MACHINE LEARNING

1. **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**

```
import csv

a = []
print("\n The Given Training Data Set \n")
with open('1_lab.csv', 'r') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        a.append(row)
        print(row)

num_attributes = len(a[0]) - 1
print("\n The initial value of hypothesis: ")
hypothesis = ['0'] * num_attributes
print(hypothesis)

for j in range(0, num_attributes):
    hypothesis[j] = a[0][j];
print("\n Find S: Finding a Maximally Specific Hypothesis\n")
for i in range(0, len(a)):
    if 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)
print("\n The Maximally Specific Hypothesis for a given Training Examples :\n")
print(hypothesis)
```

CSV File : 1_lab.csv

Sunny,Warm,Normal,Strong,Warm,Same,Yes
Sunny,Warm,High,Strong,Warm,Same,Yes
Rainy,Cold,High,Strong,Warm,Change,No
Sunny,Warm,High,Strong,Cool,Change,Yes

Output:
The Given Training Data Set

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']

Find S: Finding a Maximally Specific Hypothesis

For Training instance No:0 the hypothesis is ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
For Training instance No:1 the hypothesis is ['Sunny', 'Warm', '?', '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', '?', '?']

The Maximally Specific Hypothesis for a given Training Examples :

['Sunny', 'Warm', '?', 'Strong', '?', '?']

2. **For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.**

```
import numpy as np
import pandas as pd

data = pd.read_csv('1_2_lab.csv')
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
def learn(concepts, target):
    specific_h = ["0" for i in range(len(concepts[0]))]
    print("initialization of specific_h \n",specific_h)
    specific_h = concepts[0].copy()
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("initialization of general_h \n", general_h)

    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            print("If instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "No":
            print("If instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print(" step {}".format(i+1))
        print(specific_h)
        print(general_h)
        print("\n")
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?',
'?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

CSV File : 1_2_lab.csv

```
Sky,Air Temp,Humidity,Wind,Water,Forecast,Enjoy Sport
Sunny,Warm,Normal,Strong,Warm,Same,Yes
Sunny,Warm,High,Strong,Warm,Same,Yes
Rainy,Cold,High,Strong,Warm,Change,No
Sunny,Warm,High,Strong,Cool,Change,Yes
```

3. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
import numpy as np
import pandas as pd

PlayTennis = pd.read_csv("tennis.csv")

from sklearn.preprocessing import LabelEncoder

Le = LabelEncoder()
PlayTennis['outlook'] = Le.fit_transform(PlayTennis['outlook'])
PlayTennis['temp'] = Le.fit_transform(PlayTennis['temp'])
PlayTennis['humidity'] = Le.fit_transform(PlayTennis['humidity'])
PlayTennis['play'] = Le.fit_transform(PlayTennis['play'])
print(PlayTennis)

y=PlayTennis['play']
X=PlayTennis.drop(['play'],axis=1)

from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion='entropy')
clf=clf.fit(X,y)

X_pred = clf.predict(X)

from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(y,X_pred))
print(classification_report(y,X_pred))
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
plt.figure()
tree.plot_tree(clf,fontsize=6)
plt.savefig('tree.jpg', format='jpg',bbox_inches='tight')
plt.show()
```

CSV File : tennis.csv

```
outlook,temp,humidity,play
TRUE,hot,high,no
TRUE,hot,high,no
FALSE,hot,high,yes
FALSE,cool,normal,yes
FALSE,cool,normal,yes
TRUE,cool,high,no
TRUE,hot,high,no
TRUE,hot,normal,yes
FALSE,cool,normal,yes
FALSE,cool,high,yes
```

4. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

```
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 sigmoid_grad(x):

        return x * (1 - x)


epoch = 1000

eta = 0.2

input_neurons = 2

hidden_neurons = 3

output_neurons = 1


 wh = np.random.uniform(size=(input_neurons, hidden_neurons))

bh = np.random.uniform(size=(1, hidden_neurons))

wout = np.random.uniform(size=(hidden_neurons, output_neurons))

bout = np.random.uniform(size=(1, output_neurons))


for i in range(epoch):

        h_ip = np.dot(X, wh)+bh

        h_act = sigmoid(h_ip)
```

```
        h_act = sigmoid(h_ip)
        o_ip = np.dot(h_act, wout)+bout
        output = sigmoid(o_ip)

        Eo = y - output
        outgrad = sigmoid_grad(output)
        d_output = Eo* outgrad

        Eh = d_output.dot(wout.T)
        hiddengrad = sigmoid_grad (h_act)
        d_hidden = Eh * hiddengrad
        wout += h_act.T.dot(d_output) * eta
        wh += X.T.dot(d_hidden) * eta

print(" Normalized Input:\n" + str(X))
print("Actual Output:\n" + str(y))
print("Predicted Output:\n" ,output)
```

5. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a . CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
import pandas as pd
import numpy as np
dataset = pd.read_csv('5.csv')
x = np.array(dataset.iloc[:, :-1])
y = np.array(dataset.iloc[:, -1])
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.5, random_state=1)
model = GaussianNB()
model.fit(xtrain, ytrain)
predicted = model.predict(xtest)
print('\nConfusion Matrix:')
print(metrics.confusion_matrix(ytest, predicted))
print('\nAccuracy of the classifier:')
print(metrics.accuracy_score(ytest, predicted))
print('\nPrecision:')
print(metrics.precision_score(ytest, predicted, average='weighted'))  # Use 'weighted' for multi-class
print('\nRecall:')
print(metrics.recall_score(ytest, predicted, average='weighted'))  # Use 'weighted' for multi-class
```

CSV File : `5_data_diabetes.csv`

```
Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,DiabetesPedigreeFunction,
Age,Outcome
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1
5,116,74,0,0,25.6,0.201,30,0
3,78,50,32,88,31,0.248,26,1
10,115,0,0,0,35.3,0.134,29,0
2,197,70,45,543,30.5,0.158,53,1
8,125,96,0,0,0,0.232,54,1
4,110,92,0,0,37.6,0.191,30,0
10,168,74,0,0,38,0.537,34,1
10,139,80,0,0,27.1,1.441,57,0
1,189,60,23,846,30.1,0.398,59,1
```

6. Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn import metrics

msg=pd.read_csv('naive.csv',header=None,names=['message','label'])

print("The dimensions of the dataset",msg.shape)

msg['labelnum']=msg.label.map({'pos':1,'neg':0})

x=msg.message

y=msg.labelnum

xtrain,xtest,ytrain,ytest=train_test_split(x,y,random_state=1)

count_vect=CountVectorizer() xtrain_dtm=count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)

clf=MultinomialNB().fit(xtrain_dtm,ytrain)

 predicted=clf.predict(xtest_dtm)

print("Accuracy metrics:")

print("Accuracy of the classifier is",metrics.accuracy_score(ytest,predicted))

print("Confusion matrix:")

print(metrics.confusion_matrix(ytest,predicted))

print("Recall and Precision:")

print(metrics.recall_score(ytest,predicted))

 print(metrics.precision_score(ytest,predicted))
```

CSV File : 6_data_Text.csv

```
I love this sandwich,pos
This is an amazing placex,pos
I feel very good about these beers,pos
This is my best work,pos
What an awesome view,pos
I do not like this restaurant,neg
I am tired of this stuff,neg
I can't deal with this,neg
He is my sworn enemy,neg
My boss is horrible,neg
This is an awesome place,pos
I do not like the taste of this juice,neg
I love to dance,pos
I am sick and tired of this place,neg
What a great holiday,pos
That is a bad locality to stay,neg
We will have good fun tomorrow,pos
I went to my enemy's house today,neg
```

8. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering.

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

# import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns =  ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']

# Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X) # model.labels_ : Gives cluster no for which samples belongs to

# # Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications using Petal features
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
# Plot the Models Classifications
plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
# General EM for GMM
from sklearn import preprocessing
# transform your data such that its distribution will have a
# mean value 0 and standard deviation of 1.
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)

from sklearn.mixture import GaussianMixture
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
gmm = GaussianMixture(n_components=3)
```

```
gmm.fit(xs)
gmm_y = gmm.predict(xs)
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_y], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering matched the true labels more
closely than the Kmeans.')
```

9. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions

```
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn import datasets

iris=datasets.load_iris()

print("Iris Data set loaded...")

x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.1)

#random_state=0

for i in range(len(iris.target_names)):

    print("Label", i , "-",str(iris.target_names[i]))

classifier = KNeighborsClassifier(n_neighbors=1)

classifier.fit(x_train, y_train)

y_pred=classifier.predict(x_test)

print("Results of Classification using K-nn with K=1 ")

for r in range(0,len(x_test)):

    print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r])," Predicted-label:", str(y_pred[r]))


print("Classification Accuracy :" , classifier.score(x_test,y_test));
```

10.

```
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt

def local_regression(x0, X, Y, tau):
    x0 = np.array([1, x0])
    X = np.vstack(([1] * len(X), X)).T
    xw = X.T * np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau))
    beta = np.linalg.pinv(xw @ X) @ (xw @ Y)
    return x0 @ beta

def draw(tau):
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plt.plot(X, Y, 'o', color='black')
    plt.plot(domain, prediction, color='red')
    plt.title(f'Tau={tau}')
    plt.show()

X = np.linspace(-3, 3, num=1000)
domain = X
Y = np.log(np.abs(X ** 2 - 1) + .5)

draw(10)
draw(0.1)
draw(0.01)
draw(0.001)
```

7

```
import numpy as np

import pandas as pd

import csv

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination


heartDisease = pd.read_csv('heart.csv')

heartDisease = heartDisease.replace('?',np.nan)


print('Sample instances from the dataset are given below')

print(heartDisease.head())


print('\n Attributes and datatypes')

print(heartDisease.dtypes)


model=
BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisease'),('heartd
isease','restecg'),('heartdisease','chol')])

print('\nLearning CPD using Maximum likelihood estimators')

model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)


print('\n Inferencing with Bayesian Network:')

HeartDiseasetest_infer = VariableElimination(model)


print('\n 1. Probability of HeartDisease given evidence= restecg')
```

```
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'age':1})

print(q1)


print('\n 2. Probability of HeartDisease given evidence= cp ')

q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})

print(q2)
```