LAB PROGRAM 1

```
import csv
with open("C:/Users/dhira/ML lab/enjoysport1.csv","r") as
f:
    reader=csv.reader(f)
    data=list(reader)
print("Training Data")
for row in data:
    print(row)
attr_len=len(data[0])-1
h=['0']*attr_len
print("h= ",h)
k=0
print("The Hypothesis are")
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=j+1
    print("h",k,"=",h)
    k=k+1
print("MAximally SPecific Hypothesis :\n","h",k-1,"=",h)
```

LAB PROGRAM 2

#Candidate Elimination Algorithm

```
import csv
with open("C:/Users/dhira/ML lab/enjoysport1.csv","r") as
f :
    reader = csv.reader(f)
    data = list(reader)

#Training data from CSV file
print("Training Data")
for row in data:
    print(row)
print("-----------------------------------")

attr_len = len(data[0])-1
#initialize specific and general hypothesis
S = ['0']*attr_len
G = ['?']*attr_len
temp = [] #altered G

print("The Hypothesis are")
print("S = ",S)
print("G =", G)
print("-----------------------------------")

for row in data:
    if row[-1] == "Yes":
        j=0
```

```
    for col in row:
        if col != "Yes":
            if col != S[j] and S[j] == '0':
                S[j] = col
            elif col != S[j] and S[j] != '0':
                S[j] = '?'

        j = j+1



    for j in range (0, attr_len):
        for k in temp:
            if k[j] != S[j] and k[j] != '?':
                temp.remove(k)

if row[-1] == "No":
    j = 0
    for col in row:
        if col in row:
            if col != "No":
                if col != S[j] and S[j] != '?':
                    if S[j] == '0':
                        G[j] = col
                    else:
                        G[j] = S[j]
                    temp.append(G)
                    G = ['?'] * attr_len
            j = j+1

print("S =",S)
```

```
    if len(temp)==0:
        print("G=",G)
    else:
        print("G=",temp)

    print("-----------------------------------------")
```

LAB PROGRAM 3

```
import pandas as pd
df=pd.read_csv("C:/1DA19CS045/id3.csv")
print("\n Input Data Set is :\n",df)
t=df.keys()[-1]
print("Target Attribute is :",t)
attribute_names=list(df.keys())
attribute_names.remove(t)
print("Predicting Attributes :", attribute_names)
import math
def entropy(probs):
    return sum([-prob*math.log(prob,2) for prob in probs])
def entropy_of_list(ls,value):
    from collections import Counter
    cnt=Counter(x for x in ls)
    print("Target Attribute class count(Yes/No)",dict(cnt))
    total_instances=len(ls)
    print("Total no. of instances/records associated with {0}
is : {1}".format(value,total_instances))
```

```
    probs=[x/total_instances for x in cnt.values()]
    return entropy(probs)
def
information_gain(df,split_attribute,target_attribute,battr):
    print("\n\n--------Information Gain Calculation
of",split_attribute,"---------")
    df_split=df.groupby(split_attribute)
    glist=[]
    for gname,group in df_split:
        print("Ground Attribute values\n",group)
        glist.append(gname)
    nobs=len(df.index)
    df_agg1=df_split.agg({target_attribute : lambda x :
entropy_of_list(x,glist.pop())})
    df_agg2=df_split.agg({target_attribute : lambda x :
len(x)/nobs})
    df_agg1.columns=['Entropy']
    df_agg2.columns=['Proportion']

new_entropy=sum(df_agg1['Entropy']*df_agg2['Proportion'
])
    if battr!='S':

old_entropy=entropy_of_list(df[target_attribute],'S-'+df.iloc[
0][df.columns.get_loc(battr)])
    else:
        old_entropy=entropy_of_list(df[target_attribute],battr)
    return old_entropy-new_entropy
def
id3(df,target_attribute,attribute_names,default_class=Non
```

```
e,default_attr='S'):
    from collections import Counter
    cnt=Counter(x for x in df[target_attribute])
    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=[]
        for attr in attribute_names:

ig=information_gain(df,attr,target_attribute,default_attr)
            gainz.append(ig)
            print("Information gain of ",attr, "is :",ig)
        index_of_max=gainz.index(max(gainz))
        best_attr=attribute_names[index_of_max]
        print("\nAttribute with the maximum gain
is :",best_attr)
        tree={best_attr:{}}
        remaining_attribute_names=[i for i in attribute_names
if i!=best_attr]
        for attr_val,data_subset in df.groupby(best_attr):

subtree=id3(data_subset,target_attribute,remaining_attrib
ute_names,default_class,best_attr)
            tree[best_attr][attr_val]=subtree
        return tree
from pprint import pprint
tree=id3(df,t,attribute_names)
```

```
print("\nThe Resultant Decision Tree is :")
pprint(tree)
```

LAB PROGRAM 4

```
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 derivates_sigmoid(x):
    return x*(1-x)
epoch=5
lr=0.1
inputlayer_neurons=2
hiddenlayer_neurons=3
output_neurons=1
wh=np.random.uniform(size=(inputlayer_neurons,hiddenla
yer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,outp
ut_neurons))
bout=np.random.uniform(size=(1,output_neurons))
for i in range(epoch):
    hinp1=np.dot(x,wh)
    hinp=hinp1+bh
    hlayer_act=sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
```

```
    outinp=outinp1+bout
    output=sigmoid(outinp)
    EO=y-output
    outgrad=derivates_sigmoid(output)
    d_output=EO*outgrad
    EH=d_output.dot(wout.T)
    hiddengrad=derivates_sigmoid(hlayer_act)
    d_hiddenlayer=EH*hiddengrad
    wout+=hlayer_act.T.dot(d_output)*lr
    bout+=np.sum(d_output,axis=0,keepdims=True)*lr
    wh+=x.T.dot(d_hiddenlayer)*lr
    wh+=np.sum(d_hiddenlayer,axis=0,keepdims=True)*lr
print("Input :\n",x)
print("Actual Output :\n",y)
print("Predicted Output :\n",output)
```

LAB PROGRAM 5

```
import numpy as np
import math
import csv
def read_data(filename):
    with open(filename,"r") as csvfile:
        datareader= csv.reader(csvfile)
        traindata=list(datareader)
    metadata=traindata[0]
    traindata=traindata[1:]
    return (metadata,traindata)
def splitdataset(dataset,splitratio):
    trainsize=int(len(dataset)*splitratio)
```

```
    trainset=[]
    testset=list(dataset)
    test=list(dataset)
    i=0
    while len(trainset)<trainsize:
        trainset.append(testset.pop(i))
    return [trainset,testset]
def classify(data,test):
    totalsize=data.shape[0]
    print("\n")
    print("Training data size = ",totalsize)
    print("Test data size = ",test.shape[0])
    countyes=0
    countno=0
    probyes=0
    probno=0
    print("\n")
    print("Target \t count \t probability")
    for x  in range(data.shape[0]):
        if data[x,data.shape[1]-1]=='yes':
            countyes+=1
        if data[x,data.shape[1]-1]=='no':
            countno+=1
    probYes=countyes/totalsize
    probNo=countno/totalsize
    print("Yes \t",countyes,"\t",probYes)
    print("No \t",countno,"\t",probNo)
    prob0=np.zeros((test.shape[1]-1))
    prob1=np.zeros((test.shape[1]-1))
    accuracy=0
```

```
    print("\n")
    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]):
                if test[t,k]==data[j,k] and
data[j,data.shape[1]-1]=='no':
                    count0+=1
                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("\nAccuracy = ",finalaccuracy,"%")
metadata,traindata=read_data("id3.csv")
print("\n The attribute name of training data are  :
```

",metadata)
splitratio=0.7
trainset,testset=splitdataset(traindata,splitratio)
training=np.array(trainset)
print("\nThe Training data set are :")
for x in training :
    print(x)
testing =np.array(testset)
print("\nThe test data set are:")
for x in testing:
    print(x)
classify(training,testing)

LAB PROGRAM 6

import pandas as pd
msg=pd.read_csv("C:/1DA19CS045/
data6.csv",names=['message','label'])
print("Total instances in the dataset :",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("\nDataset is split into Training and Testing Samples")
print("\nTotal Training instances:",ytrain.shape[0])
print("\nTotal Testing instances:",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("\nTotal features extracted using Count
Vectorizer:",xtrain_dtm.shape[1])
print("\nThe words or Tokens in the text document\n")
print(count_vect.get_feature_names())
from sklearn.naive_bayes import MultinomialNB
clf=MultinomialNB().fit(xtrain_dtm,ytrain)
predicted=clf.predict(xtest_dtm)
print("Predicted")
print("==========")
print(predicted)
print("Actual")
print("==========")
print(list(ytest))
from sklearn import metrics
print("\nAccuracy metrics")
print("==========")
print("Accuracy of the classifier
is",metrics.accuracy_score(ytest,predicted))
print("Recall:",metrics.recall_score(ytest,predicted),"\nPrec
ision:",metrics.precision_score(ytest,predicted))
print("Confusion matrix")
print("==========")
print(metrics.confusion_matrix(ytest,predicted))
```

LAB PROGRAM 8

```
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','Sepal_Width','Petal_Length','P
etal_Width']
y=pd.DataFrame(iris.target)
y.columns=['Targets']
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.Tar
gets],s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
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 Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
print('The accuracy score of K-
Mean :',sm.accuracy_score(y,y_km))
```

```
print('The Confusion matrix of K-Mean :
\n',sm.confusion_matrix(y,y_km))
from sklearn.mixture import GaussianMixture
gmm=GaussianMixture(n_components=3)
gmm.fit(x)
y_gmm=gmm.predict(x)
plt.subplot(1,3,2)
plt.scatter(x.Petal_Length,x.Petal_Width,c=colormap[y_gm
m],s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
print('The accuracy score of
EM :',sm.accuracy_score(y,y_gmm))
print('The confusion matrix of EM :
\n',sm.confusion_matrix(y,y_gmm))
```

LAB PROGRAM 9

```
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_l
abels,test_size=0.30)
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[actual]))
from sklearn.metrics import
classification_report,confusion_matrix
print("Confusion matrix is as follows")
print(confusion_matrix(y_test,y_pred))
print("Accuracy matrics")
print(classification_report(y_test,y_pred))
```