Output:

Enter the value of n: 2
The 2 terms of the fibonacci are
0
1

Name of Experiment: Example Programs Date: 14/5/2021 Page No. 01
Experiment No. Experiment Result

Example Programs

1) Defined as a function F as $F_n = F_{n-1} + F_{n-2}$. Write a python program which accepts a value of n (where $n \geq 0$) as input and pass this value to the function. Display suitable error message if the condition for input value is not followed.

```
def F(n):
    ft = 0
    st = 1
    tt = 0
    if n<=0:
        print("The value of n is expected to be
              greater than 0 but received", n)
    else:
        print(F"The {n} terms of fibonacci are:")
        print(ft, '\n', st)
        n = n-2
        while(n>0):
            tt = ft+st
            print(tt, '\n')
            ft = st
            st = tt
            n-=1
            ##n = n-1
n = int(input("Enter the value of n:"))
F(n)
```

Output:

```
Enter the marks of test1 : 20
Enter the marks of test2 : 19
Enter the marks of test3 : 18
The average is : 19.5
```

Name of Experiment: Example Programs Date: 14/5/2024 Page No. 02

Experiment No.: Experiment Result:

3) Write a python program to find the best of two test average marks out of three test marks accepted from the user.

```
test1 = int(input('Enter the marks of test1:'))
test2 = int(input('Enter the marks of test2:'))
test3 = int(input('Enter the marks of test3:'))
a=0
b=0
c=0
if test1>test2 or test1>test3:
    a = test1
if test2>test1 or test2>test3:
    b=test2
if test3>test1 or test3>test2:
    c=test3
if test1==test2==test3:
    print('The average is:', (test1+test2)/2)
else:
    print('The average is:', (a+b+c)/2)
```

Output:

104

Output:

Enter a number : 15
Sum of natural numbers which are multiples of 5 = 30

Name of Experiment: Example Programs Date: 14/5/2024 Page No. 03
Experiment No. Experiment Result

3) Write a program to convert roman numbers into integer values using dictionaries.

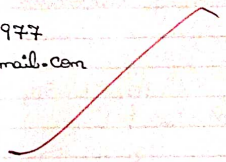
```
def RomantoInt(s):
    x = {'I':1, 'V':5, 'X':10, 'L':50, 'C':100,
         'D':500, 'M':1000}
    y=0
    for i in range(len(s)):
        if i>0 and x[s[i]] > x[s[i-1]]:
            y+=x[s[i]]-2*x[s[i-1]]
        else:
            y+=x[s[i]]
    return y
print(RomantoInt('CIV'))
```

4) Write a program to add multiples of 5 upto N natural numbers.

```
num = int(input('Enter a number:'))
sum =0
for number in range(1, num+1):
    if number%5 ==0:
        sum+= number
print('Sum of natural numbers which are
      multiples of 5=', sum)
```

Output:

+919900889977
sample@gmail.com



Name of Experiment: Example Programs Date: 14/5/2024 Page No. 04
Experiment No. Experiment Result.

5) Develop a python program that could search the text in a file for phone numbers (+919900889977) and email address (sample@gmail.com).

```
import re
phone_regex = re.compile(r'\+\d{12}')
email_regex = re.compile(r'[A-Za-z0-9._]
        +@[A-Za-z0-9]+\.[A-Z|a-z]
                {2,}')
with open('example.text','r') as f:
    for line in f:
        matches = phone_regex.findall(line)
        for match in matches:
            print(match)
        matches = email_regex.findall(line)
        for match in matches:
            print(match)
```

14/5/24

W-4
C-4
V-2
10

w-8.CSV:

Sky, temp, Humid, Wind, Water, Forecast, EnjoySpt
Sunny, Warm, Normal, Strong, Warm, same, true
Sunny, Warm, High, Strong, Warm, same, true
Rainy, Cold, High, Strong, Warm, change, false
Sunny, Warm, High, Strong, Cool, change, true

Output:

Maximally specific hypothesis is:
['Sunny', 'Warm', '?', 'Strong', '?', '?']

Name of Experiment: Find-S algorithm Date: 21/5/2024 Page No. 05
Experiment No.: 01 Experiment Result:

Program 1:
Aim: Illustrate and Demonstrate the working model and principle of Find.-S algorithm.

Program 1:
For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.

```
import csv

#Initialize the hypothesis to the most specific hypothesis
h = ['0', '0', '0', '0', '0', '0']

#Load the dataset from CSV
with open('w-8.CSV', 'r') as f:
    reader = csv.reader(f)
    next(reader)  #Skip header
    your_list = list(reader)

#Apply the Find-S algorithm
for i in your_list:
    if len(i)>0 and i[-1] == "true":
        for j in range(len(i)-1):
            if i[j] != h[j] and h[j]=='0':
                h[j] = i[j]
```

Name of Experiment: Find-S algorithm Date: 21/5/2024 Page No. 06

Experiment No: 01 Experiment Result:

```
if i[j] != h[j] and h[j] == '0':
    h[j] = '?'

#Print the maximally specific hypothesis
print("Maximally specific hypothesis is:")
print(h)
```

21/5/24

W - 4
C - 4
V - 2
10

tennis.csv:

Sky, temp, Humid, Wind, Water, Forecast, EnjoySpt
Sunny, Warm, Normal, Strong, Warm, same, true
Sunny, Warm, High, Strong, Warm, same, true
Rainy, cold, High, Strong, Warm, Change, false
Sunny, Warm, High, Strong, Cool, Change, true

Name of Experiment: Candidate Elimination Date: Page No. 07
Experiment No.: 02 Experiment Result:

Program 2:

Aim: Demonstrate the working model and principle of candidate elimination algorithm.

Program 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

#Define the path to your CSV file
path = 'tennis.csv' #Use the correct relative path to your file

#Load the data
data = pd.read_csv(path)

#Separate the data into instances (concepts) and target values
concepts = np.array(data.iloc[:,0:-1])
print("\n Instances are:\n", concepts)
target = np.array(data.iloc[:,-1])
print("\n Target Values are:", target)
```

Output:

```
Instances are:
 [['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'same']
 ['Rainy' 'cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]

Target Values are: [ True  True False  True]

Initialization of specific_h and general_h

Specific Boundary: ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'same']

Generic Boundary: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'same']
Instance is Positive
Specific Boundary after 1 Instance is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm'
'same']
Generic Boundary after 1 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Instance 2 is ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'same']
Instance is Positive
Specific Boundary after 2 Instance is ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'sam
e']
Generic Boundary after 2 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Instance 3 is ['Rainy' 'cold' 'High' 'Strong' 'Warm' 'Change']
Instance is Negative
Specific Boundary after 3 Instance is ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'sam
e']
Generic Boundary after 3 Instance is [['Sunny', '?', '?', '?', '?', '?'], ['?',
'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'sam
e']]


Instance 4 is ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']
Instance is Positive
Specific Boundary after 4 Instance is ['Sunny' 'Warm' '?' 'Strong' '?' '?']
Generic Boundary after 4 Instance is [['Sunny', '?', '?', '?', '?', '?'], ['?',
'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']
Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

Name of Experiment: Candidate Elimination Date: Page No. 08

Experiment No.: 02 Experiment Result:

```
#Function to learn the hypotheses
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\n Initialization of specific_h and
                                    general_h")
    print("\n Specific Boundary:", specific_h)
    general_h = [["?" for i in range(len(specific_h))]
                 for i in range(len(specific_h))]
    print("\n Generic Boundary:", general_h)

#Iterate through each instance and update
 hypotheses
    for i, h in enumerate(concepts):
        print("\n Instance", i+1, "is", h)
        if target[i]==True:
            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]==False:
            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]='?'
```

Name of Experiment: Candidate Elimination Date: Page No. 09

Experiment No.: 02 Experiment Result:

```
        print("Specific Boundary after", i+1, "Instance is", specific_h)
        print("Generic Boundary after", i+1, "Instance is", general_h)
        print("\n")

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

#Learn from the data
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

data3.csv:

outlook, temperature, humidity, wind, target
sunny, hot, high, weak, no
sunny, hot, high, strong, no
overcast, hot, high, weak, yes
rain, mild, high, weak, yes
rain, cool, normal, weak, yes
rain, cool, normal, strong, no
overcast, cool, normal, strong, yes
sunny, mild, high, weak, no
sunny, cool, normal, weak, yes
rain, mild, normal, weak, yes
sunny, mild, normal, strong, yes
overcast, mild, high, strong, yes
overcast, hot, normal, weak, yes
rain, mild, high, strong, no

Name of Experiment... ID3...algorithm Date ..... Page No. 10
Experiment No....03..... Experiment Result.....

Program 3:
Aim: To construct the Decision tree using the training data sets under supervised learning concepts.

Program 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.

data loader Program:

```
import csv
def read_data(filename):
    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        headers = next(datareader)
        metadata = []
        traindata = []
        for name in headers:
            metadata.append(name)
        for row in datareader:
            traindata.append(row)
    return(metadata, traindata)
```

Output:

Display Tree
{'outlook': {'outlook': 'target', 'overcast': 'yes', 'rain':
{'wind': {'strong': 'no', 'weak': 'yes'}}, 'sunny': {'humid-
-ity': {'high': 'no', 'normal': 'yes'}}}}

Name of Experiment: ID3 algorithm Date: Page No. 11
Experiment No.: 03 Experiment Result:

Main program:

```
import pandas as pd
import numpy as np
dataset = pd.read_csv('data3.csv', names=['outlook',
        'temperature', 'humidity', 'wind', 'class',])
def entropy(target_col):
    elements, counts = np.unique(target_col, return
                    _counts = True)
    entropy = np.sum([(-counts[i]/np.sum(counts))
            *np.log2(counts[i]/np.sum(counts)) for
            i in range(len(elements))])
    return entropy

def InfoGain(data, split_attribute_name, target_
            name = "class"):
    total_entropy = entropy(data[target_name])
    vals, counts = np.unique(data[split_attribute
                _name], return_counts = True)
    Weighted_Entropy = np.sum([((counts[i]/np.sum
            (counts))*entropy(data.where(data[split
            _attribute_name] == vals[i]).dropna()[target
            _name]) for i in range(len(vals))])
    Information_Gain = total_entropy - Weighted_Entropy
    return Information_Gain

def ID3(data, originaldata, features, target_attribute
        _name = "class", parent_node_class = None):
    if len(np.unique(data[target_attribute_name])) <= 1:
```

Name of Experiment: ID3 algorithm Date ..... Page No. 12

Experiment No.: 03 Experiment Result.....

```
        return np.unique(data[target_attribute_name])[0]
    elif len(data)==0:
        return np.unique(originaldata[target_attribute
            _name])[np.argmax(np.unique(original
            data[target_attribute_name], return_
            counts = True)[1])]
    elif len(features) ==0:
        return parent_node_class
    else:
        parent_node_class=np.unique(data[target_
            attribute_name])[np.argmax(np.unique
            (data[target_attribute_name], return
            _counts = True)[1])]
        item_values = [InfoGain(data, feature, target_
            attribute_name) for feature in
            features]
        best_feature_index = np.argmax(item_values)
        best_feature = features[best_feature_index]
        tree = {best_feature:{}}
        features = [i for i in features if i !=
            best_feature]
        for value in np.unique(data[best_feature
            ]):
            value = value
            sub_data = data.where(data[best_feature]
                ==value).dropna()
            subtree = ID3(sub_data, dataset,
                features, target_attribute_name,
                parent_node_class)
```

Name of Experiment: ID3 algorithm Date: Page No. 13

Experiment No.: 03 Experiment Result:

```
        tree[best_feature][value] = subtree
    return(tree)
tree = ID3(dataset, dataset, dataset.columns[:-1])
print('\n Display Tree \n', tree)
```

Output:

input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]

Actual output:
[[0.92]
[0.86]
[0.89]]

predicted output:
[[0.89378511]
[0.88123975]
[0.89480415]]

Name of Experiment: Backpropagation algorithm Date: Page No. 14
Experiment No.: 04 Experiment Result:

Program 1:
Aim: To understand the working principle of Artificial Neural network with feed forward and feed backward principles.

Program 1:
Build an Artificial Neural Network by implementing the Backpropagation 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 derivatives_sigmoid(x):
    return x*(1-x)
epoch = 7000
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_neu-
                              -rons))
```

Name of Experiment: Backpropagation algorithm Date: Page No. 15

Experiment No.: 04 Experiment Result:

```
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("input:\n" + str(X))
print("Actual output:\n" + str(Y))
print("predicted output:\n", output)
```

Inputs:

1,1,1,1,5
1,1,1,2,5
2,1,1,2,10
3,2,1,1,10
3,3,2,1,10
3,3,2,2,5
2,3,2,2,10
1,2,1,1,5
1,3,2,1,10
3,2,2,2,10
1,2,2,2,10
2,2,1,2,10
2,1,2,1,10
3,2,1,2,5
1,2,1,2,5

Name of Experiment: Naive bayes classifier algorithm Date Page No. 16

Experiment No. 05 Experiment Result

Program 5:

Aim: Demonstrate the text classifier using Naive bayes classifier algorithm.

Program 5:

Write a program to implement the naive bayesian classifier for a sample training dataset stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
import pandas as pd
df = pd.read_csv("5data.csv")
headers = df.columns.values
data = df.iloc[:, 0:-1].values
target = df.iloc[:, -1].values

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data, target, test_size = 0.2, random_state = 1)
print("\nThe first 3 rows into:" .format(len(data)))
print("Number of training data:" +(repr(len(x_train))))
print("Number of test data:" +(repr(len(x_test))))
print("\n The values assumed for concept learning attributes are:")
print("Outlook => Sunny=1 overcast=2 rain=3
      Temperature => Hot=1 mild=2 cool=3
      Humidity => high=1 normal=2
```

Output:

```
split 14 rows into:
Number of training data: 11
Number of test data: 3

The values assumed for concept learning attributes are:
Outlook => sunny=1  overcast=2  rain=3 Temperature => Hot=1  mild=2  cool=3 Humidity => high=1 normal=2 Wind => weak=1 strong=2
Target cincept: Play Teenis => Yes=10 No=5
The training set:
(array([3, 2, 1, 1], dtype=int64), 10)
(array([2, 2, 1, 2], dtype=int64), 10)
(array([3, 3, 2, 2], dtype=int64), 5)
(array([2, 1, 1, 2], dtype=int64), 10)
(array([3, 2, 1, 2], dtype=int64), 5)
(array([1, 1, 1, 2], dtype=int64), 5)
(array([1, 2, 1, 2], dtype=int64), 5)
(array([1, 2, 2, 2], dtype=int64), 10)
(array([3, 2, 2, 2], dtype=int64), 10)
(array([2, 1, 2, 1], dtype=int64), 10)
(array([2, 3, 2, 2], dtype=int64), 10)
The test set:
(array([3, 3, 2, 1], dtype=int64), 10)
(array([1, 3, 2, 1], dtype=int64), 10)
(array([1, 2, 1, 1], dtype=int64), 5)
prediction for tbe given training set  [10 10 10]
Gaussiannaivebayes odel accuracy (in %):  66.66666666666666
```

Name of Experiment: Naive bayes classifier algorithm Date: Page No. 17

Experiment No.: 05 Experiment Result:

```
(wind => weak=1 strong=2")
print("Target cincept: Play Tennis => Yes=10 No=5")
print("The training set:")

for x, y in zip(x_train, y_train):
    print((x, y))
print("The test set:")
for x, y in zip(x_test, y_test):
    print((x, y))

from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)
y_pred = gnb.predict(x_test)
print("prediction for the given training set", y_pred)

from sklearn import metrics
print("Gaussiannaivebayes odel accuracy(in %):",
      metrics.accuracy_score(y_test, y_pred)*100)
```