COURSE LABORATORY MANUAL

A. LABORATORY OVERVIEW

Degree:	BE	Programme:	CS
Semester:	7	Academic Year:	2018-19
Laboratory Title:	**Machine Learning Laboratary**	Laboratory Code:	15CSL76
L-T-P-S:	1-0-2-0	Duration of SEE:	3 Hrs
Total Contact Hours:	40	SEE Marks:	80
Credits:	2	CIE Marks:	20
Lab Manual Author:	Harivinod N	Sign Harivinod.N	Date 30/06/2018
Checked By:	Pramod Kumar P M	Sign	Dt :

B. DESCRIPTION

1. PREREQUISITES:

- Creative thinking, sound mathematical insight and programming skills.
- Design and Analysis of Algorithms (15CS43)
- Design and Analysis of Algorithms Laboratory (15CSL47)
- Fundamentals of Data Structures (15CS33)
- Data Structures Laboratary (15CSL37)
- Computer Programming Laboratory (15CPL16/26)

2. BASE COURSE:

- Machine Learning (15CS73)

3. COURSE OUTCOMES:

At the end of the course, the student will be able to;

1. Understand the implementation procedures for the machine learning algorithms.
2. Design python programs for various learning algorithms.
3. Apply appropriate data sets to the machine learning algorithms.
4. Identify and apply machine learning algorithms to solve real world problems.

4. RESOURSES REQUIRED:

- Hardware resources
  - Desktop PC
  - Windows / Linux operating system
- Software resources
  - Python
  - Anaconda IDE with Spider
- Datasets from standard reporsitaries (Ex: https://archive.ics.uci.edu/ml/datasets.html)

Harivinod.N

Prepared by: Harivinod N   Checked by: Pramod Kumar P M   HOD

5. RELEVANCE OF THE COURSE:

- Project work (15CSP78, 15CSP85)

6. GENERAL INSTRUCTIONS:

- Implement the program in Python editor like Spider and demosnstrate the same.
- External practical examination.
  - All laboratory experiments are to be included
  - Students are allowed to pick one experiment from the lot.
  - Marks distribution: Procedure + Conduction + Viva: 20 + 50 +10 (80)
  - Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.

7. CONTENTS:

Expt No.	Title of the Experiments	RBT	CO
1	Implement and demonstratethe **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.	L3	CO 1,2,3,4
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.	L3	CO 1,2,3,4
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 toclassify a new sample.	L3	CO 1,2,3,4
4	Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets.	L3	CO 1,2,3,4
5	Write a program to implement the **naïve Bayesian classifier** for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.	L3	CO 1,2,3,4
6	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.	L3	CO 1,2,3,4
7	Write a 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. You can use Java/Python ML library classes/API.	L3	CO 1,2,3,4
8	Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using **k-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.	L3	CO 1,2,3,4
9	Write a program to implement **k-Nearest Neighbour algorithm** to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.	L3	CO 1,2,3,4
10	Implement the non-parametric **Locally Weighted Regression algorithm** in order to fit data points. Select appropriate data set for your experiment and draw graphs.	L3	CO 1,2,3,4
11	Open ended experiment - 1		
12	Open ended experiment - 2		

8. REFERENCE:

1. Tom M. Mitchell, Machine Learning, India Edition 2013, McGraw Hill Education.
2. Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning, 2nd edition, Springer series in statistics.
3. Ethem Alpaydın, Introduction to machine learning, second edition, MIT press.

-

C. EVALUATION SCHEME

For CBCS 2015 scheme:

1. Laboratory Components : 12 Marks
   (Record writing, Laboratory performance and Viva-voce)
2. Laboratory IA tests: 8 Marks
   (Minimum 2 IAs are mandatory. For the final IA test marks, average of the 2 IA test marks shall be considered and converted to maximum of 8)
3. Continuous Internal Evaluation (CIE) = 12+ 8 = 20 Marks
4. *SEE* : 80 Marks

-

D1. ARTICULATION MATRIX

Mapping of CO to PO

	POs											
COs	1	2	3	4	5	6	7	8	9	10	11	12
1. Understand the implementation procedures for the ML algorithms.	3	3	3	3	2	1	1	-	3	3	2	1
2. Design python programs for various learning algorithms.	3	3	3	3	3	1	-	-	3	2	1	1
3. Apply appropriate data sets to the machine learning algorithms.	3	3	3	3	2	2	-	-	3	1	-	-
4. Identify and apply machine learning algorithms to solve real world problems.	3	3	3	3	3	3	1	2	3	3	2	1

Note: Mappings in the Tables D1 (above) and D2 (below) are done by entering in the corresponding cell the Correllation Levels in terms of numbers. For Slight (Low): 1, Moderate (Medium): 2, Substantial (High): 3 and for no correllation: " - ".

D2. ARTICULATION MATRIX CO v/s PSO

Mapping of CO to PSO

	PSOs		
COs	1	2	3
1. Understand the implementation procedures for the ML algorithms.	3	-	-
2. Design python programs for various ML algorithms.	3	-	-
3. Apply appropriate data sets to the ML algorithms.	3	-	-
4. Identify and apply ML algorithms to solve real world problems.	3	-	-

-

E. EXPERIMENTS

1. EXPERIMENT NO: 1

2. TITLE: **FIND-S ALGORITHM**

3. LEARNING OBJECTIVES:
- Make use of Data sets in implementing the machine learning algorithms.
- Implement ML concepts and algorithms in Python

4. AIM:
- Implement and demonstratethe 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.

5. THEORY:
- The concept learning approach in machine learning, can be formulated as "Problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples".
- Find-S algorithm for concept learning is one of the most basic algorithms of machine learning.

Find-S Algorithm
1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
   For each attribute constraint a i in h :
      If the constraint a i in h is satisfied by x then do nothing
      Else replace a i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

- It is Guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples.
- Also Notice that negative examples are ignored.

Limitations of the Find-S algorithm:
- No way to determine if the only final hypothesis (found by Find-S) is consistent with data or there are more hypothesis that is consistent with data.
- Inconsistent sets of training data can mislead the finds algorithm as it ignores negative data samples.
- A good concept learning algorithm should be able to backtrack the choice of hypothesis found so that the resulting hypothesis can be improved over time. Unfortunately, Find-S provide no such method.

6. PROCEDURE / PROGRAMME :

FindS.py

```
import csv

def loadCsv(filename):
    lines = csv.reader(open(filename, "r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset, headers

def print_hypothesis(h):
    print('<',end=' ')
    for i in range(0,len(h)-1):
        print(h[i],end=',')
    print('>')
```

```
def findS():
    dataset,features=loadCsv('data11_sports6.csv')
    rows=len(dataset);
    cols=len(dataset[0]);

    flag = 0
    for x in range(0,rows):
        t=dataset[x]

        # Initialize h with first +ve sample
        if t[-1]=='1' and flag==0:
            flag=1
            h = dataset[x]
                # Update h with remaining +ve samples
        elif t[-1]=='1':
            for y in range(cols):
                if h[y]!=t[y]:
                    h[y]='?'
        #print("Training instance {0} the hypothesis is : ".format(x+1),end=' ')
        #print_hypothesis(h)

    print("The maximally specific hypothesis for a given training examples")
    #print(h)
    print_hypothesis(h)


findS()
```

7. RESULTS & CONCLUSIONS:

Result-1

Dataset: data11_tennis6.csv

```
Sky,AirTemp,Humidity,Wind,EnjoySport
sunny,warm,normal,strong,warm,same,1
sunny,warm,high,strong,warm,same,1
rainy,cold,high,strong,warm,change,0
sunny,warm,high,strong,cool,change,1
```

Output:

```
The Maximally Specific Hypothesis for a given Training Examples
< sunny,warm,?,strong,?,?,>
```

Result-2

Dataset: data12_tennis4.csv

```
Sky,AirTemp,Humidity,Wind,EnjoySport
sunny,hot,high,weak,1
sunny,hot,high,strong,1
overcast,hot,high,weak,1
rain,mild,high,weak,0
rain,cool,normal,weak,1
rain,cool,normal,strong,0
overcast,cool,normal,strong,1
sunny,cool,normal,weak,1
rain,mild,normal,weak,1
```

Output
The Maximally Specific Hypothesis for a given Training Examples
< ?,?,?,?,>

8. LEARNING OUTCOMES :

- Students will be able to apply Find-S algorithm to the real world problem and find the most specific hyposis from the training data.

9. APPLICATION AREAS:

- Classification based problems.

10. REMARKS:

-

1. EXPERIMENT NO: 2

2. TITLE: **CANDIDATE-ELIMINATION ALGORITHM**

3. LEARNING OBJECTIVES:
- Make use of Data sets in implementing the machine learning algorithms.
- Implement ML concepts and algorithms in Python

4. AIM:
- 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.

5. THEORY:
- The key idea in the Candidate-Elimination algorithm is to output a description of the set of all hypotheses consistent with the training examples.
- It computes the description of this set without explicitly enumerating all of its members.
- This is accomplished by using the more-general-than partial ordering and maintaining a compact representation of the set of consistent hypotheses.
- The algorithm represents the set of all hypotheses consistent with the observed training examples. This subset of all hypotheses is called the version space with respect to the hypothesis space H and the training examples D, because it contains all plausible versions of the target concept.
- A version space can be represented with its general and specific boundary sets.
- The Candidate-Elimination algorithm represents the version space by storing only its most general members G and its most specific members S.
- Given only these two sets S and G, it is possible to enumerate all members of a version space by generating hypotheses that lie between these two sets in general-to-specific partial ordering over hypotheses. Every member of the version space lies between these boundaries

Algorithm

1. Initialize G to the set of maximally general hypotheses in H
2. Initialize S to the set of maximally specific hypotheses in H
3. For each training example d, do
   - 3.1. If d is a positive example
     - Remove from G any hypothesis inconsistent with d ,
     - For each hypothesis s in S that is not consistent with d ,
       - Remove s from S
       - Add to S all minimal generalizations h of s such that h is consistent with d, and some member of G is more general than h
       - Remove from S, hypothesis that is more general than another hypothesis in S
   - 3.2. If d is a negative example
     - Remove from S any hypothesis inconsistent with d
     - For each hypothesis g in G that is not consistent with d
       - Remove g from G
       - Add to G all minimal specializations h of g such that h is consistent with d, and some member of S is more specific than h
     - Remove from G any hypothesis that is less general than another hypothesis in G

6. PROCEDURE / PROGRAMME :

```
import csv

def get_domains(examples):
    d = [set() for i in examples[0]]
    for x in examples:
        for i, xi in enumerate(x):
            d[i].add(xi)
    return [list(sorted(x)) for x in d]

def more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        mg = x == "?" or (x != "0" and (x == y or y == "0"))
        more_general_parts.append(mg)
    return all(more_general_parts)

def fulfills(example, hypothesis):
    # the implementation is the same as for hypotheses:
    return more_general(hypothesis, example)

def min_generalizations(h, x):
    h_new = list(h)
    for i in range(len(h)):
        if not fulfills(x[i:i+1], h[i:i+1]):
            h_new[i] = '?' if h[i] != '0' else x[i]
    return [tuple(h_new)]

def min_specializations(h, domains, x):
    results = []
    for i in range(len(h)):
        if h[i] == "?":
            for val in domains[i]:
                if x[i] != val:
                    h_new = h[:i] + (val,) + h[i+1:]
                    results.append(h_new)
        elif h[i] != "0":
            h_new = h[:i] + ('0',) + h[i+1:]
            results.append(h_new)
    return results

def generalize_S(x, G, S):
    S_prev = list(S)
    for s in S_prev:
        if s not in S:
            continue
        if not fulfills(x, s):
            S.remove(s)
            Splus = min_generalizations(s, x)
            ## keep only generalizations that have a counterpart in G
            S.update([h for h in Splus if any([more_general(g,h)
                                               for g in G])])
            ## remove hypotheses less specific than any other in S
            S.difference_update([h for h in S if
                                 any([more_general(h, h1)
                                      for h1 in S if h != h1])])
    return S
```

```
def specialize_G(x, domains, G, S):
    G_prev = list(G)
    for g in G_prev:
        if g not in G:
            continue
        if fulfills(x, g):
            G.remove(g)
            Gminus = min_specializations(g, domains, x)
            ## keep only specializations that have a conuterpart in S
            G.update([h for h in Gminus if any([more_general(h, s)
                                                for s in S])])
            ## remove hypotheses less general than any other in G
            G.difference_update([h for h in G if
                                 any([more_general(g1, h)
                                      for g1 in G if h != g1])])
    return G

def candidate_elimination(examples):
    domains = get_domains(examples)[:-1]
    n = len(domains)
    G = set([("?",)*n])
    S = set([("0",)*n])

    print("Maximally specific hypotheses - S ")
    print("Maximally general hypotheses  - G ")

    i=0
    print("\nS[0]:",str(S),"\nG[0]:",str(G))
    for xcx in examples:
        i=i+1
        x, cx = xcx[:-1], xcx[-1]  # Splitting data into attributes and decisions
        if cx=='Y': # x is positive example
            G = {g for g in G if fulfills(x, g)}
            S = generalize_S(x, G, S)
        else: # x is negative example
            S = {s for s in S if not fulfills(x, s)}
            G = specialize_G(x, domains, G, S)
        print("\nS[{0}]:".format(i),S)
        print("G[{0}]:".format(i),G)
    return

with open('data22_sports.csv')  as csvFile:
        examples = [tuple(line) for line in csv.reader(csvFile)]

candidate_elimination(examples)
```

7. RESULTS & CONCLUSIONS:

Result-1

Data: data21_sports.csv (Sky,AirTemp,Humidity,Wind,Water,Forecast,EnjoySport)

```
sunny,warm,normal,strong,warm,same,Y
sunny,warm,high,strong,warm,same,Y
rainy,cold,high,strong,warm,change,N
sunny,warm,high,strong,cool,change,Y
```

Output

```
Maximally specific hypotheses - S
Maximally general hypotheses  - G
```

S[0]: {('0', '0', '0', '0', '0', '0')}
G[0]: {('?', '?', '?', '?', '?', '?')}

S[1]: {('sunny', 'warm', 'normal', 'strong', 'warm', 'same')}
G[1]: {('?', '?', '?', '?', '?', '?')}

S[2]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}
G[2]: {('?', '?', '?', '?', '?', '?')}

S[3]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}
G[3]: {('?', 'warm', '?', '?', '?', '?'), ('sunny', '?', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'same')}

S[4]: {('sunny', 'warm', '?', 'strong', '?', '?')}
G[4]: {('?', 'warm', '?', '?', '?', '?'), ('sunny', '?', '?', '?', '?', '?')}

Result-2

Data: data22_shape.csv (Size,Color,Shape,Label)

big,red,circle,N
small,red,triangle,N
small,red,circle,Y
big,blue,circle,N
small,blue,circle,Y

Output

Maximally specific hypotheses - S
Maximally general hypotheses - G

S[0]: {('0', '0', '0')}
G[0]: {('?', '?', '?')}

S[1]: {('0', '0', '0')}
G[1]: {('?', '?', 'triangle'), ('?', 'blue', '?'), ('small', '?', '?')}

S[2]: {('0', '0', '0')}
G[2]: {('big', '?', 'triangle'), ('small', '?', 'circle'), ('?', 'blue', '?')}

S[3]: {('small', 'red', 'circle')}
G[3]: {('small', '?', 'circle')}

S[4]: {('small', 'red', 'circle')}
G[4]: {('small', '?', 'circle')}

S[5]: {('small', '?', 'circle')}
G[5]: {('small', '?', 'circle')}

8. LEARNING OUTCOMES :

- The students will be able to apply candidate elimination algorithm and output a description of the set of all hypotheses consistent with the training examples

9. APPLICATION AREAS:

- Classification based problems.

10. REMARKS:

-

1. EXPERIMENT NO: 3

2. TITLE: **ID3 ALGORITHM**

3. LEARNING OBJECTIVES:
- Make use of Data sets in implementing the machine learning algorithms.
- Implement ML concepts and algorithms in Python

4. 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 toclassify a new sample.

5. THEORY:
- ID3 algorithm is a basic algorithm that learns decision trees by constructing them topdown, beginning with the question "which attribute should be tested at the root of the tree?".
- To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected and used as the test at the root node of the tree.
- A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node (i.e., down the branch corresponding to the example's value for this attribute).
- The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.
- A simplified version of the algorithm, specialized to learning boolean-valued functions (i.e., concept learning), is described below.

Algorithm: ID3(Examples, TargetAttribute, Attributes)
Input: Examples are the training examples.
Targetattribute is the attribute whose value is to be predicted by the tree.
Attributes is a list of other attributes that may be tested by the learned decision tree.
Output: Returns a decision tree that correctly classiJies the given Examples
Method:
1. Create a Root node for the tree
2. If all Examples are positive, Return the single-node tree Root, with label = +
3. If all Examples are negative, Return the single-node tree Root, with label = -
4. If Attributes is empty,
   Return the single-node tree Root, with label = most common value of TargetAttribute in Examples
   Else
   A ← the attribute from Attributes that best classifies Examples
   The decision attribute for Root ←A
   For each possible value, vi, of A,
   Add a new tree branch below Root, corresponding to the test A = vi
   Let $Examples_{vi}$ be the subset of Examples that have value vi for A
   If $Examples_{vi}$ is empty Then below this new branch add a leaf node with label = most common value of TargetAttribute in Examples
   Else
   below this new branch add the subtree ID3($Examples_{vi}$, TargetAttribute, Attributes–{A})
   End
5. Return Root

6. PROCEDURE / PROGRAMME :

```
import math
import csv

def load_csv(filename):
    lines = csv.reader(open(filename, "r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset, headers

class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""      # NULL indicates children exists.
                              # Not Null indicates this is a Leaf Node

def subtables(data, col, delete):
    dic = {}
    coldata = [ row[col] for row in data]
    attr = list(set(coldata)) # All values of attribute retrived

    for k in attr:
        dic[k] = []

    for y in range(len(data)):
        key = data[y][col]
        if delete:
            del data[y][col]
        dic[key].append(data[y])

    return attr, dic

def entropy(S):
    attr = list(set(S))
    if len(attr) == 1: #if all are +ve/-ve then entropy = 0
        return 0

    counts = [0,0] # Only two values possible 'yes' or 'no'
    for i in range(2):
        counts[i] = sum( [1 for x in S if attr[i] == x] ) / (len(S) * 1.0)

    sums = 0
    for cnt in counts:
        sums += -1 * cnt * math.log(cnt, 2)
    return sums

def compute_gain(data, col):
    attValues, dic = subtables(data, col, delete=False)

    total_entropy = entropy([row[-1] for row in data])
    for x in range(len(attValues)):
        ratio = len(dic[attValues[x]]) / ( len(data) * 1.0)
        entro = entropy([row[-1] for row in dic[attValues[x]]])
        total_entropy -= ratio*entro

    return total_entropy
```

```
def build_tree(data, features):

    lastcol = [row[-1] for row in data]
    if (len(set(lastcol))) == 1:   # If all samples have same labels return that label
        node=Node("")
        node.answer = lastcol[0]
        return node

    n = len(data[0])-1
    gains = [compute_gain(data, col) for col in range(n) ]

    split = gains.index(max(gains)) # Find max gains and returns index
    node = Node(features[split])  # 'node' stores attribute selected
    #del (features[split])
    fea = features[:split]+features[split+1:]

    attr, dic = subtables(data, split, delete=True) # Data will be spilt in subtables
    for x in range(len(attr)):
        child = build_tree(dic[attr[x]], fea)
        node.children.append((attr[x], child))

    return node

def print_tree(node, level):
    if node.answer != "":
        print("   "*level, node.answer) # Displays leaf node yes/no
        return

    print("   "*level, node.attribute)  # Displays attribute Name
    for value, n in node.children:
        print("   "*(level+1), value)
        print_tree(n, level + 2)

def classify(node,x_test,features):
    if node.answer != "":
        print(node.answer)
        return

    pos = features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
          classify(n,x_test,features)

''' Main program '''
dataset, features = load_csv("data3.csv") # Read Tennis data
node = build_tree(dataset, features)     # Build decision tree

print("The decision tree for the dataset using ID3 algorithm is ")
print_tree(node, 0)

testdata, features = load_csv("data3_test.csv")
for xtest in testdata:
    print("The test instance :  ",xtest)
    print("The predicted label :  ", end="")
    classify(node,xtest,features)
```

7. RESULTS & CONCLUSIONS:

Training instances: 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
```

Testing instances: data3_test.csv

```
Outlook,Temperature,Humidity,Wind
rain,cool,normal,strong
sunny,mild,normal,strong
```

Output

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
     overcast
         yes
     rain
         Wind
             weak
                 yes
             strong
                 no
     sunny
         Humidity
             normal
                 yes
             high
                 no
The test instance    :  ['rain', 'cool', 'normal', 'strong']
The predicted label :  no
The test instance    :  ['sunny', 'mild', 'normal', 'strong']
The predicted label :  yes
```

8. LEARNING OUTCOMES :

- The student will be able 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 toclassify a new sample.

9. APPLICATION AREAS:

- Classification related prblem areas

10. REMARKS

1. EXPERIMENT NO: 4

2. TITLE: **BACKPROPAGATION ALGORITHM**

3. LEARNING OBJECTIVES:
   - Make use of Data sets in implementing the machine learning algorithms.
   - Implement ML concepts and algorithms in Python

4. AIM:
   - Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

5. THEORY:
   - Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples.
   - Algorithms such as BACKPROPAGATION gradient descent to tune network parameters to best fit a training set of input-output pairs.
   - ANN learning is robust to errors in the training data and has been successfully applied to problems such as interpreting visual scenes, speech recognition, and learning robot control strategies.

Backpropogation algorithm

1. Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.
2. Initialize each wi to some small random value (e.g., between -.05 and .05).
3. Until the termination condition is met, do

   For each training example <($x_1,\ldots x_n$),t>, do

   // Propagate the input forward through the network:

   a. Input the instance ($x_1, .., x_n$) to the n/w & compute the n/w outputs o_k for every unit

   // Propagate the errors backward through the network:

   b. For each output unit k, calculate its error term $\square_k$; $\square_k = o_k(1-o_k)(t_k-o_k)$

   c. For each hidden unit h, calculate its error term $\square_h$; $\square_h=o_h(1-o_h)$ $\square_k\, w_{h,k}$ $\square_k$

   d. For each network weight $w_{i,j}$ do; $w_{i,j}=w_{i,j}+\square w_{i,j}$ where $\square w_{i,j}=$ $\square$ $\square_j\, x_{i,j}$

6. PROCEDURE / PROGRAMME :

```
import numpy as np  # numpy is commonly used to process number array

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) # Features ( Hrs Slept, Hrs Studied)
y = np.array(([92], [86], [89]), dtype=float)       # Labels(Marks obtained)

X = X/np.amax(X,axis=0) # Normalize
y = y/100

def sigmoid(x):
   return 1/(1 + np.exp(-x))
def sigmoid_grad(x):
   return x * (1 - x)

# Variable initialization
epoch=1000              #Setting training iterations
eta =0.2                #Setting learning rate (eta)
input_neurons  = 2      #number of features in data set
hidden_neurons = 3      #number of hidden layers neurons
output_neurons = 1      #number of neurons at output layer
```

```
# Weight and bias - Random initialization
wh=np.random.uniform(size=(input_neurons,hidden_neurons))       # 2x3
bh=np.random.uniform(size=(1,hidden_neurons))                   # 1x3
wout=np.random.uniform(size=(hidden_neurons,output_neurons))  # 1x1
bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
  #Forward Propogation
    h_ip=np.dot(X,wh) + bh          # Dot product + bias
    h_act = sigmoid(h_ip)           # Activation function
    o_ip=np.dot(h_act,wout) + bout
    output = sigmoid(o_ip)

  #Backpropagation
    # Error at Output layer
    Eo = y-output                      # Error at o/p
    outgrad = sigmoid_grad(output)
    d_output = Eo* outgrad             # Errj=Oj(1-Oj)(Tj-Oj)

    # Error at Hidden later
    Eh = d_output.dot(wout.T)              # .T means transpose
    hiddengrad = sigmoid_grad(h_act)       # How much hidden layer wts contributed to error
    d_hidden = Eh * hiddengrad
    wout += h_act.T.dot(d_output) *eta     # Dotproduct of nextlayererror and currentlayerop
    wh += X.T.dot(d_hidden) *eta

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

7. RESULTS & CONCLUSIONS:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
[[0.89427812]
 [0.88503667]
 [0.89099058]]
```

8. LEARNING OUTCOMES :

- The student will be able to build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

9. APPLICATION AREAS:

- Speech recognition, Character recognition, Human Face recognition

10. REMARKS:

1. EXPERIMENT NO: 5

2. TITLE: **NAÏVE BAYESIAN CLASSIFIER**

3. LEARNING OBJECTIVES:

- Make use of Data sets in implementing the machine learning algorithms.
- Implement ML concepts and algorithms in Python

4. AIM:

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

5. THEORY:

Naive Bayes algorithm : Naive Bayes algorithm is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

$$\underbrace{P(c\,|\,x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x\,|\,c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c\,|\,X) = P(x_1\,|\,c)\times P(x_2\,|\,c)\times\cdots\times P(x_n\,|\,c)\times P(c)$$

where

P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
P(c) is the prior probability of class.
P(x|c) is the likelihood which is the probability of predictor given class.
P(x) is the prior probability of predictor.

The naive Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function f (x) can take on any value from some finite set V. A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values (a1, a2, ... ,an). The learner is asked to predict the target value, or classification, for this new instance.

The Bayesian approach to classifying the new instance is to assign the most probable target value,

v_{MAP}, given the attribute values ($a_1, a_2, \ldots, a_n$) that describe the instance.

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2 \ldots a_n)$$

We can use Bayes theorem to rewrite this expression as

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)}$$

$$= \underset{v_j \in V}{\operatorname{argmax}} P(a_1, a_2 \ldots a_n | v_j) P(v_j)$$

Now we could attempt to estimate the two terms in Equation (19) based on the training data. It is easy to estimate each of the P(vj) simply by counting the frequency with which each target value vj occurs in the training data.

The naive Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. In other words, the assumption is that given the target value of the instance, the probability of observing the conjunction $a_1, a_2, \ldots, a_n$, is just the product of the probabilities for the individual attributes: $P(a_1, a_2, \ldots, a_n | v_j) = \Pi_i P(a_i | v_j)$. Substituting this, we have the approach used by the naive Bayes classifier.

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j)$$

where v_{NB} denotes the target value output by the naive Bayes classifier.

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution. For example, suppose the training data contains a continuous attribute, x. We first segment the data by the class, and then compute the mean and variance of x in each class.

Let μ be the mean of the values in x associated with class Ck, and let $\sigma^2 k$ be the variance of the values in x associated with class Ck. Suppose we have collected some observation value v. Then, the probability distribution of v given a class Ck, p(x=v|Ck) can be computed by plugging v into the equation for a Normal distribution parameterized by μ and $\sigma^2 k$. *That is*

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Above method is adopted in our implementation of the program.

<u>Pima Indian diabetis dataset</u>

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

6. PROCEDURE / PROGRAMME :

```
import csv, random, math
import statistics as st

def loadCsv(filename):
	lines = csv.reader(open(filename, "r"));
	dataset = list(lines)
	for i in range(len(dataset)):
		dataset[i] = [float(x) for x in dataset[i]]
	return dataset

def splitDataset(dataset, splitRatio):
	testSize = int(len(dataset) * splitRatio);
	trainSet = list(dataset);
	testSet =  []
	while len(testSet) < testSize:
	#randomly pick an instance from training data
		index = random.randrange(len(trainSet));
		testSet.append(trainSet.pop(index))
	return [trainSet, testSet]

#Create a dictionary of classes 1 and 0 where the values are the
#instacnes belonging to each class

def separateByClass(dataset):
	separated = {}
	for i in range(len(dataset)):
		x = dataset[i]
		if (x[-1] not in separated):
			separated[x[-1]] = []
		separated[x[-1]].append(x)
	return separated

def compute_mean_std(dataset):
	mean_std = [ (st.mean(attribute), st.stdev(attribute))
		 for attribute in zip(*dataset)];  #zip(*res) transposes a matrix (2-d array/list)
	del mean_std[-1] # Exclude label
	return mean_std

def summarizeByClass(dataset):
	separated = separateByClass(dataset);
	summary = {} # to store mean and std of +ve and -ve instances
	for classValue, instances in separated.items():
		#summaries is a dictionary of tuples(mean,std) for each class value
		summary[classValue] = compute_mean_std(instances)
	return summary

#For continuous attributes p is estimated using Gaussion distribution
def estimateProbability(x, mean, stdev):
	exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
	return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent
```

```
def calculateClassProbabilities(summaries, testVector):
	p = {}
	#class and attribute information as mean and sd
	for classValue, classSummaries in summaries.items():
		p[classValue] = 1
		for i in range(len(classSummaries)):
			mean, stdev = classSummaries[i]
			x = testVector[i] #testvector's first attribute
			#use normal distribution
			p[classValue] *= estimateProbability(x, mean, stdev);
	return p

def predict(summaries, testVector):
	all_p = calculateClassProbabilities(summaries, testVector)
	bestLabel, bestProb = None, -1
	for lbl, p in all_p.items():#assigns that class which has he highest prob
		if bestLabel is None or p > bestProb:
			bestProb = p
			bestLabel = lbl
	return bestLabel

def perform_classification(summaries, testSet):
	predictions = []
	for i in range(len(testSet)):
		result = predict(summaries, testSet[i])
		predictions.append(result)
	return predictions

def getAccuracy(testSet, predictions):
	correct = 0
	for i in range(len(testSet)):
		if testSet[i][-1] == predictions[i]:
			correct += 1
	return (correct/float(len(testSet))) * 100.0

dataset = loadCsv('data51.csv');
print('Pima Indian Diabetes Dataset loaded...')
print('Total instances available :',len(dataset))
print('Total attributes present  :',len(dataset[0])-1)

print("First Five instances of dataset:")
for i in range(5):
   print(i+1 , ':' , dataset[i])

splitRatio = 0.2
trainingSet, testSet = splitDataset(dataset, splitRatio)
print('\nDataset is split into training and testing set.')
print('Training examples = {0} \nTesting examples  = {1}'.format(len(trainingSet),
                                                   len(testSet)))
summaries = summarizeByClass(trainingSet);
predictions = perform_classification(summaries, testSet)

accuracy = getAccuracy(testSet, predictions)
print('\nAccuracy of the Naive Baysian Classifier is :', accuracy)
```

7. RESULTS & CONCLUSIONS:

Sample Result

```
Pima Indian Diabetes Dataset loaded...
Total instances available : 768
Total attributes present  : 8
First Five instances of dataset:
1 : [6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]
2 : [1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0, 0.0]
3 : [8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0, 1.0]
4 : [1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0, 0.0]
5 : [0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0, 1.0]

Dataset is split into training and testing set.
Training examples = 615
Testing examples  = 153

Accuracy of the Naive Baysian Classifier is : 73.85
```

8. LEARNING OUTCOMES :

- The student will be able to apply naive baysian classifier for the relevent problem and analyse the results.

9. APPLICATION AREAS:

- Real time Prediction: Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- Multi class Prediction: This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- Text classification/ Spam Filtering/ Sentiment Analysis: Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
- Recommendation System: Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

10. REMARKS:

1. EXPERIMENT NO: 6

2. TITLE: **DOCUMENT CLASSIFICATION USING NAÏVE BAYESIAN CLASSIFIER**

3. LEARNING OBJECTIVES:

- Make use of Data sets in implementing the machine learning algorithms.
- Implement ML concepts and algorithms in Python

4. 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.

5. THEORY:

For the theoey of the naive bayesian classifier refer Experiment No. 5. Theory of performance anaysis analysis is ellaborated here.

Analysis of Document Classification

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

- For classification tasks, the terms true positives, true negatives, false positives, and false negatives compare the results of the classifier under test with trusted external judgments. The terms positive and negative refer to the classifier's prediction (sometimes known as the expectation), and the terms true and false refer to whether that prediction corresponds to the external judgment (sometimes known as the observation).
- Precision - Precision is the ratio of correctly predicted positive documents to the total predicted positive documents. High precision relates to the low false positive rate.

  Precision = (Σ True positive) / (Σ True positive + Σ False positive)
- Recall (Sensitivity) - Recall is the ratio of correctly predicted positive documents to the all observations in actual class.

  Recall = (Σ True positive) / (Σ True positive + Σ False negative)
- Accuracy - Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. Therefore, you have to look at other parameters to evaluate the performance of your model. For our model, we have got 0.803 which means our model is approx. 80% accurate.

  Accuracy = (Σ True positive + Σ True negative) / Σ Total population

6. PROCEDURE / PROGRAMME :

```
import pandas as pd
msg=pd.read_csv('data6.csv',names=['message','label']) #Tabular form data
print('Total instances in the dataset:',msg.shape[0])

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

print('\nThe message and its label of first 5 instances are listed below')
X5, Y5 = X[0:5], msg.label[0:5]
for x, y in zip(X5,Y5):
    print(x,',',y)

# Splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,Y)
print('\nDataset is split into Training and Testing samples')
print('Total training instances :', xtrain.shape[0])
print('Total testing  instances :', xtest.shape[0])

# Output of count vectoriser is a sparse matrix
# CountVectorizer - stands for 'feature extraction'
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain) #Sparse matrix
xtest_dtm  = count_vect.transform(xtest)
print('\nTotal features extracted using CountVectorizer:',xtrain_dtm.shape[1])

print('\nFeatures for first 5 training instances are listed below')
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
print(df[0:5])#tabular representation
#print(xtrain_dtm) #Same as above but sparse matrix representation

# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)

print('\nClassstification results of testing samples are given below')
for doc, p in zip(xtest, predicted):
   pred = 'pos' if p==1 else 'neg'
   print('%s -> %s ' % (doc, pred))

#printing accuracy metrics
from sklearn import metrics
print('\nAccuracy metrics')
print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))

print('Recall   :',metrics.recall_score(ytest,predicted),
        '\nPrecison :',metrics.precision_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
```

7. RESULTS & CONCLUSIONS:

Data set

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

Output

```
Total instances in the dataset: 18

The message and its label of first 5 instances are listed below
I love this sandwich , pos
This is an amazing place , pos
I feel very good about these beers , pos
This is my best work , pos
What an awesome view , pos

Dataset is split into Training and Testing samples
Total training instances : 13
Total testing  instances : 5

Total features extracted using CountVectorizer: 46

Features for first 5 training instances are listed below
   am  amazing  an  and  awesome  bad  ...  view  we  went  what  will  with
0   1        0   0    1        0    0  ...     0   0     0     0     0     0
1   0        0   0    0        0    0  ...     0   0     0     0     0     0
2   0        0   1    0        1    0  ...     1   0     0     1     0     0
3   0        1   1    0        0    0  ...     0   0     0     0     0     0
4   0        0   0    0        0    1  ...     0   0     0     0     0     0

Classstification results of testing samples are given below
This is an awesome place -> pos
I love this sandwich -> pos
I love to dance -> pos
This is my best work -> pos
I feel very good about these beers -> pos

Accuracy metrics
Accuracy of the classifer is 0.4
Recall   : 0.4
Precison : 1.0
Confusion matrix
```

```
[[0 0]
 [3 2]]
```

8. LEARNING OUTCOMES :

- The student will be able to apply naive baysian classifier for document classification and analyse the results.

9. APPLICATION AREAS:

- Applicable in document classification

10. REMARKS:

1. EXPERIMENT NO: 7

2. TITLE: **BAYESIAN NETWORK**

3. LEARNING OBJECTIVES:

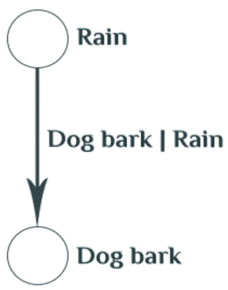
- Make use of Data sets in implementing the machine learning algorithms.
- Implement ML concepts and algorithms in Python

4. AIM:

- Write a 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. You can use Java/Python ML library classes/API.

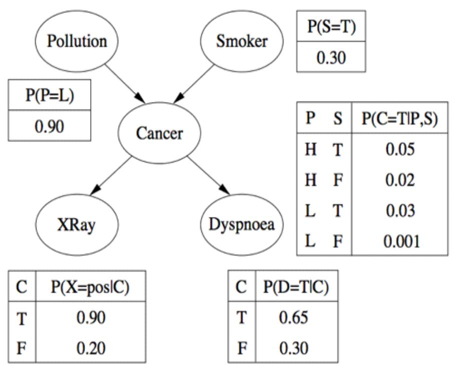
5. THEORY:

- Bayesian networks are very convenient for representing similar probabilistic relationships between multiple events.
- Bayesian networks as graphs - People usually represent Bayesian networks as directed graphs in which each node is a hypothesis or a random process. In other words, something that takes at least 2 possible values you can assign probabilities to. For example, there can be a node that represents the state of the dog (barking or not barking at the window), the weather (raining or not raining), etc.
- The arrows between nodes represent the conditional probabilities between them — how information about the state of one node changes the probability distribution of another node it's connected to.



6. PROCEDURE / PROGRAMME :

<u>Program for the Illustration of Baysian Belief networks using 5 nodes using Lung cancer data. (The Conditional probabilities are given)</u>




```
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination

#Define a Structure with nodes and edge
cancer_model = BayesianModel([('Pollution', 'Cancer'),
                              ('Smoker', 'Cancer'),
                              ('Cancer', 'Xray'),
                              ('Cancer', 'Dyspnoea')])

print('Baysian network nodes are:')
print('\t',cancer_model.nodes())
print('Baysian network edges are:')
print('\t',cancer_model.edges())

#Creation of Conditional Probability Table

cpd_poll = TabularCPD(variable='Pollution', variable_card=2,
                values=[[0.9], [0.1]])
cpd_smoke= TabularCPD(variable='Smoker', variable_card=2,
                values=[[0.3], [0.7]])
cpd_cancer= TabularCPD(variable='Cancer', variable_card=2,
```

```
                values=[[0.03, 0.05, 0.001, 0.02],
                       [0.97, 0.95, 0.999, 0.98]],
                        evidence=['Smoker', 'Pollution'],
                        evidence_card=[2, 2])
cpd_xray = TabularCPD(variable='Xray', variable_card=2,
                values=[[0.9, 0.2], [0.1, 0.8]],
                evidence=['Cancer'], evidence_card=[2])
cpd_dysp = TabularCPD(variable='Dyspnoea', variable_card=2,
                values=[[0.65, 0.3], [0.35, 0.7]],
                evidence=['Cancer'], evidence_card=[2])

# Associating the parameters with the model structure.
cancer_model.add_cpds(cpd_poll, cpd_smoke, cpd_cancer, cpd_xray, cpd_dysp)
print('Model generated by adding conditional probability disttributions(cpds)')

# Checking if the cpds are valid for the model.
print('Checking for Correctness of model : ', end='' )
print(cancer_model.check_model())

'''print('All local idependencies are as follows')
cancer_model.get_independencies()
'''
print('Displaying CPDs')
print(cancer_model.get_cpds('Pollution'))
print(cancer_model.get_cpds('Smoker'))
print(cancer_model.get_cpds('Cancer'))
print(cancer_model.get_cpds('Xray'))
print(cancer_model.get_cpds('Dyspnoea'))

##Inferencing with Bayesian Network

# Computing the probability of Cancer given smoke.
cancer_infer = VariableElimination(cancer_model)

print('\nInferencing with Bayesian Network');

print('\nProbability of Cancer given Smoker')
q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1})
print(q['Cancer'])

print('\nProbability of Cancer given Smoker,Pollution')
q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1,'Pollution': 1})
print(q['Cancer'])
```

<u>Program as per the Syllubus</u>

```
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

#Read the attributes
lines = list(csv.reader(open('data7_names.csv', 'r')));
attributes = lines[0]
#Read Cleveland Heart dicease data
heartDisease = pd.read_csv('data7_heart.csv', names = attributes)
heartDisease = heartDisease.replace('?', np.nan)
```

```
# Display the data
#print('Few examples from the dataset are given below')
#print(heartDisease.head())
#print('\nAttributes and datatypes')
#print(heartDisease.dtypes)

# Model Baysian Network
model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'), ('sex', 'trestbps'),
                       ('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),
                      ('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')])

# Learning CPDs using Maximum Likelihood Estimators
print('\nLearning CPDs using Maximum Likelihood Estimators...');
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network
print('\nInferencing with Bayesian Network:')
HeartDisease_infer = VariableElimination(model)

# Computing the probability of bronc given smoke.
print('\n1.Probability of HeartDisease given Age=28')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age': 28})
print(q['heartdisease'])

print('\n2. Probability of HeartDisease given chol (Cholestoral) =100')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'chol': 100})
print(q['heartdisease'])
```

7. RESULTS & CONCLUSIONS:

Dataset (For the program given in syllubus)

data7_names.csv (14 attributes)

```
age,sex,cp,trestbps,chol,fbs,restecg,thalach,exang,oldpeak,
slope,ca,thal,heartdisease
```

data7_heart.csv (5 instances out of 303)

```
63.0,1.0,1.0,145.0,233.0,1.0,2.0,150.0,0.0,2.3,3.0,0.0,6.0,0
67.0,1.0,4.0,160.0,286.0,0.0,2.0,108.0,1.0,1.5,2.0,3.0,3.0,2
67.0,1.0,4.0,120.0,229.0,0.0,2.0,129.0,1.0,2.6,2.0,2.0,7.0,1
37.0,1.0,3.0,130.0,250.0,0.0,0.0,187.0,0.0,3.5,3.0,0.0,3.0,0
41.0,0.0,2.0,130.0,204.0,0.0,2.0,172.0,0.0,1.4,1.0,0.0,3.0,0
```

Output

```
Learing CPDs using Maximum Likelihood Estimators...
Inferencing with Bayesian Network:
1.Probability of HeartDisease given Age=28
```

heartdisease	phi(heartdisease)
heartdisease_0	0.6791
heartdisease_1	0.1212
heartdisease_2	0.0810
heartdisease_3	0.0939
heartdisease_4	0.0247

```
2. Probability of HeartDisease given chol (Cholestoral) =100
```

heartdisease	phi(heartdisease)
heartdisease_0	0.5400
heartdisease_1	0.1533
heartdisease_2	0.1303
heartdisease_3	0.1259
heartdisease_4	0.0506

8. LEARNING OUTCOMES :

- The student will be able to apply baysian network for the medical data and demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

9. APPLICATION AREAS:

- Applicable in prediction and classification
- Gene Regulatory Networks
- Medicine
- Biomonitoring
- Document Classification
- Information Retrieval
- Semantic Search

10. REMARKS:

1. EXPERIMENT NO: 8

2. TITLE: **CLUSTERING BASED ON EM ALGORITHM AND K-MEANS**

3. LEARNING OBJECTIVES:
- Make use of Data sets in implementing the machine learning algorithms.
- Implement ML concepts and algorithms in Python

4. AIM: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

5. THEORY:

Expectation Maximization algorithm
- The basic approach and logic of this clustering method is as follows.
- Suppose we measure a single continuous variable in a large sample of observations. Further, suppose that the sample consists of two clusters of observations with different means (and perhaps different standard deviations); within each sample, the distribution of values for the continuous variable follows the normal distribution.
- The goal of EM clustering is to estimate the means and standard deviations for each cluster so as to maximize the likelihood of the observed data (distribution).
- Put another way, the EM algorithm attempts to approximate the observed distributions of values based on mixtures of different distributions in different clusters. The results of EM clustering are different from those computed by k-means clustering.
- The latter will assign observations to clusters to maximize the distances between clusters. The EM algorithm does not compute actual assignments of observations to clusters, but classification probabilities.
- In other words, each observation belongs to each cluster with a certain probability. Of course, as a final result we can usually review an actual assignment of observations to clusters, based on the (largest) classification probability.

K means Clustering
- The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement.
- The algorithm works as follows:
    1. First we initialize k points, called means, randomly.
    2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that mean so far.
    3. We repeat the process for a given number of iterations and at the end, we have our clusters.
- The "points" mentioned above are called means, because they hold the mean values of the items categorized in it. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set (if for a feature x the items have values in [0,3], we will initialize the means with values for x at [0,3]).
- Pseudocode:
    1. Initialize k means with random values
    2. For a given number of iterations:

        Iterate through items:

            Find the mean closest to the item

            Assign item to mean

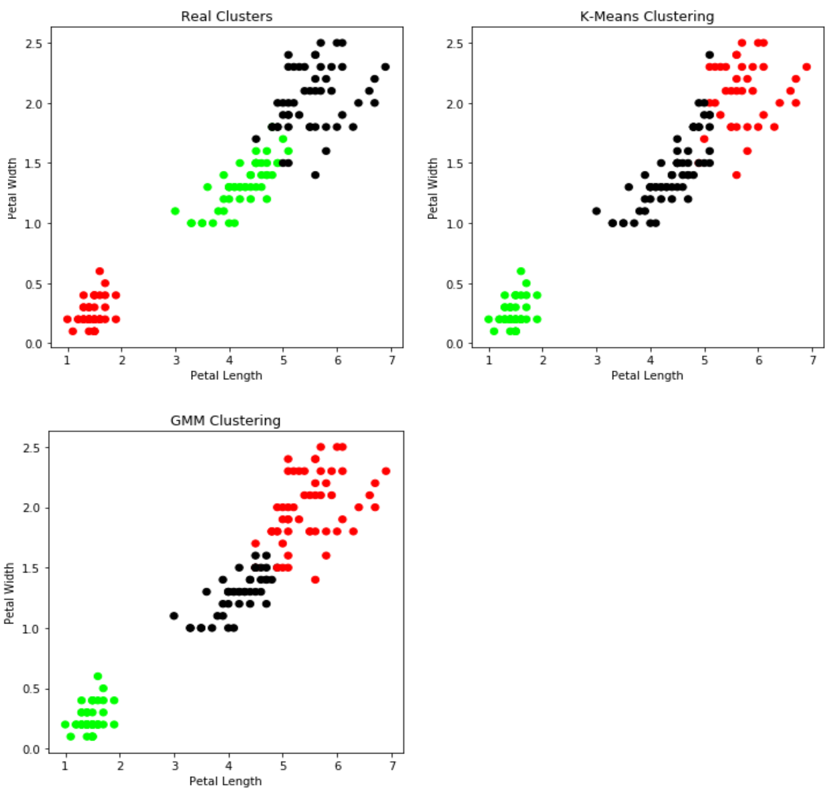
            Update mean

6. PROCEDURE / PROGRAMME :

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

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

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

# # Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications using Petal features
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
# Plot the Models Classifications
plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# General EM for GMM
from sklearn import preprocessing
# transform your data such that its distribution will have a
# mean value 0 and standard deviation of 1.
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)


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

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

7. RESULTS & CONCLUSIONS:

Sample Output



Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.

8. LEARNING OUTCOMES :

- The students will be apble to apply EM algorithm and k-Means algorithm for clustering and anayse the results.

9. APPLICATION AREAS:

- Text mining
- Pattern recognition
- Image analysis
- Web cluster engines

10. REMARKS:

1. EXPERIMENT NO: 9

2. TITLE: **K-NEAREST NEIGHBOUR**

3. LEARNING OBJECTIVES:
- Make use of Data sets in implementing the machine learning algorithms.
- Implement ML concepts and algorithms in Python

4. AIM:
- Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

5. THEORY:
- K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.
- It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data.
- Algorithm

  Input: Let m be the number of training data samples. Let p be an unknown point.

  Method:
  1. Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).
  2. for i=0 to m

     Calculate Euclidean distance d(arr[i], p).
  3. Make set S of K smallest distances obtained. Each of these distances correspond to an already classified data point.
  4. Return the majority label among S.

6. PROCEDURE / PROGRAMME :

```
# import the required packages
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets

# Load dataset
iris=datasets.load_iris()
print("Iris Data set loaded...")

# Split the data into train and test samples
x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.1)
print("Dataset is split into training and testing...")
print("Size of trainng data and its label",x_train.shape,y_train.shape)
print("Size of trainng data and its label",x_test.shape, y_test.shape)

# Prints Label no. and their names
for i in range(len(iris.target_names)):
    print("Label", i , "-",str(iris.target_names[i]))
```

```
# Create object of KNN classifier
classifier = KNeighborsClassifier(n_neighbors=1)

# Perform Training
classifier.fit(x_train, y_train)
# Perform testing
y_pred=classifier.predict(x_test)

# Display the results
print("Results of Classification using K-nn with K=1 ")
for r in range(0,len(x_test)):
   print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r]), " Predicted-label:",
                                                                   str(y_pred[r]))
print("Classification Accuracy :" , classifier.score(x_test,y_test));

#from sklearn.metrics import classification_report, confusion_matrix
#print('Confusion Matrix')
#print(confusion_matrix(y_test,y_pred))
#print('Accuracy Metrics')
#print(classification_report(y_test,y_pred))
```

7. RESULTS & CONCLUSIONS:

Result-1

```
Iris Data set loaded...
Dataset is split into training and testing samples...
Size of trainng data and its label (135, 4) (135,)
Size of trainng data and its label (15, 4) (15,)
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=1
 Sample:  [4.4 3.  1.3 0.2]  Actual-label: 0   Predicted-label:  0
 Sample:  [5.1 2.5 3.  1.1]  Actual-label: 1   Predicted-label:  1
 Sample:  [6.1 2.8 4.  1.3]  Actual-label: 1   Predicted-label:  1
 Sample:  [6.  2.7 5.1 1.6]  Actual-label: 1   Predicted-label:  2
 Sample:  [6.7 2.5 5.8 1.8]  Actual-label: 2   Predicted-label:  2
 Sample:  [5.1 3.8 1.5 0.3]  Actual-label: 0   Predicted-label:  0
 Sample:  [6.7 3.1 4.4 1.4]  Actual-label: 1   Predicted-label:  1
 Sample:  [4.8 3.4 1.6 0.2]  Actual-label: 0   Predicted-label:  0
 Sample:  [5.1 3.5 1.4 0.3]  Actual-label: 0   Predicted-label:  0
 Sample:  [5.4 3.7 1.5 0.2]  Actual-label: 0   Predicted-label:  0
 Sample:  [5.7 2.8 4.1 1.3]  Actual-label: 1   Predicted-label:  1
 Sample:  [4.5 2.3 1.3 0.3]  Actual-label: 0   Predicted-label:  0
 Sample:  [4.4 2.9 1.4 0.2]  Actual-label: 0   Predicted-label:  0
 Sample:  [5.1 3.5 1.4 0.2]  Actual-label: 0   Predicted-label:  0
 Sample:  [6.2 3.4 5.4 2.3]  Actual-label: 2   Predicted-label:  2
Classification Accuracy : 0.93
```

Result-2

Iris Data set loaded...
Dataset is split into training and testing samples...
Size of trainng data and its label (135, 4) (135,)
Size of trainng data and its label (15, 4) (15,)
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=1
Sample: [6.5 3. 5.5 1.8] Actual-label: 2 Predicted-label: 2
Sample: [5.7 2.8 4.1 1.3] Actual-label: 1 Predicted-label: 1
Sample: [6.6 3. 4.4 1.4] Actual-label: 1 Predicted-label: 1
Sample: [6.9 3.1 5.1 2.3] Actual-label: 2 Predicted-label: 2
Sample: [5.1 3.8 1.9 0.4] Actual-label: 0 Predicted-label: 0
Sample: [7.2 3.2 6. 1.8] Actual-label: 2 Predicted-label: 2
Sample: [5.5 2.6 4.4 1.2] Actual-label: 1 Predicted-label: 1
Sample: [6. 2.9 4.5 1.5] Actual-label: 1 Predicted-label: 1
Sample: [5.1 3.7 1.5 0.4] Actual-label: 0 Predicted-label: 0
Sample: [5.2 3.4 1.4 0.2] Actual-label: 0 Predicted-label: 0
Sample: [5. 3.5 1.6 0.6] Actual-label: 0 Predicted-label: 0
Sample: [4.9 3.1 1.5 0.1] Actual-label: 0 Predicted-label: 0
Sample: [5. 3. 1.6 0.2] Actual-label: 0 Predicted-label: 0
Sample: [5.7 3. 4.2 1.2] Actual-label: 1 Predicted-label: 1
Sample: [5.8 2.7 5.1 1.9] Actual-label: 2 Predicted-label: 2
Classification Accuracy : 1.0

8. LEARNING OUTCOMES :

- The student will be able to implement k-Nearest Neighbour algorithm to classify the iris data set and Print both correct and wrong predictions.

9. APPLICATION AREAS:

- Recommender systems
- Classification problems

10. REMARKS:

1. EXPERIMENT NO: 10

2. TITLE: **LOCALLY WEIGHTED REGRESSION ALGORITHM**

3. LEARNING OBJECTIVES:
   - Make use of Data sets in implementing the machine learning algorithms.
   - Implement ML concepts and algorithms in Python

4. 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.

5. THEORY:
   - Given a dataset X, y, we attempt to find a linear model h(x) that minimizes residual sum of squared errors. The solution is given by Normal equations.
   - Linear model can only fit a straight line, however, it can be empowered by polynomial features to get more powerful models. Still, we have to decide and fix the number and types of features ahead.
   - Alternate approach is given by locally weighted regression.
   - Given a dataset X, y, we attempt to find a model h(x) that minimizes residual sum of weighted squared errors.
   - The weights are given by a kernel function which can be chosen arbitrarily and in my case I chose a Gaussian kernel.
   - The solution is very similar to Normal equations, we only need to insert diagonal weight matrix W.

Algorithm

```
def local_regression(x0, X, Y, tau):
        # add bias term
        x0 = np.r_[1, x0]
        X = np.c_[np.ones(len(X)), X]

        # fit model: normal equations with kernel
        xw = X.T * radial_kernel(x0, X, tau)
        beta = np.linalg.pinv(xw @ X) @ xw @ Y

        # predict value
        return x0 @ beta
def radial_kernel(x0, X, tau):
        return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
```

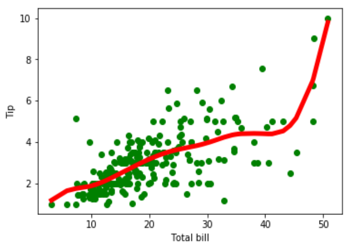
6. PROCEDURE / PROGRAMME :

```
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(np.eye((m))) # eye - identity matrix
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights
```

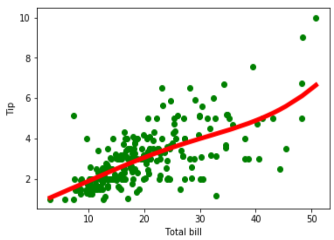
```
def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

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

def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    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();

# load data points
data = pd.read_csv('data10_tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips data
tip  = np.array(data.tip)

mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols

ypred = localWeightRegression(X,mtip,0.5) # increase k to get smooth curves
graphPlot(X,ypred)
```

7. RESULTS & CONCLUSIONS:

Regession with parameter k = 3



Regession with parameter k = 9



8. LEARNING OUTCOMES :

- To understand and implement linear regression and analyse the results with change in the parameters

9. APPLICATION AREAS:

- Demand anaysis in business
- Optimization of business processes
- Forecasting

10. REMARKS