S.no	Experiments	Dates
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.	20/1/23
2.	Python Implementation of Candidate-Elimination	10/2/23
3.	Write s program to demonstrate working of decision tree based ID3 algorithm	17/2/23
4.	Exercises to solve the real world problems using the following machine learning methods. • Linear Regression • Logistic Regression • Binary classifier	3/3/23
5.	Develop a program for Bias, Variance, Remove duplicates , Cross Validation	17/3/23
6.	Build an Artificial Neural Networks by Implementing the Back Propagation Algorithm and text the same using appropriate data sets	17/3/23
7.	Write a program to implement categorical Encoding. One-Hot Encoding	24/3/23
8.	Write a program to Implement support vector machine	31/3/23
9.	Write a program to implement k-means algorithm to classify the iris dataset print both correct and wrong predictions	19/4/23
10	Write a program to implement principle component analysis	19/4/23

WEEK – 1: EXPERIMENT 1:

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.

FIND-S Algorithm

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

Training Examples:

Example Sky AirTemp Humidity Wind Water Forecast EnjoySport
1 Sunny Warm Normal Strong Warm Same Yes
2 Sunny Warm High Strong Warm Same Yes
3 Rainy Cold High Strong Warm Change No
4 Sunny Warm High Strong Cool Change Yes

Program:

```
import csv
num_attributes = 6
a = []
print("\n The Given Training Data Set \n")
with open('enjoysport.csv', 'r') as csvfile:
 reader = csv.reader(csvfile)
 for row in reader:
 a.append (row)
 print(row)
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)
```

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

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 Example No:0 the hypothesis is
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
For Training Example No:1 the hypothesis is

['sunny', 'warm', '?', 'strong', 'warm', 'same']
For Training Example No:2 the hypothesis is
'sunny', 'warm', '?', 'strong', 'warm', 'same']
For Training Example No:3 the hypothesis is
'sunny', 'warm', '?', 'strong', '?', '?']
The Maximally Specific Hypothesis for a given Training Examples:
['sunny', 'warm', '?', 'strong', '?', '?']

WEEK-2: EXPERIMENT 2 :

AIM: Python Implementation of Candidate-Elimination

Below is the algorithm for Candidate-Elimination

- Firstly, read the data from the CSV file.
- Initialize General and Specific Hypothesis.
- If the example is positive, [Follow Find-S algorithm]
- If attribute == hypothesis value then do nothing.
  Else
- make the attribute more general i.e replace the attribute with ?

- If the example is negative
- Make the generalized hypothesis more specific.

Below is the code for Candidate-Elimination

Contents in candidate.csv

```
sky, air temp,    humidity, wind, water, for cast, 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.
```

program

```
import numpy as np
import pandas as pd
# Reading the data from CSV file
data = pd.read_csv('candidate.csv')
concepts = np.array(data.iloc[:,:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
```

```
def train(concepts, target):

  # Initializing general and specific hypothesis
  specific_h = concepts[0].copy()
  print("\nInitialization of specific hypothesis and general hypothesis")
  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, val in enumerate(concepts):
    print("\nInstance", i+1 , "is ", val)
    #positive example
    if target[i] == "yes":
      print("Instance is Positive ")
      for x in range(len(specific_h)):
        if val[x]!= specific_h[x]:
          specific_h[x] ='?'
          general_h[x][x] ='?'
    #negative example
    if target[i] == "no":
      print("Instance is Negative ")
      for x in range(len(specific_h)):
        if val[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 = train(concepts, target)
# displaying Specific_hypothesis
print("Final Specific_h: ", s_final, sep="\n")
# displaying Generalized_Hypothesis
print("Final General_h: ", g_final, sep="\n")
```

OUTPUT:

Instances are:

```
[['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame']
['\twarm' '\thigh' '\tstrong' '\twarm' '\tsame' '\tyes.']
['\tcold' '\thigh' '\tstrong' '\twarm' '\tchange' '\tno.']
['\twarm' '\thigh' '\tstrong' '\tcool' '\tchange' '\tyes.']]
 Target Values are:  ['\tyes.' nan nan nan]
Initialization of specific hypothesis and general hypothesis
Specific Boundary:  ['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame']
Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Instance 1 is  ['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame']
Specific Bundary after  1 Instance is  ['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame']
Generic Boundary after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Instance 2 is  ['\twarm' '\thigh' '\tstrong' '\twarm' '\tsame' '\tyes.']
Specific Bundary after  2 Instance is  ['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame']
Generic Boundary after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Instance 3 is  ['\tcold' '\thigh' '\tstrong' '\twarm' '\tchange' '\tno.']
Specific Bundary after  3 Instance is  ['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame']
Generic Boundary after  3 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 4 is  ['\twarm' '\thigh' '\tstrong' '\tcool' '\tchange' '\tyes.']
Specific Bundary after  4 Instance is  ['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame']
Generic Boundary after  4 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Final Specific_h:
['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame']
Final General_h:
[]
```

WEEK-3: EXPERIMENT 3:

AIM: Write s program to demonstrate working of decision tree based I

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Program:

```
import pandas as pd
import math
import numpy as np
data = pd.read_csv("dataset.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
```

```python
    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)
        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":
```

classify (root, new)

Week 4 : Experiment 4:

Aim: Exercises to solve the real world problems using the following machine learning methods.

- Linear Regression
- Logistic Regression
- Binary classifier

Program:

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
dataset = load_breast_cancer(as_frame=True)
dataset['data'].head()

dataset['target'].head()
dataset['target'].value_counts()
X = dataset['data']
y = dataset['target']
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

ss_train = StandardScaler()
X_train = ss_train.fit_transform(X_train)
ss_test = StandardScaler()
X_test = ss_test.fit_transform(X_test)

predictions = model.predict(X_test)
models = {}
from Sklearn.linear_model import LogisticRegression
      models('logistic Regression') = logistic Regression()
from Sklearn.svm  import Linearsvc
      models('Support Vector Machine') = Linearsvc()
from Sklearn.tree  import DecisionTreeclassifier
      models('Decision Trees') = DecisionTreeclassifier ()
from Sklearn.ensemble  import  RandomForestclassifier
      models('Random Forest') = RandomForestclassifier ()
from Sklearn.neighbors  import  kneighborsclassifier
      models('k_Nearest neighbors') = kneighborsclassifier()
from Sklearn.metrics  import  accuracy_score,precision_score,rec
accuracy,precision,recall={}{}{}

for key in models.keys():
      models[key].fit(x_train,y_train)
      predictions = models[key].predict(x_test)
      accuracy[key] = accuracy_score(predictions,y_test)
      precision[key] = precision_score(predictions,y_test)
      recall[key] = recall_score(predictions,y_test)
      logistic Regression()
      linear svc()
      decisiontree classifier()
      randomForestclassifier()
      GaussianNB()
      KNeighbourclassifier()
      Import pandas as pd
      df_model = pd.dataframe(index = models.keys(),columns =
                                             ['Accuracy','precisi
      df_model('Accuracy') = accuracy.values()
      df_model('precision') = precision.values()
      df_model('recall') = recall.values()
      df_model
```

Output:

	Accuracy	precision	recall
Logistic Regression	0.95804	0.955556	0.977273
Support Vector Machine	0.937063	0.93333	0.965517
Decision tree	0.881119	0.84444	0.962025
Random Forest	0.965035	0.95556	0.988506
Naïve Bayes	0.937063	0.95556	0.945055
K-Nearest neighbor	0.951049	0.98889	0.936842

Week 5 : Experiment 5:

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

Program:

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from mlxtend.evaluate import bias_variance_decomp
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
dataframe = read_csv(url, header=None)
# separate into inputs and Outputs
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
# define the model
model = LinearRegression()
# estimate bias and variance
mse, bias, var = bias_variance_decomp(model, X_train, y_train, X_test, y_test, los
num_rounds=200, random_seed=1)
# summarize results
print('MSE: %.3f' % mse)
print('Bias: %.3f' % bias)
print('Variance: %.3f' % var)
```

Output:

MSE: 22.418

BIAS: 20.744

VARAINCE: 1.674

Week 6: Experiment 6:

AIM: Build an Artificial Neural Networks by Implementing the Back Propagation Algorithm and text the same using appropriate data sets

Program:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
data = load_iris()
X=data.data
y=data.target
y = pd.get_dummies(y).values
y[:3]
```

Output:

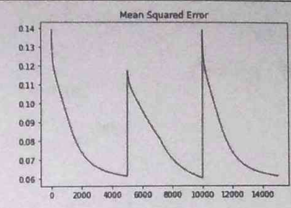
```
array([[1, 0, 0],
 [1, 0, 0],
```

[1, 0, 0]], dtype=uint8)

Program:

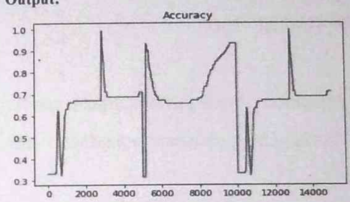
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20,
random_state=4)
learning_rate = 0.1
iterations = 5000
N = y_train.size
input_size = 4
hidden_size = 2
output_size = 3
results = pd.DataFrame(columns=["mse", "accuracy"])
np.random.seed(10)

W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))

W2 = np.random.normal(scale=0.5, size=(hidden_size , Output_size))

def sigmoid(x):
  return 1 / (1 + np.exp(-x))
  def mean_squared_error(y_pred, y_true):
  return ((y_pred - y_true)**2).sum() / (2*y_pred.size)
  def accuracy(y_pred, y_true):
  acc = y_pred.argmax(axis=1) == y_true.argmax(axis=1)
  return acc.mean()
  for itr in range(iterations):
  Z1 = np.dot(x_train, W1)
  A1 = sigmoid(Z1)
  Z2 = np.dot(A1, W2)
  A2 = sigmoid(Z2)
  mse = mean_squared_error(A2, y_train)
  acc = accuracy(A2, y_train)
  results=results.append({"mse":mse, "accuracy":acc},ignore_index=True )
  E1 = A2 - y_train
  dW1 = E1 * A2 * (1 - A2)
  E2 = np.dot(dW1, W2.T)
  dW2 = E2 * A1 * (1 - A1)
  W2_update = np.dot(A1.T, dW1) / N
  W1_update = np.dot(x_train.T, dW2) / N
  W2 = W2 - learning_rate * W2_update
  W1 = W1 - learning_rate * W1_update

results.mse.plot(title="Mean Squared Error")
```

Output:




```
results.accuracy.plot(title="Accuracy")
```

Output:



Program:

```
# feedforward
Z1 = np.dot(x_test, W1)
A1 = sigmoid(Z1)

Z2 = np.dot(A1, W2)
A2 = sigmoid(Z2)

acc = accuracy(A2, y_test)
print("Accuracy: {}".format(acc))
```

Output:

```
Accuracy: 0.8
```

Week 7: Experiment 7:

Aim: Write a program to implement categorical Encoding. One-Hot E

Program: