VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM

LABORATORY MANUAL

COURSE CODE: 21AIL66

COURSE NAME: MACHINE LEARNING LABORATORY

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

ACHARYA INSTITUTE OF TECHNOLOGY, BANGALORE

2023-2024

COMPILED BY

Dr. S Anu Pallavi
Assistant Professor Grade-I

Vision & Mission of the Institute

Vision of the Institute

Acharya Institute of Technology, committed to the cause of value-based education in all disciplines, envisions itself as a fountainhead of innovative human enterprise, with inspirational initiatives for Academic Excellence.

Mission of the institute

Acharya Institute of Technology strives to provide excellent academic ambiance to the students for achieving global standards of technical education, foster intellectual and personal development, meaningful research and ethical service to sustainable societal needs.

Motto of the institute

“Nurturing Aspirations Supporting Growth”

Vision & Mission of the Department

Vision of the Department

To establish as a centre of excellence in moulding young professionals in the domain of Artificial Intelligence and Machine Learning who are capable of contributing to the wellness of the society as well as the industry.

Mission of the Department

1. To nurture the students with multi-disciplinary skills to be able to build sustainable solutions to engineering problems which in turn helps them to grab dynamic and promising career in the industry.
2. To mould energetic and committed engineers by inculcating moral values of professional ethics, social concerns, environment protection, sustainable solutions and life-long learning.
3. To facilitate research cooperation in the fields of artificial intelligence, machine learning, and data analytics by handholding with national and international research organizations, thus enabling the students to analyse and apply various levels of learning.

Program Outcomes (POs)

Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. 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.
12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Educational Objectives (PEOs)

Engineering Graduates will

1. Have a strong foundation in key concepts and techniques related to artificial intelligence and its related fields, thus enabling them to analyze complex problems in various domains and apply AI and ML techniques to develop innovative and effective solutions.
2. Be conscious of the ethical implications of AI/ML technologies, be aware of potential biases and ensure fairness, transparency, and accountability and demonstrate a commitment to develop AI/ML applications for the betterment of society.
3. Possess effective communication skills and be capable of working collaboratively in multidisciplinary teams to address real-world challenges.
4. Have an entrepreneurial mindset and be equipped with the skills and knowledge to create innovative AI/ML solutions and potentially start their own ventures.
5. Demonstrate a commitment to lifelong learning and stay updated with advancements in AI and ML technologies and industry trends to emerge as leaders in their fields of expertise and domain that support service and economic development of the society.

Program Specific Outcomes (PSOs)

Engineering Graduates will be able to:

1. Apply the Artificial Intelligence and Data Analytics skills in the areas of Disaster Management, Health Care, Education, Agriculture, Transportation, Environment, Society and in other multi-disciplinary areas.
2. Analyse and demonstrate the knowledge of Human cognition, Artificial Intelligence, Machine Learning and Data engineering in terms of real-world problems to meet the challenges of the future.
3. Develop computational knowledge and project management skills using innovative tools and techniques to solve problems in the areas related to Deep Learning, Machine learning, Artificial Intelligence which will lead to lifelong learning.

VI Semester

MACHINE LEARNING LABORATORY			
Course Code	21AIL66	CIE Marks	50
Teaching Hours/Week(L:T:P:S)	0:0:2:0	SEE Marks	50
Total Hours of Pedagogy	24	Total Marks	100
Credits	1	Exam Hours	03

Course Learning Objectives:

CLO 2. To learn and understand the Importance Machine learning Algorithms
CLO 3. Compare and contrast the learning techniques like ANN approach, Bayesian learning and reinforcement learning.
CLO 4. Able to solve and analyse the problems on ANN, Instance based learning and Reinforcement learning techniques.
CLO 5. To impart the knowledge of clustering and classification Algorithms for predictions and evaluating Hypothesis.

	Prerequisite
	• Students should be familiarized about Python installation and setting Python environment • Usage and installation of Anaconda should be introduced https://www.anaconda.com/products/individual • Should have the knowledge about Probability theory,Statistics theory and linear Algebra. • Should have the knowledge of numpy,pandas,scikit-learn and scipy library packages.
Sl. No.	**PART A – List of problems for which student should develop program and execute in the Laboratory**
1	Aim: Illustrate and Demonstrate the working model and principle of Find-S algorithm. Program: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples. **Text Book 1: Ch2**
2	Aim: Demonstrate the working model and principle of candidate elimination algorithm. Program: 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. **Text Book 1: Ch2** Reference: https://www.youtube.com/watch?v=tfpAm4kxGQI
3	Aim: To construct the Decision tree using the training data sets under supervised learning concept. Program: Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample. **Text Book 1: Ch 3**
4	Aim: To understand the working principle of Artificial Neural network with feed forward and feed backward principle. Program: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets. **Text Book 1: Ch 4**

5	Aim: Demonstrate the text classifier using Naïve bayes classifier algorithm. Program: Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. **Text Book 1: Ch6**
6	Aim: Demonstrate and Analyse the results sets obtained from Bayesian belief network Principle. Program:- 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 Python ML library classes/API. **Text Book 1: Ch 6**
7	Aim: Implement and demonstrate the working model of K-means clustering algorithm with Expectation Maximization Concept. Program: 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 Python ML library classes/API in the program. **Text Book 1: Ch 8**
8	Aim: Demonstrate and analyse the results of classification based on KNN Algorithm. Program: 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. **Text Book 1: Ch 8**
9	Aim: Understand and analyse the concept of Regression algorithm techniques. Program: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs. **Text Book 1: Ch8**
10	Aim: Implement and demonstrate classification algorithm using Support vector machine Algorithm. Program: Implement and demonstrate the working of SVM algorithm for classification. **Text Book 2: Ch6**
Pedagogy	For the above experiments the following pedagogy can be considered. Problem based learning, Active learning, MOOC, Chalk & Talk
PART B	
	A problem statement for each batch is to be generated in consultation with the co-examiner and student should develop an algorithm, program and execute the Program for the given problem with appropriate outputs.

Course Outcomes: At the end of the course the student will be able to:

CO 1. Understand the Importance of different classification and clustering algorithms.
CO 2. Demonstrate the working of various algorithms with respect to training and test data sets.
CO 3. Illustrate and analyze the principles of Instance based and Reinforcement learning techniques.
CO 4. Elicit the importance and Applications of Supervised and unsupervised machine learning.
CO 5. Compare and contrast the Bayes theorem principles and Q learning approach.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks). A student

shall be deemed to have satisfied the academic requirements and earned the credits allotted to each course. The student has to secure not less than 35% (18 Marks out of 50) in the semester-end examination (SEE).

Continuous Internal Evaluation (CIE):

CIE marks for the practical course is **50 Marks**.

The split-up of CIE marks for record/ journal and test are in the ratio **60:40**.

- Each experiment to be evaluated for conduction with observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments designed by the faculty who is handling the laboratory session and is made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- Total marks scored by the students are scaled downed to 30 marks (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- Department shall conduct 02 tests for 100 marks, the first test shall be conducted after the 8th week of the semester and the second test shall be conducted after the 14th week of the semester.
- In each test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability. Rubrics suggested in Annexure-II of Regulation book
- The average of 02 tests is scaled down to **20 marks** (40% of the maximum marks). The Sum of scaled-down marks scored in the report write-up/journal and average marks of two tests is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

- SEE marks for the practical course is 50 Marks.
- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the University
- All laboratory experiments are to be included for practical examination.
- (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. **OR** based on the course requirement evaluation rubrics shall be decided jointly by examiners.
- Students can pick one question (experiment) from the questions lot prepared by the internal /external examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.
- General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)
- *Students can pick one experiment from the questions lot of PART A with equal choice to all the students in a batch. For PART B examiners should frame a question for each batch, student should*

develop an algorithm, program, execute and demonstrate the results with appropriate output for the given problem.

- *Weightage of marks for PART A is 80% and for PART B is 20%. General rubrics suggested to be followed for part A and part B.*
- Change of experiment is allowed only once and Marks allotted to the procedure part to be made zero (Not allowed for Part B).
- The duration of SEE is 03 hours
- Rubrics suggested in Annexure-II of Regulation book

Text Books:

1. Tom M Mitchell, "Machine Lerning",1st Edition, McGraw Hill Education, 2017.
2. Nello Cristianini, John Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, Cambridge University Press, 2013
3. Allen B. Downey, "Think Python: How to Think Like a Computer Scientist", 2nd Edition, Green Tea Press, 2015. (Available under CC-BY-NC license at http://greenteapress.com/thinkpython2/thinkpython2.pdf)

Suggested Web Links / E Resource

1. https://www.kaggle.com/general/95287
2. https://web.stanford.edu/~hastie/Papers/ESLII.pdf

CONTENT

Program number	Program list	Page no
1	For a given set of training data examples stored in a . CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.	6
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.	8
3	Write a program to demonstrate the working of the decision tree-based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.	11
4	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.	17
5	Write a program to implement the naive Bayesian classifier for a sample training data set stored as a . CSV file. Compute the accuracy of the classifier, considering a few test data sets.	21
6	Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using a standard Heart Disease Data Set. You can use Python ML library classes/API.	25
7	Apply the EM algorithm to cluster a set of data stored in a . CSV file. Use the same data set for clustering using the k-means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Python ML library classes/API to the program.	27
8	Write a program to implement the 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.	30
9	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.	32
10	Implement and demonstrate the working of the SVM algorithm for classification.	34

PROGRAM NO.1

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

Aim:

To implement the Find-S algorithm is to learn a maximally specific hypothesis from a given set of training data examples. The Find-S algorithm produces a hypothesis that is consistent with all positive training examples while being as specific as possible.

Objective:

The objective of the Find-S algorithm is to find the most specific hypothesis that fits all positive examples in the training dataset. This specific hypothesis is then used to make predictions on new unseen examples. By maximizing specificity, the algorithm aims to generalize well to unseen data while avoiding overfitting.

Pseudo code for Find S algorithm:

1. Initialize hypothesis h to the most specific hypothesis in H
2. For each positive training instance x
   a. For each attribute constraint ai in h
      i. If the constraint ai is satisfied by x
         - Do nothing
      ii. Else
         - Replace ai in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

Training Examples:

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

Python code:

```
import pandas as pd
df = pd.read_csv('/content/ML LAB/enjoysport.csv')
df.values.tolist()
print(df)
n = len(a[0]) - 1
S = ['?'] * n  # Initialize with '?'
print("Initial hypothesis:", S)

print("FIND S ALGORITHM")
S = a[0][:-1]

for i in range(len(a)):
    if a[i][n] == "yes":
        for j in range(n):
            if a[i][j] != S[j]:
                S[j] = '?'
    print("\nTraining example no {0}, Hypothesis is: {1}".format(i + 1, S))

print("\nMaximally specific hypothesis is:", S)
```

Output:

```
sky airtemp humidity    wind water forcast enjoysport
0  sunny    warm   normal  strong  warm    same        yes
1  sunny    warm     high  strong  warm    same        yes
2  rainy    cold     high  strong  warm  change         no
3  sunny    warm     high  strong  cool  change        yes
Initial hypothesis: ['?', '?', '?', '?', '?', '?']
FIND S ALGORITHM

Training example no 1, Hypothesis is: ['sunny', 'warm', 'normal', 'strong',
'warm', 'same']

Training example no 2, Hypothesis is: ['sunny', 'warm', '?', 'strong',
'warm', 'same']

Training example no 3, Hypothesis is: ['sunny', 'warm', '?', 'strong',
'warm', 'same']

Training example no 4, Hypothesis is: ['sunny', 'warm', '?', 'strong', '?',
'?']

Maximally specific hypothesis is: ['sunny', 'warm', '?', 'strong', '?', '?']
```

PROGRAM NO. 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.

Aim:

To implement the Candidate-Elimination algorithm for machine learning classification tasks using a given set of training data examples stored in a CSV file.

Objective:

The objective of this exercise is to implement the Candidate-Elimination algorithm to learn a hypothesis space consistent with the training data by iteratively refining the version space through analysis of each training example.

Pseudo code for Candidate-Elimination algorithm:

1. Read the training data from the CSV file.
2. Initialize the specific and general hypotheses.
   - Set the initial specific hypothesis S0 to the first training example without the target attribute.
   - Set the initial general hypothesis G0 to '?' for each attribute.
3. Iterate through each training example in the dataset.
   a. If the target attribute of the current example is 'yes':
      - Update the specific hypothesis S by setting '?' for any attribute that does not match the current example.
      - Update the general hypothesis G by removing any inconsistent hypotheses from the version space.
   b. If the target attribute of the current example is 'no':
      - Update the general hypothesis G by replacing any attribute that matches the current example with the corresponding attribute from the specific hypothesis.
      - Add the updated general hypothesis to the version space if it is not already present.
4. Output the final specific and general hypotheses after processing all training examples.

Training Examples:

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

Python Code:

```
import pandas as pd
df = pd.read_csv('/content/ML LAB/enjoysport.csv')

a = df.values.tolist()
print(df)
n=len(a[0])-1
print("\n The initial value of hypothesis: ")
s = ['0'] * n
g = ['?'] * n
print ("\n The most specific hypothesis S0 :",s)
print (" \n The most general hypothesis G0 :",g)

s=a[0][:-1]
temp=[]
print("\n Candidate Elimination algorithm\n")

for i in range(len(a)):
    if a[i][n]=="yes": #Use Positive for manufacture.csv
        for j in range(n):
            if a[i][j]!=s[j]:
                s[j]='?'
        for j in range(n):
            for k in range(len(temp)):  #Use len(temp)-1 for manufacture.csv
                if temp[k][j]!='?' and temp[k][j]!=s[j]:
                    del temp[k]

    if a[i][n]=="no": #Use Negative for manufacture.csv
        for j in range(n):
             if s[j]!=a[i][j] and s[j]!='?':
                 g[j]=s[j]
                 if g not in temp:
                  temp.append(g)
                 g= ['?']*n

    print("\n For Training Example No :{0} the hypothesis is S{0}
".format(i+1),s)
    if (len(temp)==0):
            print(" For Training Example No :{0} the hypothesis is G{0}
".format(i+1),g)
    else:
            print(" For Training Example No :{0} the hypothesis is
G{0}".format(i+1),temp)
```

Output:

```
sky airtemp humidity    wind water forcast enjoysport
0  sunny    warm   normal  strong  warm    same        yes
1  sunny    warm     high  strong  warm    same        yes
2  rainy    cold     high  strong  warm  change         no
3  sunny    warm     high  strong  cool  change        yes

 The initial value of hypothesis:

 The most specific hypothesis S0 : ['0', '0', '0', '0', '0', '0']

 The most general hypothesis G0 : ['?', '?', '?', '?', '?', '?']

 Candidate Elimination algorithm


 For Training Example No :1 the hypothesis is S1  ['sunny', 'warm',
'normal', 'strong', 'warm', 'same']
 For Training Example No :1 the hypothesis is G1  ['?', '?', '?', '?', '?',
'?']

 For Training Example No :2 the hypothesis is S2  ['sunny', 'warm', '?',
'strong', 'warm', 'same']
 For Training Example No :2 the hypothesis is G2  ['?', '?', '?', '?', '?',
'?']

 For Training Example No :3 the hypothesis is S3  ['sunny', 'warm', '?',
'strong', 'warm', 'same']
 For Training Example No :3 the hypothesis is G3 [['sunny', '?', '?', '?',
'?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'same']]

 For Training Example No :4 the hypothesis is S4  ['sunny', 'warm', '?',
'strong', '?', '?']
 For Training Example No :4 the hypothesis is G4 [['sunny', '?', '?', '?',
'?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

PROGRAM NO.3

Write a program to demonstrate the working of the decision tree-based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Aim:

To demonstrate the working of the decision tree-based ID3 algorithm. This algorithm is used for building a decision tree from a given dataset, where each node in the tree represents an attribute, and each branch represents a possible value of that attribute. The decision tree is built by recursively selecting the best attribute to split the data based on the information gain criterion.

Objective:

This experiment provides a comprehensive guide for understanding and implementing decision tree learning with the ID3 algorithm, covering data loading, preprocessing, tree construction, visualization, and classification tasks.

Pseudocode for decision tree-based ID3 algorithm:

1. Read the training dataset from a CSV file.
2. Calculate the entropy of the target attribute.
3. For each attribute in the dataset:
   a. Calculate the information gain.
4. Select the attribute with the highest information gain as the root of the decision tree.
5. Partition the dataset based on the values of the selected attribute.
6. Recursively build the decision tree for each partitioned subset.
7. Repeat steps 2-6 until one of the following conditions is met:
   -All instances in a subset belong to the same class.
   - There are no more attributes to split on.
8. Return the decision tree.
9. Classify new instances by traversing the decision tree based on their attribute values.

Entropy:

Entropy measures the impurity of a collection of examples.

$$Entropy\ (S) \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus}$$

Where, $p+$ is the proportion of positive examples in S
$p-$ is the proportion of negative examples in S.

Information gain:

- *Information gain,* is the expected reduction in entropy caused by partitioning the examples according to this attribute.
- The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Training Dataset Example:

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

Testing Dataset Example:

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*
T1	Rain	Cool	Normal	Strong
T2	Sunny	Mild	Normal	Strong

Python code:

```
import pandas as pd
from math import log
from pprint import pprint

def entropy(pos, neg):
    if pos == 0 or neg == 0:
        return 0
    tot = pos + neg
    return -pos / tot * log(pos / tot, 2) - neg / tot * log(neg / tot, 2)

def gain(data, attr, pos, neg):
    d, E, acu = {}, entropy(pos, neg), 0
    for i in data:
        if i[attr] not in d:
            d[i[attr]] = {}
        d[i[attr]][i[-1]] = 1 + d[i[attr]].get(i[-1], 0)
    for i in d:
        tot = d[i].get('Yes', 0) + d[i].get('No', 0)

        acu += tot / (pos + neg) * entropy(d[i].get('Yes', 0),
d[i].get('No', 0))
    return E - acu
```

```
def build(data, attr_names):
    pos, sz = len([x for x in data if x[-1] == 'Yes']), len(data[0]) - 1
    neg = len(data) - pos
    if neg == 0 or pos == 0:
        return 'Yes' if neg == 0 else 'No'

    root = max([[gain(data, i, pos, neg), i] for i in range(sz)])[1]
    fin, res = {}, {}
    uniq_attr = set([x[root] for x in data])
    for i in uniq_attr:
        res[i] = build([x[:root] + x[root + 1:] for x in data if x[root] ==
i], attr_names[:root] + attr_names[root+1:])
    fin[attr_names[root]] = res
    return fin

def classify(instance, tree):
    if isinstance(tree, dict):
        root = next(iter(tree))
        attr_index = attr_names.index(root)
        child_tree = tree[root].get(instance[attr_index], None)
        if child_tree:
            return classify(instance, child_tree)


    else:
        return tree

# Load data
df = pd.read_csv('/content/ML LAB/tennis.csv')
data = df.values.tolist()
attr_names = df.columns.values.tolist()

# Build the decision tree
tree = build(data, attr_names)
pprint(tree)

# Classify a new sample
new_sample = ['Sunny', 'Cool', 'Normal', 'Strong']
print("Classified as:", classify(new_sample, tree))
```

Output:

```
{'Outlook': {'Overcast': 'Yes',
             'Rainy': {'Windy': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
Classified as: Yes
```

Method 2:
Python code:

```
import pandas as pd
import math

def load_csv(file_path):
    df = pd.read_csv(file_path)
    return df

class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

def subtables(data, col, delete):
    dic = {}
    coldata = [row[col] for row in data]
    attr = list(set(coldata))
    counts = [0] * len(attr)
    r = len(data)
    c = len(data[0])

    for x in range(len(attr)):
        for y in range(r):
            if data[y][col] == attr[x]:
                counts[x] += 1

    for x in range(len(attr)):
        dic[attr[x]] = [[0 for i in range(c)] for j in range(counts[x])]

    pos = 0
    for y in range(r):
        if data[y][col] == attr[x]:
            if delete:
                del data[y][col]
            dic[attr[x]][pos] = data[y]
            pos += 1
    return attr, dic

def entropy(S):
    attr = list(set(S))
    if len(attr) == 1:
        return 0

    counts = [0, 0]
    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):
    attr, dic = subtables(data, col, delete=False)
    total_size = len(data)
    entropies = [0] * len(attr)
    ratio = [0] * len(attr)

    total_entropy = entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x] = len(dic[attr[x]]) / (total_size * 1.0)
        entropies[x] = entropy([row[-1] for row in dic[attr[x]]])
        total_entropy -= ratio[x] * entropies[x]
    return total_entropy

def build_tree(data, features):
    lastcol = [row[-1] for row in data]
    if len(set(lastcol)) == 1:
        node = Node("")
        node.answer = lastcol[0]
        return node

    n = len(data[0]) - 1
    gains = [0] * n
    for col in range(n):
        gains[col] = compute_gain(data, col)
    split = gains.index(max(gains))
    node = Node(features[split])
    fea = features[:split] + features[split + 1:]
    attr, dic = subtables(data, split, delete=True)

    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)
        return

    print(" " * level, node.attribute)
    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)

# Load training dataset
dataset = load_csv("/content/ml lab sample/id3.csv")
features = dataset.columns[:-1].tolist()
data = dataset.values.tolist()

# Build the decision tree
node1 = build_tree(data, features)

print("The decision tree for the dataset using ID3 algorithm is:")
print_tree(node1, 0)

# Load test dataset
testdata = load_csv("/content/ml lab sample/id3_test_1.csv")
testdata_values = testdata.values.tolist()

# Classify test instances
for xtest in testdata_values:
    print("The test instance:", xtest)
    print("The label for test instance:", end=" ")
    classify(node1, xtest, features)
```

Output:

```
The decision tree for the dataset using ID3 algorithm is:
 Outlook
       rain
             0
       sunny
             0
       overcast
             yes
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance: 0
The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance: 0
```

PROGRAM NO. 4

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

Aim:

To build an Artificial Neural Network (ANN) using the Backpropagation algorithm and test its performance using appropriate datasets.

Objective:

The objective of this experiment is to implement an Artificial Neural Network (ANN) using the Backpropagation algorithm. Through this exercise, participants will gain practical experience in preprocessing data, initializing network parameters, conducting forward and backward propagation, updating weights and biases, and evaluating the network's performance.

Pseudo code for the Backpropagation algorithm:

1. Data Preprocessing: Normalize the input features and target output to ensure they are within a similar scale.

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

2. Initialize the weights (whwh, woutwout) and biases (bhbh, boutbout) with small random values.

$$\text{derivatives\_sigmoid}(x) = x \times (1 - x)$$

3. Forward Propagation:
   - Calculate the weighted sum of inputs to the hidden and apply the activation function to get the output of the hidden layer.
   - Calculate the weighted sum of inputs to the output layer and apply the activation function to get the final output.

Weighted sum of inputs to the hidden layer:

$$\text{hinp1} = X \cdot \text{wh}$$

Total input to the hidden layer:

$$\text{hinp} = \text{hinp1} + \text{bh}$$

Activation of hidden layer:

$$\text{hlayer\_act} = \text{sigmoid}(\text{hinp})$$

Input to the output layer:

$$\text{outinp1} = \text{hlayer\_act} \cdot \text{wout}$$

Total input to the output layer:

$$\text{outinp} = \text{outinp1} + \text{bout}$$

Activation of output layer:

$$\text{output} = \text{sigmoid}(\text{outinp})$$

4. Compute the error between the predicted output.

$$\text{EO} = y - \text{output}$$

5. Backpropagation:

Gradient of the output layer:

$$\text{outgrad} = \text{derivatives\_sigmoid}(\text{output})$$

Delta of the output layer:

$$\text{d\_output} = \text{EO} \times \text{outgrad}$$

Error contribution from the hidden layer:

$$\text{EH} = \text{d\_output} \cdot \text{wout}^T$$

Gradient of the hidden layer:

$$\text{hiddengrad} = \text{derivatives\_sigmoid}(\text{hlayer\_act})$$

Delta of the hidden layer:

$$\text{d\_hiddenlayer} = \text{EH} \times \text{hiddengrad}$$

- Compute the gradient of the output layer using the derivative of the sigmoid function.
- Propagate the error back to the hidden layer.
- Compute the gradient of the hidden using the derivative of the sigmoid function.

6. Weight Update:
   - Update the using the gradient descent algorithm.
   - Update the weights using the gradient descent algorithm.

Update weights of output layer:

$$\text{wout} += \text{hlayer\_act}^T \cdot \text{d\_output} \times \text{lr}$$

Update weights of hidden layer:

$$\text{wh} += X^T \cdot \text{d\_hiddenlayer} \times \text{lr}$$

7. Repeat steps 3 to 6 for a specified number of iterations (epochs) or until a termination condition is met
8. Output: After training, the network can be used to make predictions on new data.

Training example:

Example	Sleep	Study	Expected % in Exams
1	2	9	92
2	1	5	86
3	3	6	89

Normalize the input:

Example	Sleep	Study	Expected % in Exams
1	2/3 = 0.66666667	9/9 = 1	0.92
2	1/3 = 0.33333333	5/9 = 0.55555556	0.86
3	3/3 = 1	6/9 = 0.66666667	0.89

Python Code:

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)

X = X / np.amax(X, axis=0) # maximum of X array longitudinally
y = y / 100

# Sigmoid Function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

# Variable initialization
epoch = 5000  # Setting training iterations
lr = 0.1      # Setting learning rate
inputlayer_neurons = 2  # number of features in data set
hiddenlayer_neurons = 3  # number of hidden layers neurons
output_neurons = 1  # number of neurons at output layer

# Weight and bias initialization
wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))
bh = np.random.uniform(size=(1, hiddenlayer_neurons))
wout = np.random.uniform(size=(hiddenlayer_neurons, output_neurons))
bout = np.random.uniform(size=(1, output_neurons))

for i in range(epoch):
    # Forward Propagation
    hinp1 = np.dot(X, wh)
    hinp = hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1 = np.dot(hlayer_act, wout)
    outinp = outinp1 + bout
    output = sigmoid(outinp)

    # Backpropagation
    EO = y - output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)

    # How much hidden layer weights contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad
```

```
    # Update weights
    wout += hlayer_act.T.dot(d_output) * lr
    wh += X.T.dot(d_hiddenlayer) * lr

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

Output:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89401359]
 [0.88228311]
 [0.89391465]]
```

PROGRAM NO.5

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

Aim:

To implement a naive Bayesian classifier using a sample training dataset stored as a CSV file and compute the accuracy of the classifier using test datasets.

Objective:

The objective of this program is to implement a naive Bayesian classifier for a sample training dataset stored as a CSV file. To achieve this, the first step involves loading the training dataset from the CSV file. Following that, the data is preprocessed by encoding categorical variables into numerical values using LabelEncoder. Subsequently, the dataset is split into train and test sets to facilitate model evaluation. The Gaussian Naive Bayes classifier is then trained using the training data. Next, the classifier is used to predict the classes for the test data. Finally, the accuracy of the classifier is calculated by comparing the predicted classes with the actual classes. This evaluation metric provides insight into the performance of the classifier.

Pseudocode for the Naive Bayes Algorithm:

1. Load the dataset from a CSV file and convert the loaded data to numerical format.
2. Split the dataset into training and testing sets based on a specified split ratio.
3. Separate by Class as group the instances in the dataset by their class labels.
4. Calculate the mean and standard deviation for each attribute in the dataset grouped by class.
5. Summarize the dataset statistics (mean and standard deviation) for each class.
6. Calculate the class conditional probability using the Gaussian probability density function:

$$P(x|C_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

where x is the input value, μ is the mean, and σ is the standard deviation

7. For each class, calculate the product of the class conditional probabilities for all attributes.
8. Predict the class label for a given instance by selecting the class with the highest probability.
9. Generate predictions for all instances in the test set using the previously trained model.
10. Compare the predicted class labels with the actual class labels in the test set and calculate the classification accuracy.

Sample of the dataset:

Examples	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetic Pedigree Function	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0	0.232	54	1

Python Code:

```
import csv
import random
import math

def loadcsv(filename):
    with open(filename, "r") as file:
        lines = csv.reader(file)
        dataset = list(lines)
    headers = dataset[0]  # Extract column headers
    dataset = dataset[1:]  # Exclude the first row (column headers)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return headers, dataset

def splitdataset(dataset, splitratio):
    trainsize = int(len(dataset) * splitratio)
    trainset = []
    copy = list(dataset)
    while len(trainset) < trainsize:
        index = random.randrange(len(copy))
        trainset.append(copy.pop(index))
    return [trainset, copy]

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

def mean(numbers):
    return sum(numbers)/float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)

def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in
zip(*dataset)]
    del summaries[-1]
    return summaries
```

```
def summarizebyclass(dataset):
    separated = separatebyclass(dataset)
    summaries = {}
    for classvalue, instances in separated.items():


        summaries[classvalue] = summarize(instances)
    return summaries

def calculateprobability(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, inputvector):
    probabilities = {}
    for classvalue, classsummaries in summaries.items():
        probabilities[classvalue] = 1
        for i in range(len(classsummaries)):
            mean, stdev = classsummaries[i]
            x = inputvector[i]
            probabilities[classvalue] *= calculateprobability(x, mean,
stdev)
    return probabilities

def predict(summaries, inputvector):
    probabilities = calculateclassprobabilities(summaries, inputvector)
    bestlabel, bestprob = None, -1
    for classvalue, probability in probabilities.items():
        if bestlabel is None or probability > bestprob:
            bestprob = probability
            bestlabel = classvalue
    return bestlabel

def getpredictions(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
```

```
def main():
    filename = '/content/ml lab sample/pima_indian.csv'
    splitratio = 0.67
    headers, dataset = loadcsv(filename)
    trainingset, testset = splitdataset(dataset, splitratio)
    print('Split {0} rows into train={1} and test={2}
rows'.format(len(dataset), len(trainingset), len(testset)))
    summaries = summarizebyclass(trainingset)
    predictions = getpredictions(summaries, testset)
    accuracy = getaccuracy(testset, predictions)
    print('Accuracy of the classifier is: {0}%'.format(accuracy))

main()
```

Output:

```
Split 768 rows into train=514 and test=254 rows
Accuracy of the classifier is: 75.19685039370079%
```

PROGRAM .6:

Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using a standard Heart Disease Data Set. You can use Python ML library classes/API.

Aim:

To implement a Bayesian Network model using the pgmpy library to predict heart disease based on patient data.

Objective :

This project aims to analyze a heart disease dataset by building a Bayesian Network. We structure the network based on known connections between patient attributes and heart disease. Then, we train the model using Maximum Likelihood Estimation. By employing Variable Elimination, we make predictions about heart disease from the data provided. Finally, we evaluate how accurately the model predicts heart disease.

Pseudocode to construct a Bayesian network:

1. The heart disease dataset is loaded from a CSV file named 'datasetheart.csv', with appropriate column names assigned.
2. Based on the dataset's attributes and domain knowledge, a Bayesian Network structure is defined using the `BayesianModel` class from `pgmpy.models`.
3. The model is trained using the `fit` method, with Maximum Likelihood Estimation as the estimator, using the `fit` method.
4Variable Elimination is employed for inference. The `query` method is used to predict the 'RESULT' variable given the evidence 'C=2'.
5. The inference results are printed, providing the probabilities of 'RESULT' given the evidence 'C=2'.

Sample from the dataset:

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
60	1	4	130	206	0	2	132	1	2.4	2	2	7	4

Python Code:

```
!pip install pgmpy
data = pd.read_csv('/content/ML LAB/datasetheart.csv', names=['A', 'B', 'C',
'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'RESULT'])
from pgmpy.models import BayesianModel
from pgmpy.estimators import BDeuScore, BayesianEstimator
from pgmpy.inference import VariableElimination
print(data.head(5))
print(data.tail(5))
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator



model = BayesianModel([("A","B"),("B","C"),("C","D"),("D","RESULT")])
model.fit(data,estimator=MaximumLikelihoodEstimator)

from pgmpy.inference import VariableElimination
infer = VariableElimination(model)
q = infer.query(variables=['RESULT'],evidence={"C":2})
print(q)
```

Output:

```
A  B  C    D    E  F  G    H  I    J  K  L  M  RESULT
0  63  1   3  145  233  1  0  150  0  2.3  0  0  1       1
1  37  1   2  130  250  0  1  187  0  3.5  0  0  2       1
2  41  0   1  130  204  0  0  172  0  1.4  2  0  2       1
3  56  1   1  120  236  0  1  178  0  0.8  2  0  2       1
4  57  0   0  120  354  0  1  163  1  0.6  2  0  2       1
      A  B  C    D    E  F  G    H  I    J  K  L  M  RESULT
298  57  0  0  140  241  0  1  123  1  0.2  1  0  3       0
299  45  1  3  110  264  0  1  132  0  1.2  1  0  3       0
300  68  1  0  144  193  1  1  141  0  3.4  1  2  3       0
301  57  1  0  130  131  0  1  115  1  1.2  1  1  3       0
302  57  0  1  130  236  0  0  174  0  0.0  1  1  2       0
+-----------+---------------+
| RESULT    |   phi(RESULT) |
+===========+===============+
| RESULT(0) |        0.3893 |
+-----------+---------------+
| RESULT(1) |        0.6107 |
+-----------+---------------+
```

PROGRAM .7:

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

Aim:

To compare the clustering results obtained from the K-Means algorithm and the Gaussian Mixture Model (GMM) using the Expectation-Maximization (EM) algorithm on a given dataset.

Objective:

The objective is to analyze and compare the performance of K-Means and Gaussian Mixture Model (GMM) clustering algorithms on a dataset. Firstly, the dataset stored in a .CSV file is loaded and preprocessed if required. Then, both algorithms are implemented to cluster the data into three groups. Visualizations of the clustering results are created using scatter plots. Silhouette scores are calculated for each clustering method to assess the quality of clustering. Finally, by comparing the silhouette scores of K-Means and GMM, the algorithm that demonstrates better performance for the given dataset is determined.

Pseudo code for EM algorithm:

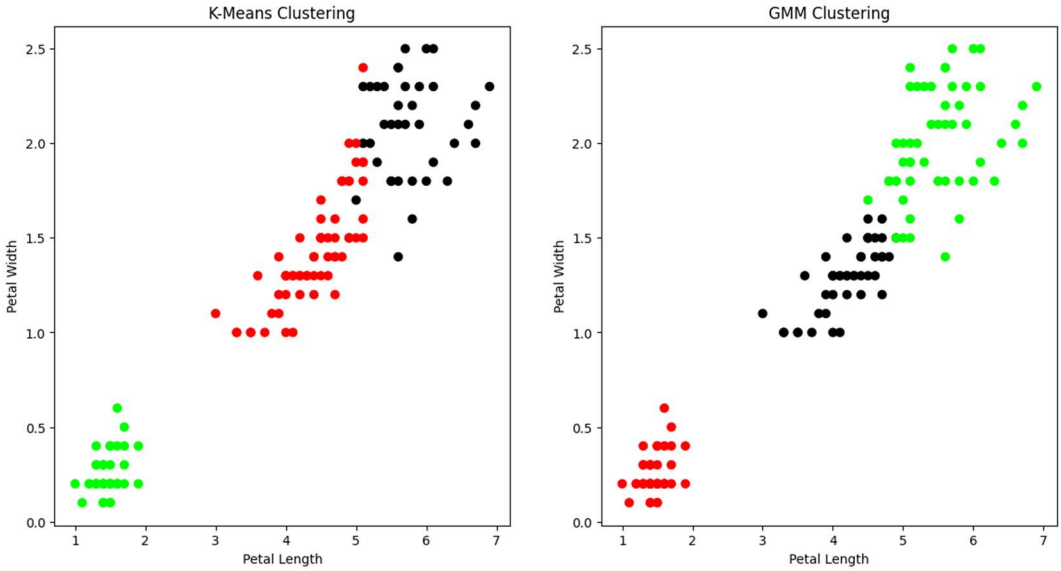
1. Load the dataset from the .CSV file and preprocess it if required.
2. Implement the K-Means algorithm:
    a. Initialize the K-Means model with the desired number of clusters (k=3).
    b. Fit the model to the dataset.
    c. Obtain the cluster labels assigned by K-Means.
3. Visualize the K-Means clustering results using scatter plots.
4. Implement the Gaussian Mixture Model (GMM) using the EM algorithm:
    a. Initialize the GMM model with the desired number of components (n=3).
    b. Fit the model to the dataset.
    c. Predict the cluster labels using the trained GMM model.
5. Visualize the GMM clustering results using scatter plots.
6. Calculate the silhouette scores for both clustering methods.
7. Print the silhouette scores of K-Means and GMM.
8. Compare the silhouette scores to determine the quality of clustering.

Python code:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn import metrics
# Load the Iris dataset
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
kmeans_model = KMeans(n_clusters=3)
kmeans_model.fit(X)
kmeans_labels = kmeans_model.labels_

# Visualize K-Means Clustering
plt.figure(figsize=(14, 7))
colormap = np.array(['red', 'lime', 'black'])
plt.subplot(1, 2, 1)
plt.scatter(X['Petal_Length'], X['Petal_Width'], c=colormap[kmeans_labels],
s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# Build Gaussian Mixture Model (GMM) using EM Algorithm
gmm = GaussianMixture(n_components=3, random_state=0).fit(X)
gmm_labels = gmm.predict(X)

# Visualize GMM Clustering
plt.subplot(1, 2, 2)
plt.scatter(X['Petal_Length'], X['Petal_Width'], c=colormap[gmm_labels],
s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# Calculate silhouette scores for both clustering methods
silhouette_kmeans = metrics.silhouette_score(X, kmeans_labels)
silhouette_gmm = metrics.silhouette_score(X, gmm_labels)

print("Silhouette Score (K-Means):", silhouette_kmeans)
print("Silhouette Score (GMM):", silhouette_gmm)

plt.show()
```

Output:

```
Silhouette Score (K-Means): 0.5528190123564095
Silhouette Score (GMM): 0.5011761635067206
```



PROGRAM 8:

Write a program to implement the 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.

Aim:

To implement the K-Nearest Neighbors (KNN) algorithm to classify instances in the Iris dataset and evaluate its performance using accuracy metrics.

Objective:

The objective of this program is to use the K-Nearest Neighbors (KNN) algorithm to classify Iris flowers into their respective species based on features like sepal length, sepal width, petal length, and petal width.

Pseudo code for the k-Nearest Neighbour algorithm:

1. Load the dataset containing features and target classes.
2. Split the dataset into training and testing sets.
3. Train the k-NN classifier using the training data with a specified value of k.
4. Make predictions on the test data using the trained classifier.
5. Evaluate the performance of the classifier using a confusion matrix and accuracy metrics.

Sample from the dataset

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Python code:

```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets
#Iris Plants Dataset, dataset contains 150 (50 in each of three
classes)Number of Attributes: 4 numeric, predictive attributes and the Class

# Load the Iris dataset
iris = datasets.load_iris()

#The x variable contains the first four columns of the dataset (i.e.
attributes) while y contains the labels.
x = iris.data
y = iris.target
```

```python
# Print the column names and data
print('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1-Iris-Versicolour, 2-Iris-Virginica')
print(y)

# Splits the dataset into 70% train data and 30% test data. This means that
out of total 150 records, the training set will contain
105 records and the test set contains 45 of those records
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

# Train the K-Nearest Neighbors model with k=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

# Make predictions on the test data
y_pred = classifier.predict(x_test)

# Print the confusion matrix
print('Confusion Matrix')
print(confusion_matrix(y_test, y_pred))

# Print accuracy metrics
print('Accuracy Metrics')
print(classification_report(y_test, y_pred))
```

Output:

```
class: 0-Iris-Setosa, 1-Iris-Versicolour, 2-Iris-Virginica
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Confusion Matrix
[[18  0  0]
 [ 0 10  2]
 [ 0  0 15]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      0.83      0.91        12
           2       0.88      1.00      0.94        15

    accuracy                           0.96        45
   macro avg       0.96      0.94      0.95        45
weighted avg       0.96      0.96      0.95        45
```

PROGRAM 9:

Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.

Aim:

This experiment aims to implement the non-parametric Locally Weighted Regression (LWR) algorithm to fit data points and visualize the regression curve.

Objective:

- Generate synthetic data points using a predefined function and add noise to mimic real-world scenarios.
- Implement the Locally Weighted Regression algorithm to fit the data points.
- Visualize the regression curve along with the original data points for different values of bandwidth (tau) to observe its effect on the regression curve.

Pseudocode for the non-parametric Locally Weighted Regression algorithm:

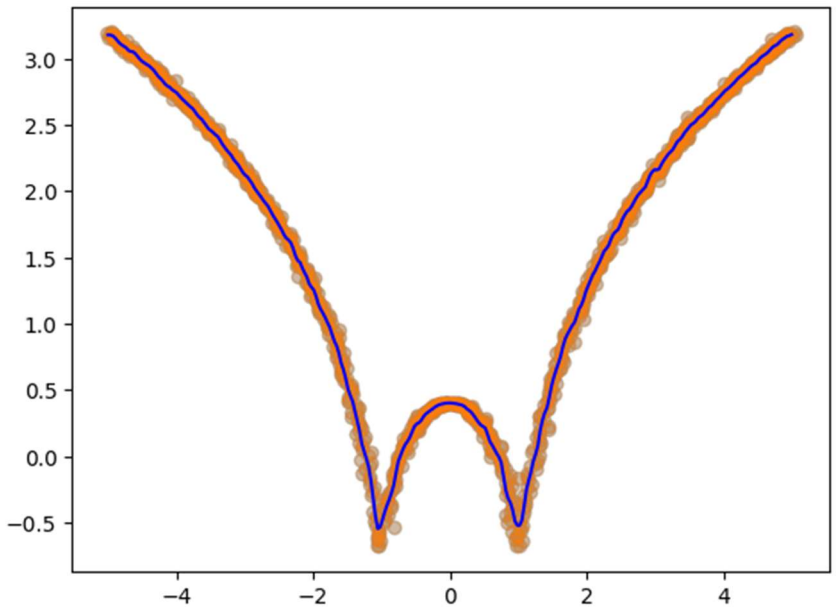
1. Define the radial kernel function to compute the weights for each data point based on its distance from the query point.
2. Implement the local_regression function to perform the locally weighted regression:
   a. For each query point x0:
      i. Calculate the weights for each training data point using the radial kernel function.
      ii. Formulate the weighted design matrix by multiplying each feature vector by its corresponding weight.
      iii. Compute the regression coefficients using the weighted least squares method.
      iv. Predict the output for the query point using the computed coefficients.
3. Define a plotting function plot_lr(tau) to visualize the regression curve for a given bandwidth (tau):
   a. Generate a set of query points in the desired domain.
   b. For each query point, predict the output using the local_regression function.
   c. Plot the original data points along with the regression curve.
4. Call the plot_lr function with different values of bandwidth (tau) to observe the effect on the regression curve**.**

Python code:

```
import numpy as np
import matplotlib.pyplot as plt
# Generating Data
x = np.linspace(-5, 5, 1000)
y = np.log(np.abs((x ** 2) - 1) + 0.5)
x = x + np.random.normal(scale=0.05, size=1000)
plt.scatter(x, y, alpha=0.3)

# Locally Weighted Regression Functions
def local_regression(x0, x, y, tau):
    x0 = np.r_[1, x0]
    x = np.c_[np.ones(len(x)), x]
    xw = x.T * radial_kernel(x0, x, tau)
    beta = np.linalg.pinv(xw @ x) @ xw @ y
    return x0 @ beta
```

```
def radial_kernel(x0, x, tau):
    return np.exp(np.sum((x - x0) ** 2, axis=1) / (-2 * tau ** 2))

# Plotting Function
def plot_lr(tau):
    domain = np.linspace(-5, 5, num=300)
    pred = [local_regression(x0, x, y, tau) for x0 in domain]
    plt.scatter(x, y, alpha=0.3)
    plt.plot(domain, pred, color="blue")
    plt.show()  # Corrected show() function
    return plt

# Plotting with specified bandwidth (tau)
plot_lr(0.03)
```

Output:



PROGRAM 10:

Demonstrate the working of SVM classifier for a suitable dataset.

Aim:

To demonstrate the application of the Support Vector Machine (SVM) classifier on suitable datasets to perform classification tasks

Objective :

The objective of the program is to illustrate the practical implementation of the Support Vector Machine (SVM) classifier for performing classification tasks on a suitable dataset.

Pseudo code for the SVM classifier:

1. Import necessary libraries (NumPy, Matplotlib, scikit-learn).
2. Load the dataset using a suitable function (e.g., `load_iris()` from scikit-learn).
3. Separate the features (X) and target labels (y) from the dataset.
4. Create an instance of the SVM classifier with desired parameters (e.g., kernel type).
5. The decision boundary is determined by the equation of a hyperplane. In the case of a linear kernel (kernel='linear'), the decision boundary equation is represented as:

$$w^T \cdot x + b = 0$$

w represents the weights (coefficients) of the features, x is the input feature vector, and b is the bias term.

$$K(x_i, x_j) = x_i^T \cdot x_j$$

This kernel computes the dot product between the feature vectors x_i and x_j in the original feature space.

6. Fit the SVM classifier to the training data using the `fit()` method.

$$f(x) = w^T \cdot x + b$$

If $f(x)$ is positive, the sample is assigned to one class, and if it is negative, it is assigned to another class.

7. Use the trained classifier to predict labels for the test data using the `predict()` method.
8. Evaluate Performance:
    a. Calculate the accuracy of the classifier using `accuracy_score()` from scikit-learn.
    b. Generate a classification report using `classification_report()` to get precision, recall, F1-score, etc.
9. Visualize Decision Boundary:
    a. Define a function to plot the decision boundary of the classifier.
    b. Use meshgrid to create a grid of points and predict the class for each point.
    c. Plot the decision boundary along with the data points using Matplotlib.
10. Show the decision boundary plots for visualization.

Python code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Initialize the SVM classifier
svm_classifier = SVC(kernel='linear')

# Train the SVM classifier
svm_classifier.fit(X_train, y_train)

# Predict the labels for test data
y_pred = svm_classifier.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate classification report
report = classification_report(y_test, y_pred,
target_names=iris.target_names)
print("Classification Report:")
print(report)
# Function to plot decision boundary
def plot_decision_boundary(clf, X, y, title):
    plt.figure(figsize=(8, 6))
    h = 0.02  # step size in the mesh
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max,
h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.spring, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.spring, edgecolors='k')
    plt.xlabel('Feature 1')
```

```
    plt.ylabel('Feature 2')
    plt.title(title)
    plt.show()

# Generating and plotting the make_blobs dataset
X_blob, Y_blob = make_blobs(n_samples=500, centers=2, random_state=0,
cluster_std=0.40)
plt.scatter(X_blob[:, 0], X_blob[:, 1], c=Y_blob, s=50, cmap='spring')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Make Blobs Dataset')
plt.show()

# Training an SVM classifier on the make_blobs dataset
clf_blob = SVC(kernel='linear')
clf_blob.fit(X_blob, Y_blob)

# Plotting the decision boundary for the make_blobs dataset
plot_decision_boundary(clf_blob, X_blob, Y_blob, 'SVM Decision Boundary for
Make Blobs Dataset')

# Loading the Iris dataset
iris = load_iris()
X_iris = iris.data[:, :2]  # considering only the first two features for
visualization
y_iris = iris.target

# Splitting the Iris dataset into training and testing sets
X_train_iris, X_test_iris, y_train_iris, y_test_iris =
train_test_split(X_iris, y_iris, test_size=0.3, random_state=42)

# Training an SVM classifier on the Iris dataset
clf_iris = SVC(kernel='linear')
clf_iris.fit(X_train_iris, y_train_iris)

# Plotting the decision boundary for the Iris dataset
plot_decision_boundary(clf_iris, X_iris, y_iris, 'SVM Decision Boundary for
Iris Dataset')
```

Output:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```