KRUTHIK B ML CODES

1)Find S Algorithm

In [29]:

```
import csv
with open('general1.csv','r') as f:
    reader=csv.reader(f)
    data=list(reader)
print("Training data is : \n")
for row in data:
    print(row)
att_len=len(data[0])-1
h=['0']*att_len
print("Hypothesis H initially is :",h)
k=0
print("Hypothesis Finally are : \n")
for row in data:
    if row[-1]=='yes':
        j=0
        for col in row:
            if col!='yes':
                if col!=h[j] and h[j]=='0':
                    h[j]=col
                elif col!=h[j] and h[j]!='0':
                    h[j]='?'
            j+=1
    print("Hypothesis",k,"=",h,)
    k+=1
print("Maximally Specific Hypothesis :","h",k-1,"=",h)
```

```
Training data is :

['sunny', 'hot', 'high', 'weak', 'no']
['sunny', 'hot', 'high', 'strong', 'no']
['overcast', 'hot', 'high', 'weak', 'yes']
['rain', 'mild', 'high', 'weak', 'yes']
Hypothesis H initially is : ['0', '0', '0', '0']
Hypothesis Finally are :

Hypothesis 0 = ['0', '0', '0', '0']
Hypothesis 1 = ['0', '0', '0', '0']
Hypothesis 2 = ['overcast', 'hot', 'high', 'weak']
Hypothesis 3 = ['?', '?', 'high', 'weak']
Maximally Specific Hypothesis : h 3 = ['?', '?', 'high', 'weak']
```

2)Candidate Elimination Algorithm

In [31]:

```
import csv
with open('general1.csv','r') as f:
    reader=csv.reader(f)
    data=list(reader)
print("Training data is : \n")
for row in data:
    print(row)
print('-'*50)
att_len=len(data[0])-1
h=['0']*att_len
g=['?']*att_len
temp=[]
print("The Hypothesis are :")
print("More Specific :",h)
print("More Generic :",g)
print('-'*50)
for row in data:
    if row[-1]=="yes":
        j=0
        for col in row:
            if col!="yes":
                if col!=h[j] and h[j]=="0":
                    h[j]=col
                elif col!=h[j] and h[j]!="0":
                    h[j]='?'
            j+=1
        for j in range(0,att_len):
            for k in temp:
                if k[j]!=h[j] and k[j]!='?':
                    temp.remove(k)
    if row[-1]=="no":
        j=0
        for col in row:
            if col!='no':
                if col!=h[j] and h[j]!="?":
                    g[j]=h[j]
                    temp.append(g)
                    g=['?']*att_len
            j+=1
    print("Most Specific Hypothesis is :",h)
    if len(temp)==0:
        print("Most General Hypothesis are :",g)
    else:
        print("Most General Hypothesis are :",temp)
    print('-'*50)
```

```
Training data is :

['sunny', 'hot', 'high', 'weak', 'no']
['sunny', 'hot', 'high', 'strong', 'no']
['overcast', 'hot', 'high', 'weak', 'yes']
['rain', 'mild', 'high', 'weak', 'yes']
--------------------------------------------------
The Hypothesis are :
More Specific : ['0', '0', '0', '0']
More Generic : ['?', '?', '?', '?']
--------------------------------------------------
Most Specific Hypothesis is : ['0', '0', '0', '0']
```

```
Most General Hypothesis are : [['0', '?', '?', '?'], ['?', '0', '?', '?'],
['?', '?', '0', '?'], ['?', '?', '?', '0']]
-------------------------------------------------
Most Specific Hypothesis is : ['0', '0', '0', '0']
Most General Hypothesis are : [['0', '?', '?', '?'], ['?', '0', '?', '?'],
['?', '?', '0', '?'], ['?', '?', '?', '0'], ['0', '?', '?', '?'], ['?', '0',
'?', '?'], ['?', '?', '0', '?'], ['?', '?', '?', '0']]
-------------------------------------------------
Most Specific Hypothesis is : ['overcast', 'hot', 'high', 'weak']
Most General Hypothesis are : [['?', '?', '?', '0']]
-------------------------------------------------
Most Specific Hypothesis is : ['?', '?', 'high', 'weak']
Most General Hypothesis are : ['?', '?', '?', '?']
-------------------------------------------------
```

9)K Neigbors Classification

In [3]:

```
from sklearn import datasets
iris=datasets.load_iris()
iris_data=iris.data
iris_labels=iris.target
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(iris_data,iris_labels,test_size=0.3)
from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)
target_names=iris.target_names
for pred, actual in zip(y_pred,y_test):
    print("Prediction is "+ str(target_names[pred]) + " Actual is " + str(target_names[actu
```

```
Prediction is versicolor Actual is versicolor
Prediction is virginica Actual is virginica
Prediction is setosa Actual is setosa
Prediction is virginica Actual is virginica
Prediction is setosa Actual is setosa
Prediction is versicolor Actual is versicolor
Prediction is setosa Actual is setosa
Prediction is versicolor Actual is versicolor
Prediction is versicolor Actual is versicolor
Prediction is versicolor Actual is versicolor
Prediction is versicolor Actual is versicolor
Prediction is setosa Actual is setosa
Prediction is setosa Actual is setosa
Prediction is setosa Actual is setosa
Prediction is setosa Actual is setosa
Prediction is versicolor Actual is versicolor
Prediction is versicolor Actual is versicolor
Prediction is virginica Actual is virginica
Prediction is virginica Actual is virginica
Prediction is versicolor Actual is versicolor
Prediction is setosa Actual is setosa
Prediction is setosa Actual is setosa
Prediction is versicolor Actual is versicolor
Prediction is setosa Actual is setosa
Prediction is setosa Actual is setosa
Prediction is setosa Actual is setosa
Prediction is setosa Actual is setosa
Prediction is versicolor Actual is versicolor
Prediction is setosa Actual is setosa
Prediction is setosa Actual is setosa
Prediction is virginica Actual is virginica
Prediction is setosa Actual is setosa
Prediction is versicolor Actual is versicolor
Prediction is virginica Actual is versicolor
Prediction is setosa Actual is setosa
Prediction is versicolor Actual is versicolor
Prediction is setosa Actual is setosa
Prediction is virginica Actual is virginica
Prediction is versicolor Actual is versicolor
Prediction is setosa Actual is setosa
Prediction is virginica Actual is virginica
Prediction is virginica Actual is virginica
Prediction is versicolor Actual is versicolor
```

```
Prediction is setosa Actual is setosa
Prediction is virginica Actual is virginica
```

8)Expecation-Maximization and KMeans Algorithm and Comaprison

In [11]:

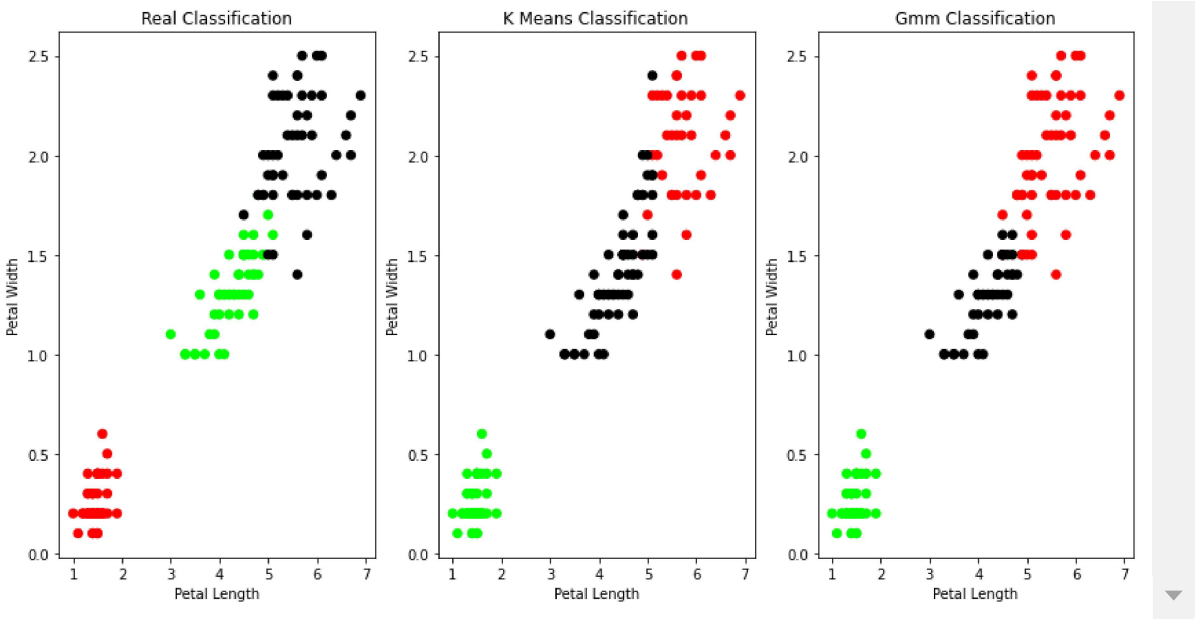
```
import matplotlib.pyplot as plt
from sklearn import datasets
import sklearn.metrics as sm
import pandas as pd
import numpy as np
iris=datasets.load_iris()
X=pd.DataFrame(iris.data)
X.columns=['Sepal_Length','Speal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(iris.target)
y.columns=['Targets']

#original
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title("Real Classification")
plt.xlabel("Petal Length")
plt.ylabel("Petal Width")

#Kmeans
from sklearn.cluster import KMeans
model=KMeans(n_clusters=3)
model.fit(X)
y_km=model.predict(X)
plt.subplot(1,3,2)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_km],s=40)
plt.title('K Means Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('Accuracy Score of K-Mean:',sm.accuracy_score(y,y_km))
print('Confusion Matrix of K-Mean:',sm.confusion_matrix(y,y_km))

#Em algorithm
from sklearn.mixture import GaussianMixture
gmm=GaussianMixture(n_components=3)
gmm.fit(X)
y_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_gmm],s=40)
plt.title('Gmm Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('Accuracy Score of EM:',sm.accuracy_score(y,y_gmm))
print('Confusion Matrix of EM:',sm.confusion_matrix(y,y_gmm))
```

```
Accuracy Score of K-Mean: 0.09333333333333334
Confusion Matrix of K-Mean: [[ 0 50  0]
 [ 2  0 48]
 [36  0 14]]
Accuracy Score of EM: 0.0
Confusion Matrix of EM: [[ 0 50  0]
 [ 5  0 45]
 [50  0  0]]
```



6)Text Classification Using Naive Bayes

In [38]:

```
import pandas as pd
msg=pd.read_csv("data.csv",names=['message','label'])
print("Total Instances in dataset are:",msg.shape[0])
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
#print("Dataset :") print(msg)
x=msg.message
y=msg.labelnum

from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y)
print("Total training and testing instances are :")
print(ytrain.shape[0])
print(ytest.shape[0])

from sklearn.feature_extraction.text import CountVectorizer
count_vect=CountVectorizer()
xtrain_dtm=count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print("Total features extraced are :",xtrain_dtm.shape[1])
print("Features are ---->")
print(count_vect.get_feature_names())

from sklearn.naive_bayes import MultinomialNB
clf=MultinomialNB()
clf.fit(xtrain_dtm,ytrain)
y_pred=clf.predict(xtest_dtm)
print("Predicted and Actual Data are :")
print(y_pred)
print(list(ytest))

from sklearn import metrics
print('-'*40)
print("Accuracy Score :",metrics.accuracy_score(ytest,y_pred))
print("Reacall :",metrics.recall_score(ytest,y_pred))
print("Precision :",metrics.precision_score(ytest,y_pred))
print("Confusion Matrix is:",metrics.accuracy_score(ytest,y_pred))
```

```
Total Instances in dataset are: 18
Total training and testing instances are :
13
5
Total features extraced are : 50
Features are ---->
['about', 'am', 'amazing', 'an', 'and', 'awesome', 'bad', 'beers', 'best',
'boss', 'can', 'dance', 'deal', 'do', 'enemy', 'feel', 'fun', 'good', 'grea
t', 'have', 'holiday', 'horrible', 'house', 'is', 'juice', 'like', 'localit
y', 'love', 'my', 'not', 'of', 'place', 'sick', 'stay', 'taste', 'that', 'th
e', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'we', 'wen
t', 'what', 'will', 'with', 'work']
Predicted and Actual Data are :
[1 1 0 0 0]
[1, 1, 0, 0, 0]
----------------------------------------
Accuracy Score : 1.0
Reacall : 1.0
Precision : 1.0
Confusion Matrix is: 1.0

c:\python\python385\lib\site-packages\sklearn\utils\deprecation.py:87: Fut
```

```
ureWarning: Function get_feature_names is deprecated; get_feature_names is
deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names
_out instead.
  warnings.warn(msg, category=FutureWarning)
```

4)Artificial Neural Networks Implementation Using Backpropogation

In [27]:

```
import numpy as np
X=np.array(([2,9],[1,5],[3,6]),dtype=float)
y=np.array(([92],[86],[89]),dtype=float)
X=X/np.amax(X,axis=0)
y=y/100
def sigmoid(x):
    return 1/(1+np.exp(-x))
def inverse_sigmoid(x):
    return x*(1-x)
epoch=5
lr=0.1
input_neuron=2
hidden_neuron=3
output_neuron=1
wh=np.random.uniform(size=(input_neuron,hidden_neuron))
bh=np.random.uniform(size=(1,hidden_neuron))
wout=np.random.uniform(size=(hidden_neuron,output_neuron))
bout=np.random.uniform(size=(1,output_neuron))
for i in range(epoch):
    hinp1=np.dot(X,wh) #inputs * weights to hidden layer
    hinp=hinp1+bh #addign bias
    hlayer=sigmoid(hinp) #activation function at hidden layer

    outinp1=np.dot(hlayer,wout) #inputs at hidden * weights to output layer
    outinp=outinp1+bout #adding bias
    output=sigmoid(outinp) #activation function at output layer

    EO=y-output #error calculation after forward propogation

    delta_out=EO*inverse_sigmoid(output)
    Error_H=delta_out.dot(wout.T) #correcting weights at hidden layer(hidden->output)
    delta_inp=Error_H*inverse_sigmoid(hlayer) #correcting weights at input layer(input->hid
    #wh bh wout bout updating them for next iteration(epochs)
    wh += X.T.dot(delta_inp)*lr
    wout += hlayer.T.dot(delta_out)*lr
    bh += np.sum(delta_inp,axis=0,keepdims=True)*lr
    bout += np.sum(delta_out,axis=0,keepdims=True)*lr


print("Input \n",X)
print("Actual Output and predicted Outputs are :")
print(y)
print('-'*40)
print(output)
print('-'*40)
print("Error is : \n",EO)

#bias bh and bout not necessarily required if we take then out also no issues
```

```
Input
 [[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output and predicted Outputs are :
[[0.92]
 [0.86]
 [0.89]]
----------------------------------------
```

```
[[0.87937305]
 [0.86033926]
 [0.87743124]]
----------------------------------------
Error is :
 [[ 0.04062695]
 [-0.00033926]
 [ 0.01256876]]
```

5)Naive Bayes Classifier

In [111]:

```
#part1
import numpy as np
import pandas as pd
import csv
dataset=pd.read_csv("general.csv")
x=dataset.iloc[:,:-1].values #data
y=dataset.iloc[:,-1].values #outcome

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
print("Training Dataset size is:",x_train.shape[0])
print("Testing Dataset size is:",x_test.shape[0])
print('-'*40)
countyes=countno=0
size=y_train.shape[0]
for x in range(y_train.shape[0]):
    if y_train[x]=="yes":
        countyes+=1
    else:
        countno+=1
probYes=countyes/size
probNo=countno/size
print("Target \t Count \t Probolaity")
print("Yes \t", countyes,"\t", probYes)
print("No \t", countno,"\t", probNo)
print('-'*40)
```

```
Training Dataset size is: 9
Testing Dataset size is: 5
----------------------------------------
Target   Count   Probolaity
Yes      5       0.5555555555555556
No       4       0.4444444444444444
----------------------------------------
```

In [128]:

```
#part2
import numpy as np
import pandas as pd
import csv

dataset=pd.read_csv("general.csv")
data=dataset.iloc[0:9].values
test=dataset.iloc[9:].values
prob0=np.zeros((test.shape[1]-1))
prob1=np.zeros((test.shape[1]-1))
accuracy=0
print("Instance \t Prediction \t Target")
for t in range(test.shape[0]):
    for k in range (test.shape[1]-1):
        count1=count0=0
        for j in range(data.shape[0]):
            #how many times appeared with no
            if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='no':
                count0+=1
            #how many times appeared with yes
            if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='yes':
                count1+=1
        prob0[k]=count0/countno
        prob1[k]=count1/countyes
    probno=probNo
    probyes=probYes
    for i in range(test.shape[1]-1):
        probno=probno*prob0[i]
        probyes=probyes*prob1[i]
    if probno>probyes:
        predict='no'
    else:
        predict='yes'
    print("   ",t+1," \t\t  ",predict,"\t\t ",test[t,test.shape[1]-1])
    if predict==test[t,test.shape[1]-1]:
        accuracy+=1
finalaccuracy=(accuracy/test.shape[0])*100
print("Accuracy=",finalaccuracy,"%")
```

```
Instance         Prediction      Target
   1               yes             yes
   2               no              yes
   3               yes             yes
   4               yes             yes
   5               no              no
Accuracy= 80.0 %
```