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

```
import csv
a = []
with open('ENJOYSPORT.csv', 'r') as csvfile:
    next(csvfile)
    for row in csv.reader(csvfile):
        a.append(row)
    print(a)

print("\nThe total number of training instances are : ",len(a))

num_attribute = len(a[0])-1

print("\nThe initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)

for i in range(0, len(a)):
    if a[i][num_attribute] == 'yes':
        print ("\nInstance ", i+1, "is", a[i], " and is Positive
Instance")
        for j in range(0, num_attribute):
            if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
                hypothesis[j] = a[i][j]
            else:
                hypothesis[j] = '?'
        print("The hypothesis for the training instance", i+1, " is:
" , hypothesis, "\n")

    if a[i][num_attribute] == 'no':
        print ("\nInstance ", i+1, "is", a[i], " and is Negative
Instance Hence Ignored")
        print("The hypothesis for the training instance", i+1, " is:
" , hypothesis, "\n")

print("\nThe Maximally specific hypothesis for the training instance
is ", hypothesis)
```

EXPERIMENT-2

AIM: For a given set of training data examples stored in a .CSV file, implement and demonstrate the CandidateElimination 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('ENJOYSPORT.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("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("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("Specific Bundary after ", i+1, "Instance is ",
specific_h)
        print("Generic Boundary after ", i+1, "Instance is ",
general_h)
        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")
```

EXPERIMENT-3

AIM: 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.

```python
import pandas as pd
import math
import numpy as np

data = pd.read_csv("ID3.csv")
features = [feat for feat in data]
features.remove("answer")
class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain
def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
```

```
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)

    return root
def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)
def classify(root: Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print ("Predicted Label for new example", new," is:",
child.pred)
                exit
            else:
                classify (child.children[0], new)
root = ID3(data, features)
print("Decision Tree is:")
printTree(root)
print ("------------------")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal",
"wind":"strong"}
classify (root, new)
```

EXPERIMENT-4

AIM: Exercises to solve the real-world problems using the following machine learning methods: a) Linear Regression b) Logistic Regression c) Binary Classifier

a) Linear Regression

```
import numpy as np
import matplotlib.pyplot as plt
```

```
import pandas as pd
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
dataset.head()
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 1/3, random_state = 0)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
y_pred = regressor.predict(X_test)
pd.DataFrame(data={'Actuals': y_test, 'Predictions': y_pred})
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

b) Logistic regression

```
import numpy as np
import pandas as pd
dataset = pd.read_csv('Dataset.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.30, random_state = 2)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='warn', tol=0.0001,
verbose=0,
                   warm_start=False)
from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test)
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
acc=accuracy_score(y_test, y_pred)
print(acc)
```

EXPERIMENT-5

AIM: Develop a program for Bias, Variance, Remove duplicates , Cross Validation

```
from mlxtend.evaluate import bias_variance_decomp

import numpy as np

import pandas as pd

from sklearn.linear_model import LinearRegression

from sklearn.utils import shuffle

from sklearn.metrics import mean_squared_error

data=pd.read_csv("https://raw.githubusercontent.com/amankharwal/Website-
data/master/student-mat.csv")

data = data[["G1", "G2", "G3", "studytime", "failures", "absences"]]

predict = "G3"

x = np.array(data.drop([predict], 1))

y = np.array(data[predict])

from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2)

linear_regression = LinearRegression()

linear_regression.fit(xtrain, ytrain)

y_pred = linear_regression.predict(xtest)

mse, bias, variance = bias_variance_decomp(linear_regression, xtrain, ytrain, xtest, ytest,

                        loss='mse', num_rounds=200, random_seed=123)

print("Average Bias : ", bias)

print("Average Variance : ", variance)
```

EXPERIMENT-6

AIM: Write a program to implement Categorical Encoding, One-hot Encoding

Categorical Encoding:

```
import pandas as pd

import numpy as np

bridge_types = ('Arch','Beam','Truss','Cantilever','Tied Arch','Suspension','Cable')

bridge_df = pd.DataFrame(bridge_types, columns=['Bridge_Types'])

bridge_df['Bridge_Types'] = bridge_df['Bridge_Types'].astype('category')

bridge_df['Bridge_Types_Cat'] = bridge_df['Bridge_Types'].cat.codes

print(bridge_df)
```

One-hot Encoding:

```
import numpy as np

colors = ["red", "green", "yellow", "red", "blue"]

total_colors = ["red", "green", "blue", "black", "yellow"]

mapping = {}

for x in range(len(total_colors)):

  mapping[total_colors[x]] = x

one_hot_encode = []

for c in colors:

  arr = list(np.zeros(len(total_colors), dtype = int))

  arr[mapping[c]] = 1

  one_hot_encode.append(arr)

print(one_hot_encode)
```

EXPERIMENT-7

AIM: 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) #maximum of X array longitudinally

y = y/100

def sigmoid (x):

   return 1/(1 + np.exp(-x))

def derivatives_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,hiddenlayer_neurons))

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

wout=np.random.uniform(size=(hiddenlayer_neurons,output_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 = derivatives_sigmoid(output)

   d_output = EO * outgrad

   EH = d_output.dot(wout.T)
```

```
    hiddengrad = derivatives_sigmoid(hlayer_act)

    d_hiddenlayer = EH * hiddengrad

    wout += hlayer_act.T.dot(d_output) *lr

    wh += X.T.dot(d_hiddenlayer) *lr

    print ("-----------Epoch-", i+1, "Starts----------")

    print("Input: \n" + str(X))

    print("Actual Output: \n" + str(y))

    print("Predicted Output: \n" ,output)

    print ("-----------Epoch-", i+1, "Ends----------\n")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)
```

EXPERIMENT-8

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

```
import numpy as np
import pandas as pd
dataset = pd.read_csv('Dataset.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.30, random_state = 42)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric =
'minkowski', p = 2)

classifier.fit(X_train, y_train)
from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
acc=accuracy_score(y_test, y_pred)
print(acc)
```

EXPERIMENT-9

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.

```
import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

def kernel(point, xmat, k):

    m,n = np.shape(xmat)

    weights = np.mat(np1.eye((m)))

    for j in range(m):

        diff = point - X[j]

        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))

    return weights

def localWeight(point, xmat, ymat, k):

    wei = kernel(point,xmat,k)

    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))

    return W

def localWeightRegression(xmat, ymat, k):

    m,n = np.shape(xmat)

    ypred = np.zeros(m)

    for i in range(m):

        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)

    return ypred

data = pd.read_csv('10-dataset.csv')

bill = np.array(data.total_bill)

tip = np.array(data.tip)
```

```
mbill = np.mat(bill)

mtip = np.mat(tip)

m= np.shape(mbill)[1]

one = np.mat(np1.ones(m))

X = np.hstack((one.T,mbill.T))

ypred = localWeightRegression(X,mtip,0.5)

SortIndex = X[:,1].argsort(0)

xsort = X[SortIndex][:,0]

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()
```

EXPERIMENT-10

AIM: Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/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('naivetext.csv',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)
print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)
cv = CountVectorizer()
xtrain_dtm = cv.fit_transform(xtrain)
xtest_dtm=cv.transform(xtest)
```

```
print('\n The words or Tokens in the text documents \n')
print(cv.get_feature_names())
df=pd.DataFrame(xtrain_dtm.toarray(),columns=cv.get_feature_names())
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)
print('\n Accuracy of the classifier is', metrics.accuracy
score(ytest,predicted))
print('\n confusion matrix')
print(metrics.confusion_matrix(ytest,predicted)
print('\n The value of precision', metrics.precision
score(ytest,predicted))
print('\n The value of
Recall',metrics.recall_score(ytest,predicted))
```

EXPERIMENT-11

Aim: Apply EM algorithm to cluster a Heart Disease Data Set. 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. You can add Java/Python ML library classes/API in the program.

```
from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

import sklearn.metrics as metrics

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']


dataset = pd.read_csv("8-dataset.csv", names=names)


X = dataset.iloc[:, :-1]


label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}


y = [label[c] for c in dataset.iloc[:, -1]]


plt.figure(figsize=(14,7))

colormap=np.array(['red','lime','black'])
```

```
# REAL PLOT
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])

# K-PLOT
model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])

print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y, model.labels_))

# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])

print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

EXPERIMENT-12

Aim: Exploratory Data Analysis for Classification using Pandas or Matplotlib

```
import pandas as pd
import numpy as np
import seaborn as sns
#Load the data
df = pd.read_csv('titanic.csv')
#View the data
df.head()
#Basic information
```

```
df.info()
#Describe the data
df.describe()
#Find the duplicates
df.duplicated().sum()
#unique values
df['Pclass'].unique()
df['Survived'].unique()
df['Sex'].unique()
array([3, 1, 2], dtype=int64)
array([0, 1], dtype=int64)
array(['male', 'female'], dtype=object)
#Plot the unique values
sns.countplot(df['Pclass']).unique()
#Find null values
df.isnull().sum()
```

EXPERIMENT-13

Aim: Write a Python program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

```
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'),('heartdisease','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={'restecg':1})

print(q1)

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

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

print(q2)

EXPERIMENT-14

AIM: Write a program to Implement Support Vector Machines and Principle Component Analysis.

```
import numpy as np
import pandas as pd
dataset = pd.read_csv('BreastCancer.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.25, random_state = 0)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3,
gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
```

```
print(cm)
acc=accuracy_score(y_test, y_pred)
print(acc)
```

EXPERIMENT-15

AIM: Write a program to Implement Principle Component Analysis

```
import numpy as np
import pandas as pd
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',
'Class']
dataset = pd.read_csv(url, names=names)
dataset.head()

X = dataset.drop('Class', 1)
y = dataset['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)


from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.decomposition import PCA

pca = PCA()
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
from sklearn.decomposition import PCA

pca = PCA(n_components=1)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)
acc=accuracy_score(y_test, y_pred)
```

```
print(acc)
```