Bachelor of Technology (B.Tech)

Department of Computer Science and Engineering

III year II sem- Machine Learning Laboratory Manual

SIDDHARTHA INSTITUTE OF TECHNOLOGY & SCIENCES

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)Accredited by NBA and NAAC with 'A+' Grade

Narapally, Korremula Road, Ghatkesar, Medchal- Malkajgiri (Dist)-501 301

Vision of the Institute

To be a reputed institute in technical education towards research, industrial and societal needs.

Mission of the Institute

Mission	Statement
IM$_1$	Provide state-of-the-art infrastructure, review, innovative and experiment teaching –learning methodologies.
IM$_2$	Promote training, research and consultancy through an integrated institute industry symbiosis
IM$_3$	Involve in activities to groom professional, ethical values and social responsibility

SIDDHARTHA INSTITUTE OF TECHNOLOGY AND SCIENCES

(*Approved by AICTE, New Delhi &Affiliated to JNTUH, Hyderabad*)

Narapally, Telangana – 500 088.

Department of Computer Science and Engineering

Vision of the Department

To be a recognized center of Computer Science education with values, and quality research

Mission of the Department

Mission	Statement
DM1	Impart high quality professional training with an emphasis on basic principles of Computer Science and allied Engineering
DM2	Imbibe social awareness and responsibility to serve the society
DM3	Provide academic facilities, organize collaborated activities to enable overall development of stakeholders

SIDDHARTHA INSTITUTE OF TECHNOLOGY AND SCIENCES

(*Approved by AICTE, New Delhi &Affiliated to JNTUH, Hyderabad*)

Narapally, Telangana – 500 088.

Department of Computer Science and Engineering

Program Educational Objectives (PEOs)

PEO’s	Statement
PEO1	Graduates will be able to solve Computer Science and allied Engineering problems, develop proficiency in computational tools.
PEO2	Graduates will be able to communicate and work efficiently in Multidisciplinary teams with a sense of professional and social responsibility.
PEO3	Graduates will be able to exhibit lifelong learning ability andpursue career as architects, software developers and entrepreneurs.

Programme Outcomes

PO1	**Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem Analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental context, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team network:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-Long learning:** Recognize the need for, and have the preparation and able to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes:

PSO1	**Program Applications:** Able to develop programs modules for cloud based applications.
PSO2	**Development Tools:** Able to use tools such as Weka, Rational Rose Raspberry-Pi, Sql and advanced tools

List of Experiments

1. The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is the probability that a student is absent given that today is Friday? Apply Baye's rule in python to get the result. (Ans: 15%)

2. Extract the data from database using python

3. Implement k-nearest neighbours classification using python

4. Given the following data, which specify classifications for nine combinations of VAR1 and VAR2

predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of kmeans clustering with 3 means (i.e., 3 centroids)

VAR1	VAR2	CLASS
1.713	1.586	0
0.180	1.786	1
0.353	1.240	1
0.940	1.566	0
1.486	0.759	1
1.266	1.106	0
1.540	0.419	1
0.459	1.799	1
0.773	0.186	1

5. 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.

6. Implement linear regression using python.

7. Implement Naïve Bayes theorem to classify the English text

8. Implement an algorithm to demonstrate the significance of genetic algorithm

9. Implement the finite words classification system using Back-propagation algorithm

10. Implement a Candidate elimination algorithm.

EXNO1. The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school

days in a week, the probability that it is Friday is 20 %. What is the probability that a student is

absent given that today is Friday? Apply Baye's rule in python to get the result. (Ans: 15%)

F: It is a Friday

A: Student is absent According to the problem statement, we have $P(A \cap F)=3\ \%$ and $P(F)=20\%$

The problem requires us to find $P(A|F)$

$$P(A \mid F) = \frac{P(A \cap F)}{P(F)}$$

or, $$P(A \mid F) = \frac{0.03}{0.2} = \frac{3}{100} \div \frac{2}{10} = \frac{3}{100} \times \frac{10}{2} = \frac{3}{20} = 0.15$$

```
# calculate the probability of cancer patient and diagnostic test
 # calculate P(A|B) given P(F), P(B|A), P(B|not A)
def bayes_theorem(p_f, p_b_given_a, p_b_given_not_a):
        # calculate P(not A)
        not_a = 1 - p_f
        # calculate P(B)
        p_b = p_b_given_a * p_f + p_b_given_not_a * not_a
        # calculate P(A|B)
        p_a_given_b = (p_b_given_a * p_f) / p_b
        return p_a_given_b

# P(A)
p_f = 0.02
# P(B|A)
p_b_given_a = 0.85
# P(B|not A)
p_b_given_not_a = 0.095
# calculate P(A|B)
result = bayes_theorem(p_f, p_b_given_a, p_b_given_not_a)
# summarize
print('P(A|B) = %.f%%' % (result * 100))
```

OUTPUT

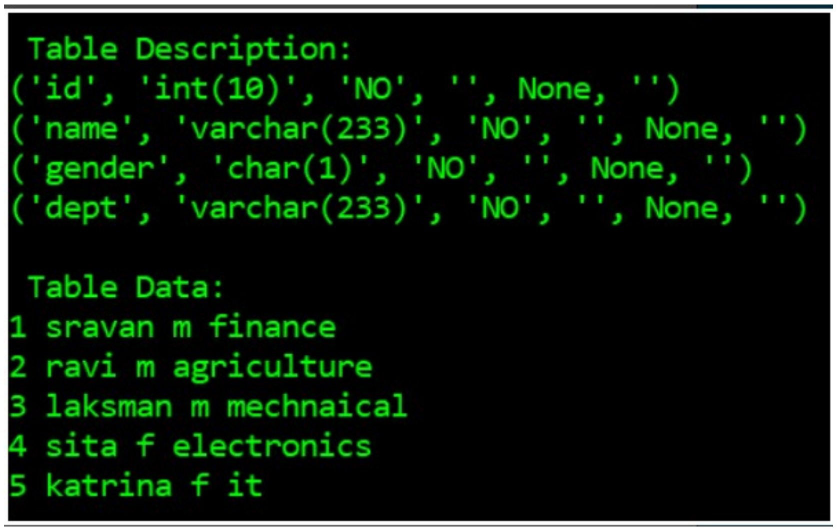
```
C:\Users\HP\Desktop\coding>python bfinal.py

P(A|B) = 15%
```

EXNO 2. Extract the data from database using python

Python program for extracting data

```
# import required modules
import mysql.connector

# create connection object
con = mysql.connector.connect(
  host="localhost", user="root",
  password="", database="GEEK")

# create cursor object
cursor = con.cursor()

# assign data query
query1 = "desc geeksdemo"

# executing cursor
cursor.execute(query1)

# display all records
table = cursor.fetchall()

# describe table
print('\n Table Description:')
for attr in table:
    print(attr)

# assign data query
query2 = "select * from geeksdemo"

# executing cursor
cursor.execute(query2)

# display all records
table = cursor.fetchall()

# fetch all columns
print('\n Table Data:')
for row in table:
    print(row[0], end=" ")
    print(row[1], end=" ")
    print(row[2], end=" ")
    print(row[3], end="\n")

# closing cursor connection
cursor.close()

# closing connection object
con.close()
```

OUTPUT



```
 Table Description:
('id', 'int(10)', 'NO', '', None, '')
('name', 'varchar(233)', 'NO', '', None, '')
('gender', 'char(1)', 'NO', '', None, '')
('dept', 'varchar(233)', 'NO', '', None, '')

 Table Data:
1 sravan m finance
2 ravi m agriculture
3 laksman m mechnaical
4 sita f electronics
5 katrina f it
```

EX NO 3: Implement k-nearest neighbours classification using python

Theory

k-Nearest Neighbors

The k-Nearest Neighbors algorithm or KNN for short is a very simple technique.

The entire training dataset is stored. When a prediction is required, the k-most similar records to a new record from the training dataset are then located. From these neighbors, a summarized prediction is made.

Similarity between records can be measured many different ways. A problem or data-specific method can be used. Generally, with tabular data, a good starting point is the Euclidean distance.

steps

First we will develop each piece of the algorithm in this section, then we will tie all of the elements together into a working implementation applied to a real dataset in the next section.

This k-Nearest Neighbors tutorial is broken down into 3 parts:

- **Step 1**: Calculate Euclidean Distance.
- **Step 2**: Get Nearest Neighbors.
- **Step 3**: Make Predictions.

These steps will teach you the fundamentals of implementing and applying the k-Nearest Neighbors algorithm for classification and regression predictive modeling problems.

# Example of getting neighbors for an instance

```
from math import sqrt
# calculate the Euclidean distance between two
vectors def euclidean_distance(row1, row2):
distance = 0.0
for i in range(len(row1)-1):
distance += (row1[i] - row2[i])**2
return sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row,
num_neighbors): distances = list()
for train_row in train:
dist = euclidean_distance(test_row, train_row)
distances.append((train_row, dist))
distances.sort(key=lambda tup:
tup[1]) neighbors = list()
for i in range(num_neighbors):
neighbors.append(distances[i][0])
return neighbors

# Test distance function
dataset = [[2.7810836,2.550537003,0],
[1.465489372,2.362125076,0],
[3.396561688,4.400293529,0],
[1.38807019,1.850220317,0],
[3.06407232,3.005305973,0],
[7.627531214,2.759262235,1],
[5.332441248,2.088626775,1],
[6.922596716,1.77106367,1],
[8.675418651,-0.242068655,1],
[7.673756466,3.508563011,1]]
neighbors = get_neighbors(dataset, dataset[0],
3) for neighbor in neighbors:
print(neighbor)
```

OUTPUT:

```
[2.7810836, 2.550537003, 0]
[3.06407232, 3.005305973, 0]
[1.465489372, 2.362125076, 0]
```

EXNO : 4.Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of kmeans clustering with 3 means (i.e., 3 centroids)

VAR1	VAR2	CLASS
1.713	1.586	0
0.180	1.786	1
0.353	1.240	1
0.940	1.566	0
1.486	0.759	1
1.266	1.106	0
1.540	0.419	1
0.459	1.799	1
0.773	0.186	1

Coding:

```
from pandas import DataFrame
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

Data = {'x': [1.713 , 0.180, 0.353, 0.940,  1.486,  1.266, 1.540,  0.459, 0.773],
       'y': [1.586, 1.786, 1.240, 1.566, 0.759, 1.106, 0.419, 1.799, 0.186]
      }

df = DataFrame(Data,columns=['x','y'])

kmeans = KMeans(n_clusters=3).fit(df)
centroids = kmeans.cluster_centers_
print(centroids)

plt.scatter(df['x'], df['y'], c= kmeans.labels_.astype(float), s=50, alpha=0.5)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=50)
plt.show()
```

OUTPUT:

```
[[1.26633333 0.45466667]

 [1.30633333 1.41933333]

 [0.33066667 1.60833333]]
```

EXNO 5. 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 in Machine Learning.

The Find-S algorithm follows the steps written below:

1. Initialize 'h' to the most specific hypothesis.
2. The Find-S algorithm only considers the positive examples and eliminates negative examples. For each positive example, the algorithm checks for each attribute in the example. If the attribute value is the same as the hypothesis value, the algorithm moves on without any changes. But if the attribute value is different than the hypothesis value, the algorithm changes it to '?'.

Now that we are done with the basic explanation of the Find-S algorithm, let us take a look at how it works.

1. The process starts with initializing 'h' with the most specific hypothesis, generally, it is the first positive example in the data set.
2. We check for each positive example. If the example is negative, we will move on to the next example but if it is a positive example we will consider it for the next step.
3. We will check if each attribute in the example is equal to the hypothesis value.
4. If the value matches, then no changes are made.
5. If the value does not match, the value is changed to '?'.
6. We do this until we reach the last positive example in the data set.

Implementation of Find-S Algorithm

To understand the implementation, let us try to implement it to a smaller data set with a bunch of examples to decide if a person wants to go for a walk.

The concept of this particular problem will be on what days does a person likes to go on walk.

Time	**Weather**	**Temperature**	**Company**	**Humidity**	**Wind**	**Goes**
Morning	Sunny	Warm	Yes	Mild	Strong	Yes
Evening	Rainy	Cold	No	Mild	Normal	No
Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
Evening	Sunny	Cold	Yes	High	Strong	Yes

Looking at the data set, we have six attributes and a final attribute that defines the positive or negative example. In this case, yes is a positive example, which means the person will go for a walk.

So now, the general hypothesis is:

h_0 = {‘Morning’, ‘Sunny’, ‘Warm’, ‘Yes’, ‘Mild’, ‘Strong’}

This is our general hypothesis, and now we will consider each example one by one, but only the positive examples.

h_1= {‘Morning’, ‘Sunny’, ‘?’, ‘Yes’, ‘?’, ‘?’}

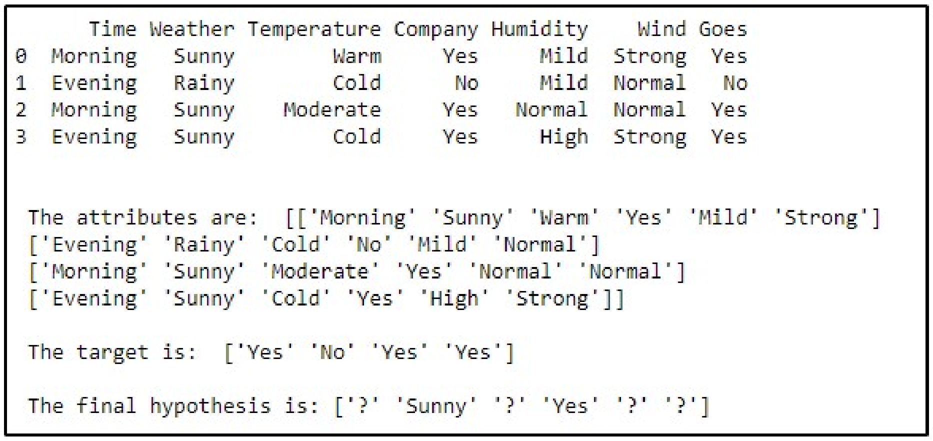
h_2 = {‘?’, ‘Sunny’, ‘?’, ‘Yes’, ‘?’, ‘?’}

We replaced all the different values in the general hypothesis to get a resultant hypothesis. Now that we know how the Find-S algorithm works, let us take a look at an implementation using Python.

CODING:

```
import pandas as pd
import numpy as np

#to read the data in the csv file
data = pd.read_csv("data.csv")
print(data,"n")

#making an array of all the attributes
d = np.array(data)[:,:-1]
print("n The attributes are: ",d)

#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("n The target is: ",target)

#training function to implement find-s algorithm
def train(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis

#obtaining the final hypothesis
print("n The final hypothesis is:",train(d,target))
```

Output:



```
      Time Weather Temperature Company Humidity    Wind Goes
0  Morning   Sunny        Warm     Yes     Mild  Strong  Yes
1  Evening   Rainy        Cold      No     Mild  Normal   No
2  Morning   Sunny    Moderate     Yes   Normal  Normal  Yes
3  Evening   Sunny        Cold     Yes     High  Strong  Yes


 The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]

 The target is:  ['Yes' 'No' 'Yes' 'Yes']

 The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

EXNO 6. Implement linear regression using python

The simple linear regression model is a line defined by coefficients estimated from training data.

Once the coefficients are estimated, we can use them to make predictions.

The equation to make predictions with a simple linear regression model is as follows:

*y = b0 + b1 * x*

Below is a function named **simple_linear_regression()** that implements the prediction equation to make predictions on a test dataset. It also ties together the estimation of the coefficients on training data from the steps above.

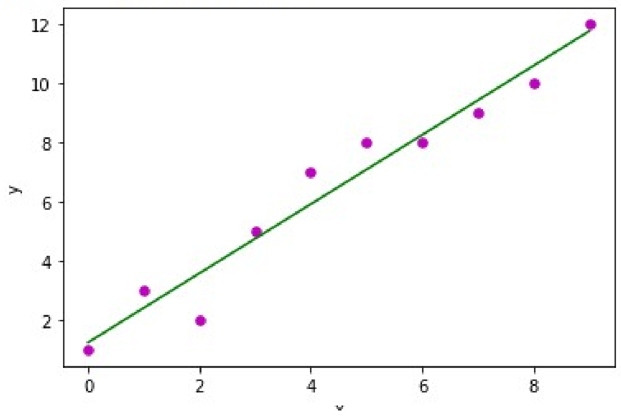
CODING:

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

def estimate_coef(x, y):
   # number of observations/points
   n = np.size(x)

   # mean of x and y vector
   m_x = np.mean(x)
   m_y = np.mean(y)

   # calculating cross-deviation and deviation about x
   SS_xy = np.sum(y*x) - n*m_y*m_x
   SS_xx = np.sum(x*x) - n*m_x*m_x

   # calculating regression coefficients
   b_1 = SS_xy / SS_xx
   b_0 = m_y - b_1*m_x

   return (b_0, b_1)

def plot_regression_line(x, y, b):
   # plotting the actual points as scatter plot
   plt.scatter(x, y, color = "m",
            marker = "o", s = 30)

   # predicted response vector
   y_pred = b[0] + b[1]*x

   # plotting the regression line
   plt.plot(x, y_pred, color = "g")

   # putting labels
   plt.xlabel('x')
   plt.ylabel('y')

   # function to show plot
   plt.show()
```

```
def main():
    # observations / data
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()
```

OUTPUT:

```
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```



EX NO 7. Implement Naïve Bayes theorem to classify the English text

Working example in Python

Text Analysis is a major application field for machine learning algorithms. **However the raw data, a sequence of symbols (i.e. strings) cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length.**

Problem Statement

As a working example, we will use some **text data** and we will build a **Naive Bayes** model to **predict** the **categories** of the **texts**. This is a **multi-class (20 classes) text classification problem**.

Let's start (I will walk you through). First, we will **load all the necessary libraries**:

```
import numpy as np, pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.metrics import confusion_matrix, accuracy_scoresns.set()

# use seaborn plotting style
```

Next, **let's load the data** (**training** and **test** sets):

```
# Load the dataset
data = fetch_20newsgroups()# Get the text categories
text_categories = data.target_names# define the training set
train_data = fetch_20newsgroups(subset="train", categories=text_categories)# define the test set
test_data = fetch_20newsgroups(subset="test", categories=text_categories)
```

Let's find out **how many classes** and **samples** we have:

```
print("We have {} unique classes".format(len(text_categories)))
print("We have {} training samples".format(len(train_data.data)))
print("We have {} test samples".format(len(test_data.data)))
```

The above prints:

```
We have 20 unique classes
We have 11314 training samples
We have 7532 test samples
```

So, this is a **20-class text classification problem** with n_train = **11314 training samples** (text sentences) and n_test = **7532 test samples** (text sentences).

EX NO 8. Implement an algorithm to demonstrate the significance of genetic algorithm

In this section, we will apply the genetic algorithm to a binary string-based optimization problem.

The problem is called OneMax and evaluates a binary string based on the number of 1s in the string. For example, a bitstring with a length of 20 bits will have a score of 20 for a string of all 1s.

Given we have implemented the genetic algorithm to minimize the objective function, we can add a negative sign to this evaluation so that large positive values become large negative values.

The *onemax()* function below implements this and takes a bitstring of integer values as input

```
# objective function

def onemax(x):

return -sum(x)
```

Next, we can configure the search.

The search will run for 100 iterations and we will use 20 bits in our candidate solutions, meaning the optimal fitness will be -20.0.

CODING

```
# genetic algorithm search of the one max optimization problem
from numpy.random import randint
from numpy.random import rand

# objective function
def onemax(x):
return -sum(x)

# tournament selection
def selection(pop, scores, k=3):
# first random selection
selection_ix = randint(len(pop))
for ix in randint(0, len(pop), k-1):
# check if better (e.g. perform a tournament)
if scores[ix] < scores[selection_ix]:
selection_ix = ix
return pop[selection_ix]

# crossover two parents to create two children
def crossover(p1, p2, r_cross):
# children are copies of parents by default
```

```
c1, c2 = p1.copy(), p2.copy()
# check for recombination
if rand() < r_cross:
# select crossover point that is not on the end of the string
pt = randint(1, len(p1)-2)
# perform crossover
c1 = p1[:pt] + p2[pt:]
c2 = p2[:pt] + p1[pt:]
return [c1, c2]

# mutation operator
def mutation(bitstring, r_mut):
for i in range(len(bitstring)):
# check for a mutation
if rand() < r_mut:
# flip the bit
bitstring[i] = 1 - bitstring[i]

# genetic algorithm
def genetic_algorithm(objective, n_bits, n_iter, n_pop, r_cross, r_mut):
# initial population of random bitstring
pop = [randint(0, 2, n_bits).tolist() for _ in range(n_pop)]
# keep track of best solution
best, best_eval = 0, objective(pop[0])
# enumerate generations
for gen in range(n_iter):
# evaluate all candidates in the population
scores = [objective(c) for c in pop]
# check for new best solution
for i in range(n_pop):
if scores[i] < best_eval:
best, best_eval = pop[i], scores[i]
print(">%d, new best f(%s) = %.3f" % (gen,  pop[i], scores[i]))
# select parents
selected = [selection(pop, scores) for _ in range(n_pop)]
# create the next generation
children = list()
for i in range(0, n_pop, 2):
# get selected parents in pairs
p1, p2 = selected[i], selected[i+1]
# crossover and mutation
for c in crossover(p1, p2, r_cross):
# mutation
mutation(c, r_mut)
# store for next generation
children.append(c)
# replace population
pop = children
return [best, best_eval]
```

```
# define the total iterations
n_iter = 100
# bits
n_bits = 20
# define the population size
n_pop = 100
# crossover rate
r_cross = 0.9
# mutation rate
r_mut = 1.0 / float(n_bits)
# perform the genetic algorithm search
best, score = genetic_algorithm(onemax, n_bits, n_iter, n_pop, r_cross, r_mut)
print('Done!')
print('f(%s) = %f' % (best, score))
```

Running the example will report the best result as it is found along the way, then the final best solution at the end of the search, which we would expect to be the optimal solution.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

In this case, we can see that the search found the optimal solution after about eight generations.

OUTPUT

```
>0, new best f([1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1]) = -14.000
>0, new best f([1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0]) = -15.000
>1, new best f([1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1]) = -16.000
>2, new best f([0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]) = -17.000
>2, new best f([1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -19.000
>8, new best f([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -20.000
Done!
f([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -20.000000
```

EXNO 9. Implement the finite words classification system using Back-propagation algorithm

This section applies the Backpropagation algorithm to the wheat seeds dataset.

The first step is to load the dataset and convert the loaded data to numbers that we can use in our neural network. For this we will use the helper function **load_csv()** to load the file, **str_column_to_float()** to convert string numbers to floats and **str_column_to_int()** to convert the class column to integer values.

Input values vary in scale and need to be normalized to the range of 0 and 1. It is generally good practice to normalize input values to the range of the chosen transfer function, in this case, the sigmoid function that outputs values between 0 and 1. The **dataset_minmax()** and **normalize_dataset()** helper functions were used to normalize the input values.

We will evaluate the algorithm using k-fold cross-validation with 5 folds. This means that 201/5=40.2 or 40 records will be in each fold. We will use the helper functions **evaluate_algorithm()** to evaluate the algorithm with cross-validation and **accuracy_metric()** to calculate the accuracy of predictions.

A new function named **back_propagation()** was developed to manage the application of the Backpropagation algorithm, first initializing a network, training it on the training dataset and then using the trained network to make predictions on a test dataset.

CODING

```
# Backprop on the Seeds Dataset
from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp

# Load a CSV file
def load_csv(filename):
dataset = list()
with open(filename, 'r') as file:
csv_reader = reader(file)
for row in csv_reader:
if not row:
continue
dataset.append(row)
return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
for row in dataset:
row[column] = float(row[column].strip())

# Convert string column to integer
```

```
def str_column_to_int(dataset, column):
class_values = [row[column] for row in dataset]
unique = set(class_values)
lookup = dict()
for i, value in enumerate(unique):
lookup[value] = i
for row in dataset:
row[column] = lookup[row[column]]
return lookup

# Find the min and max values for each column
def dataset_minmax(dataset):
minmax = list()
stats = [[min(column), max(column)] for column in zip(*dataset)]
return stats

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
for row in dataset:
for i in range(len(row)-1):
row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
dataset_split = list()
dataset_copy = list(dataset)
fold_size = int(len(dataset) / n_folds)
for i in range(n_folds):
fold = list()
while len(fold) < fold_size:
index = randrange(len(dataset_copy))
fold.append(dataset_copy.pop(index))
dataset_split.append(fold)
return dataset_split

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
correct = 0
for i in range(len(actual)):
if actual[i] == predicted[i]:
correct += 1
return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
folds = cross_validation_split(dataset, n_folds)
scores = list()
for fold in folds:
train_set = list(folds)
train_set.remove(fold)
```

```
train_set = sum(train_set, [])
test_set = list()
for row in fold:
row_copy = list(row)
test_set.append(row_copy)
row_copy[-1] = None
predicted = algorithm(train_set, test_set, *args)
actual = [row[-1] for row in fold]
accuracy = accuracy_metric(actual, predicted)
scores.append(accuracy)
return scores

# Calculate neuron activation for an input
def activate(weights, inputs):
activation = weights[-1]
for i in range(len(weights)-1):
activation += weights[i] * inputs[i]
return activation

# Transfer neuron activation
def transfer(activation):
return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
inputs = row
for layer in network:
new_inputs = []
for neuron in layer:
activation = activate(neuron['weights'], inputs)
neuron['output'] = transfer(activation)
new_inputs.append(neuron['output'])
inputs = new_inputs
return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output):
return output * (1.0 - output)

# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
for i in reversed(range(len(network))):
layer = network[i]
errors = list()
if i != len(network)-1:
for j in range(len(layer)):
error = 0.0
for neuron in network[i + 1]:
error += (neuron['weights'][j] * neuron['delta'])
errors.append(error)
```

```
else:
for j in range(len(layer)):
neuron = layer[j]
errors.append(expected[j] - neuron['output'])
for j in range(len(layer)):
neuron = layer[j]
neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])

# Update network weights with error
def update_weights(network, row, l_rate):
for i in range(len(network)):
inputs = row[:-1]
if i != 0:
inputs = [neuron['output'] for neuron in network[i - 1]]
for neuron in network[i]:
for j in range(len(inputs)):
neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
neuron['weights'][-1] += l_rate * neuron['delta']

# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
for epoch in range(n_epoch):
for row in train:
outputs = forward_propagate(network, row)
expected = [0 for i in range(n_outputs)]
expected[row[-1]] = 1
backward_propagate_error(network, expected)
update_weights(network, row, l_rate)

# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
network = list()
hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(n_hidden)]
network.append(hidden_layer)
output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(n_outputs)]
network.append(output_layer)
return network

# Make a prediction with a network
def predict(network, row):
outputs = forward_propagate(network, row)
return outputs.index(max(outputs))

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, l_rate, n_epoch, n_hidden):
n_inputs = len(train[0]) - 1
n_outputs = len(set([row[-1] for row in train]))
network = initialize_network(n_inputs, n_hidden, n_outputs)
train_network(network, train, l_rate, n_epoch, n_outputs)
predictions = list()
```

```
for row in test:
prediction = predict(network, row)
predictions.append(prediction)
return(predictions)

# Test Backprop on Seeds dataset
seed(1)
# load and prepare data
filename = 'seeds_dataset.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# normalize input variables
minmax = dataset_minmax(dataset)
normalize_dataset(dataset, minmax)
# evaluate algorithm
n_folds = 5
l_rate = 0.3
n_epoch = 500
n_hidden = 5
scores = evaluate_algorithm(dataset, back_propagation, n_folds, l_rate, n_epoch, n_hidden)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

A network with 5 neurons in the hidden layer and 3 neurons in the output layer was constructed. The network was trained for 500 epochs with a learning rate of 0.3. These parameters were found with a little trial and error, but you may be able to do much better.

Running the example prints the average classification accuracy on each fold as well as the average performance across all folds.

Back propagation and the chosen configuration achieved a mean classification accuracy of about 93% which is dramatically better than the Zero Rule algorithm that did slightly better than 28% accuracy.

OUTPUT

```
Scores: [92.85714285714286, 92.85714285714286, 97.61904761904762,
92.85714285714286, 90.47619047619048]
Mean Accuracy: 93.333%
```

EX NO 10. Implement a Candidate elimination algorithm

The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

- You can consider this as an extended form of Find-S algorithm.
- Consider both positive and negative examples.

Terms Used:

- **Concept learning:** Concept learning is basically learning task of the machine (Learn by Train data)
- **General Hypothesis:** Not Specifying features to learn the machine.
- **G = {'?', '?','?','?'…}:** Number of attributes
- **Specific Hypothesis:** Specifying features to learn machine (Specific feature)
- **S= {'pi','pi','pi'…}:** Number of pi depends on number of attributes.
- **Version Space:** It is intermediate of general hypothesis and Specific hypothesis. It not only just written one hypothesis but a set of all possible hypothesis based on training data-set.

Input data

Weather	**Temperature**	**Water**	**Forest**	**Humidity**	**Wind**	Output
Sunny	Warm	Warm	Same	Normal	Strong	Yes
Sunny	Warm	Warm	same	high	Strong	Yes
Rainy	Cold	Warm	Change	high	Strong	No
Sunny	Warm	Cold	Change	high	Strong	Yes

Coding :

```
import csv

with open("data.csv") as f:
  csv_file=csv.reader(f)
  data=list(csv_file)

  s=data[1][:-1]
  g=[['?' for i in range(len(s))] for j in range(len(s))]
```

```
for i in data:
    if i[-1]=="Yes":
        for j in range(len(s)):
            if i[j]!=s[j]:
                s[j]='?'
                g[j][j]='?'

    elif i[-1]=="No":
        for j in range(len(s)):
            if i[j]!=s[j]:
                g[j][j]=s[j]
            else:
                g[j][j]="?"
    print("\nSteps of Candidate Elimination Algorithm",data.index(i)+1)
    print(s)
    print(g)
gh=[]
for i in g:
    for j in i:
        if j!='?':
            gh.append(i)
            break
print("\nFinal specific hypothesis:\n",s)

print("\nFinal general hypothesis:\n",gh)
```

OUTPUT:

```
Steps of Candidate Elimination Algorithm 1
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 2
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 3
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 4
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 5
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
```

```
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final specific hypothesis:
 ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

Final general hypothesis:    S = ['sunny','warm',?,'strong', ?, ?]
```