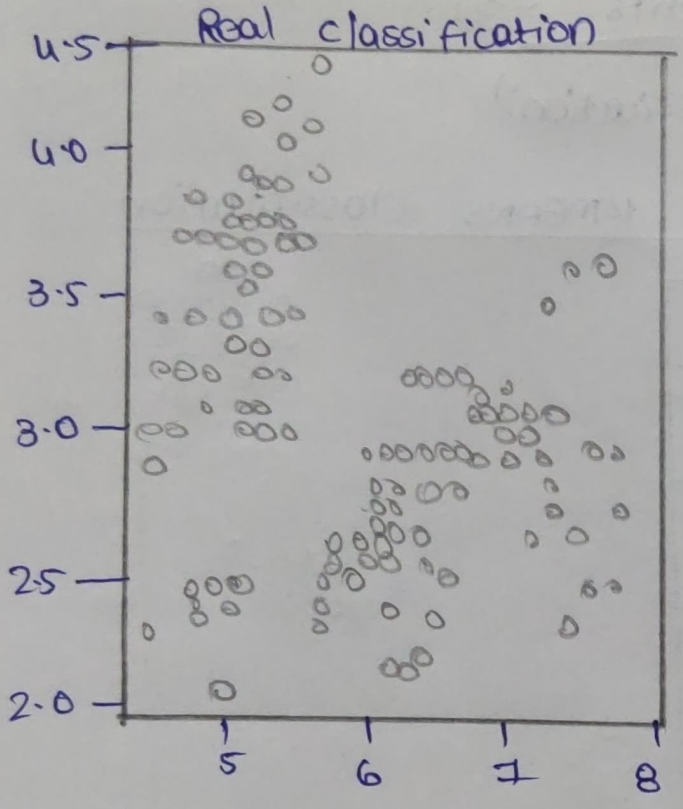
0.96

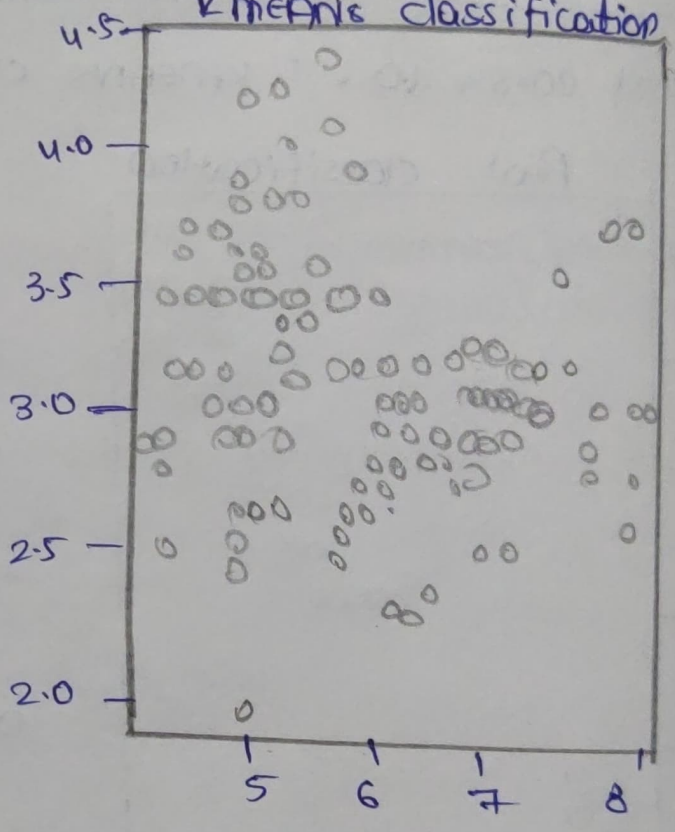
array([[50, 0, 0],
[0, 47, 3],
[0, 3, 47]])

Output:

K-means cn-clusters = 3)

Text (0.5, 1.0, 'KMEANS classification').





0.09333333333333334

array([[0, 50, 0],
[2, 0, 48],
[36, 0, 14]])

Viva Voce Questions

1) What is EM algorithm in machine learning?

Expected maximization algorithm is defined as combo of variou unsupervised machine learning also which used to define local maximum likelihood estimation for unobservable in statistical models.

2) What is K-means clustering algorithm and List out the applications?

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data The goal of this algorithm is to find groups in the data, with the no. of groups represented by the variable K.

Applications:- public data, capturing Images.

3) What are the advantages and disadvantages of K-means clustering?

Advantage:- → Relatively simple to implement
→ scaler to large data sets.

Disadvantages:- choosing K manually, Being depends on initial value

4) What is the difference between K-means and EM?

K-means and Em in the sense that they allow model refining of an iterative process to find the best congestion. The K-means algorithm differs in the method used for calculating the Euclidean distance while calculating the distance blw of two data items; and Em uses statistical methods.

5) What are the Applications of EM algorithm?

* chosen in unsupervised data clustering and Psychiatic analysis
* it has numerious uses in NCPs, complex visual quantive analysis of gentrics.