A T M E

College of Engineering

13thKM Stone, Bannur Road, Mysore - 560 028

## Department of CSE–Artificial Intelligence & Machine Learning

## (Academic Year 2024-25)

# LABORATORY MANUAL

**SUBJECT: MACHINE LEARNING 2 LAB**

**SUB CODE: BAI702**

**SEMESTER:VII**

**SCHEME: 2022**

| Prepared By | Verified By | Approved by |
|---|---|---|
| Mr. Manjunath H<br>Instructor | Mrs.KHATEEJA AMBAREEN<br>Assistant Professor<br>Dept. of CSE-AI &ML | Dr. ANIL KUMAR C.J<br>Assoc. Professor & Head<br>Dept. of CS-AI &ML |

## Institute Vision

- Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

## Institute Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

## Department Vision

To impart technical education in the field of Artificial intelligence and machine learning of topnotch quality with a high level of professional competence, social obligation, and global cognizance among the students.

## Department Mission

- To impart technical education that is up to date, relevant and makes students to compete at global level
- Fostering an ambiance where students can adopt the suitable moral, intellectual, emotional, and physical attributes to shine as the leaders of tomorrow's society.
- To strive to meet ever higher educational standard.

## Program Outcomes (PO's)

**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 modeling 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 (PEO's):

**PEO1:** Graduates will be able to hone their problem-solving abilities and capacity to offer solutions to challenges that arise in the actual world.

**PEO2:** Able to design and develop AI based solutions to real-world problems in a business, research, or social environment.

**PEO3:** Graduates shall acquire and inculcate corporate culture, core attributes, and leadership qualities as well as professional etiquette's and lifelong learning.

## Program Specific Outcomes (PSO's)

**PSO1:** Ability to design and develop artificial intelligent based solutions by applying optimal algorithms to solve real world issues.

**PSO2:** Ability to apply suitable AI tools and techniques to offer solutions in the various domains of engineering.

| Machine Learning II | | Semester | 7 |
|---|---|---|---|
| Course Code | **BAI702** | CIE Marks | 50 |
| Teaching Hours/Week (L:T:P: S) | 3:0:2:0 | SEE Marks | 50 |
| Total Hours of Pedagogy | 40 hours Theory + 8-10 Lab slots | Total Marks | 100 |
| Credits | 04 | Exam Hours | 3 |
| Examination nature (SEE) | Theory/practical | | |

**Course objectives:**

- To introduce concept learning and General to specific learning
- To learn set of rules using Sequential Covering approach
- To make decisions on decision by committee
- To understand similarities using unsupervised learning .
- To understand Markov Chain Monte Carlo (MCMC) and Graphical Methods

**Teaching-Learning Process (General Instructions)**

These are sample Strategies; that teachers can use to accelerate the attainment of the various course outcomes.

1. Lecturer method (L) need not to be only a traditional lecture method, but alternative effective teaching methods could be adopted to attain the outcomes.
2. Use of Video/Animation to explain functioning of various concepts.
3. Encourage collaborative (Group Learning) Learning in the class.
4. Ask at least three HOT (Higher order Thinking) questions in the class, which promotes critical thinking.
5. Adopt Problem Based Learning (PBL), which fosters students' Analytical skills, develop design thinking skills such as the ability to design, evaluate, generalize, and analyze information rather than simply recall it.
6. Introduce Topics in manifold representations.
7. Show the different ways to solve the same problem with different logic and encourage the students to come up with their own creative ways to solve them.
8. Discuss how every concept can be applied to the real world - and when that's possible, it helps improve the students' understanding

**MODULE-1**

**Introduction:** Well-Posed Learning Problems, Designing a Learning System, Perspectives and Issues in Machine Learning.

**Concept Learning and the General-to-Specific Ordering**: A Concept Learning Task, Concept Learning as Search, Find-S: Finding a Maximally Specific Hypothesis, Version Spaces and the Candidate-Elimination Algorithm, Remarks on Version Spaces and Candidate-Elimination, Inductive Bias.

**Text Book 1 : Ch 1 & 2**

**MODULE-2**

**Learning Sets of Rules:** Sequential Covering Algorithms, Learning Rule Sets: Example-Based Methods, Learning First-Order Rules, FOIL: A First-Order Inductive Learner.

**Analytical Learning:** Perfect Domain Theories: Explanation-Based Learning, Explanation-Based Learning of Search Control Knowledge, Inductive-Analytical Approaches to Learning.

**Text Book 1 : Ch 10 & 11**

**MODULE-3**

**Decision by Committee: Ensemble Learning:** Boosting: Adaboost , Stumping, Bagging: Subagging, Random Forests, Comparison With Boosting, Different Ways To Combine Classifiers.

**Unsupervised Learning: The K-MEANS algorithm :** Dealing with Noise ,The k-Means Neural Network , Normalisation ,A Better Weight Update Rule ,Using Competitive Learning for Clustering.

**Text Book 2: Chap 13 and 14.1**

**MODULE-4**

**Unsupervised Learning:** Vector Quantisation**,** the self-organising feature map , The SOM Algorithm, Neighbourhood Connections, Self-Organisation, Network Dimensionality and Boundary Conditions, Examples of Using the SOM.
**Markov Chain Monte Carlo (MCMC) Methods:** Sampling : Random Numbers ,Gaussian Random Numbers ,Monte Carlo Or Bust ,The Proposal Distribution , Markov Chain Monte Carlo.
**Text Book 2: Chap 14.2, 14.3, 15**

**MODULE-5**

**Graphical Models:** Bayesian Networks : Approximate Inference , Making Bayesian Networks , Markov Random Fields , Hidden Markov Models (Hmms), The Forward Algorithm , The Viterbi Algorithm , The Baum–Welch Or Forward–Backward Algorithm , Tracking Methods , The Kalman Filter, The Particle Filter.
**Text Book 2 : Chap 16**

**PRACTICAL COMPONENT OF IPCC** *(May cover all / major modules)*

| Sl.NO | Experiments |
|---|---|
| 1 | Read a dataset from the user and i. Use the Find-S algorithm to find the most specific hypothesis that is consistent with the positive examples. Ii. What is the final hypothesis after processing all the positive examples? Using the same dataset, apply the Candidate Elimination algorithm.<br>Determine the final version space after processing all examples (both positive and negative).<br>What are the most specific and most general hypotheses in the version space? |
| 2 | Read a dataset and use an **example-based method** (such as **RIPPER** or **CN2**) to generate a set of classification rules . Apply the **FOIL algorithm** (First-Order Inductive Learner) to learn first-order rules for predicting. |
| 3 | Read a supervised dataset and use bagging and boosting technique to classify the dataset. Indicate the performance of the model. |
| 4 | Read an unsupervised dataset and group the dataset based on similarity based on k-means clustering . |
| 5 | Read a dataset and perform unsupervised learning using SOM algorithm. |
| 6 | Write a function to generate **uniform random numbers** in the interval [0, 1]. Use this function to generate 10 random samples and evaluate f(x) for each sample. What are the sampled function values?<br>Using the samples generated in the previous step, estimate the integral I using the **Monte Carlo method**. |
| 7 | Read a dataset and indicate the likelihood of an event occurring using Bayesian Networks. |
| 8 | Refer to the dataset in question 7 and indicate inferences based on the sequence of steps . |

**Course outcomes (Course Skill Set):**
At the end of the course, the student will be able to:

1. Apply concept learning and General to specific learning
2. Design models to classify supervised data .
3. To analyze methods to identify similarities using unsupervised learning .
4. To understand Markov Chain Monte Carlo (MCMC) and Graphical Methods.

**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 out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

## CIE for the theory component of the IPCC (maximum marks 50)

- IPCC means practical portion integrated with the theory of the course.
- CIE marks for the theory component are **25 marks** and that for the practical component is **25 marks**.
- 25 marks for the theory component are split into **15 marks** for two Internal Assessment Tests (Two Tests, each of 15 Marks with 01-hour duration, are to be conducted) and **10 marks** for other assessment methods mentioned in 22OB4.2. The first test at the end of 40-50% coverage of the syllabus and the second test after covering 85-90% of the syllabus.
- Scaled-down marks of the sum of two tests and other assessment methods will be CIE marks for the theory component of IPCC (that is for **25 marks)**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the theory component of IPCC.

## CIE for the practical component of the IPCC

- **15 marks** for the conduction of the experiment and preparation of laboratory record, and **10 marks** for the test to be conducted after the completion of all the laboratory sessions.
- On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce and marks shall be awarded on the same day.
- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to **15 marks**.
- The laboratory test **(duration 02/03 hours)** after completion of all the experiments shall be conducted for 50 marks and scaled down to **10 marks.**
- Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for **25 marks**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

## SEE for IPCC

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the course (**duration 03 hours**)

1. The question paper will have ten questions. Each question is set for 20 marks.
2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), **should have a mix of topics** under that module.
3. The students have to answer 5 full questions, selecting one full question from each module.
4. Marks scoredby the student shall be proportionally scaled down to 50 Marks

**The theory portion of the IPCC shall be for both CIE and SEE, whereas the practical portion will have a CIE component only. Questions mentioned in the SEE paper may include questions from the practical component**.

**Suggested Learning Resources:**

**Books**

1. Tom Mitchell, ―Machine Learning, McGraw Hill, 3rd Edition, 1997.
2. Stephen Marsland, "Machine Learning - An Algorithmic Perspective", Second Edition, CRC Press - Taylor and Francis Group, 2015.

**Web links and Video Lectures (e-Resources):** https://archive.nptel.ac.in/courses/106/106/106106139
https://www.youtube.com/watch?v=i_LwzRVP7bg
https://www.youtube.com/watch?v=NWONeJKn6kc

**Activity Based Learning (Suggested Activities in Class)/ Practical Based learning**

Miniproject in the topics of machine learning.

# CONTENTS

# Introduction to Machine Learning 2

## 1)Supervised Learnig

Data Structures can be classifiedas:

- You have **labeled data** (features + target/output).
- The goal is to learn a function that maps inputs (X) to outputs (y).
- Two main tasks:
    - **Classification** (e.g., spam or not spam)
    - **Regression** (e.g., predicting house prices)

## 2)Common Algorithms

| Algorithm | Type | Description |
|---|---|---|
| **Linear Regression** | Regression | Predicts a continuous value. |
| **Logistic Regression** | Classification | Predicts class probabilities (binary/multi-class). |
| **Decision Trees** | Both | Rule-based model; easy to visualize. |
| **K-Nearest Neighbors (KNN)** | Both | Predicts using nearby data points. |
| **Support Vector Machines (SVM)** | Both | Finds optimal decision boundaries. |
| **Random Forest** | Both | Ensemble of decision trees (better accuracy). |

## 3) Example Use Case

**Predicting if a patient has diabetes**

- Features: age, BMI, glucose level, etc.
- Label: 1 (has diabetes), 0 (does not)
- Algorithm: Decision Tree or Logistic Regression
- Metric: Accuracy or F1-score

## PROGRAM 1

**1)Read a dataset from the user and i. Use the Find-S algorithm to find the most specific hypothesis that is consistent with the positive examples. Ii. What is the final hypothesis after processing all the positive examples? Using the same dataset, apply the Candidate Elimination algorithm. Determine the final version space after processing all examples (both positive and negative). What are the most specific and most general hypotheses in the version space?**

**Soln)**

```
    def get_input_data():
  print("Enter the dataset row-by-row as comma-separated values (e.g.,
Sunny,Warm,Normal,Strong,Warm,Same,Yes).")
  print("Enter an empty line to finish.\n")
  dataset = []
  while True:
    row = input("Enter example: ").strip()
    if not row:
      break
    parts = row.split(",")
    dataset.append(parts)
  return dataset


def find_s_algorithm(data):
  print("\n=== Find-S Algorithm ===")
  hypothesis = None
  for row in data:
    *attributes, label = row
    if label.lower() == "yes":
      if hypothesis is None:
        hypothesis = attributes.copy()
      else:
        for i in range(len(attributes)):
          if hypothesis[i] != attributes[i]:
            hypothesis[i] = '?'
  print("Final Hypothesis (Find-S):", hypothesis)
  return hypothesis

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

```
def generalize_S(example, S):
    for i in range(len(S)):
        if S[i] != example[i]:
            S[i] = '?'
    return S
def specialize_G(example, G, attributes):
    new_G = []
    for g in G:
        for i in range(len(g)):
            if g[i] == '?':
                for value in attributes[i]:
                    if value != example[i]:
                        new_hypothesis = g.copy()
                        new_hypothesis[i] = value
                        new_G.append(new_hypothesis)
    return new_G
def candidate_elimination_algorithm(data):
    print("\n=== Candidate Elimination Algorithm ===")
    attributes = [list(set([row[i] for row in data])) for i in range(len(data[0]) - 1)]

    S = ['0'] * (len(data[0]) - 1)
    G = [['?'] * (len(data[0]) - 1)]

    for row in data:
        *x, label = row
        if label.lower() == 'yes':
            # Remove from G any hypothesis inconsistent with x
            G = [g for g in G if all(g[i] == '?' or g[i] == x[i] for i in range(len(x)))]
            # Generalize S minimally to be consistent with x
            if S == ['0'] * len(x):
                S = x.copy()
            else:
                S = generalize_S(x, S)
            # Remove hypotheses from G that are not more general than S
            G = [g for g in G if more_general(g, S)]
        else:
            # Negative example
            if all(S[i] == x[i] or S[i] == '?' for i in range(len(x))):
                S = ['0'] * len(x)
            # Specialize G to remove inconsistent hypotheses
            G_temp = []
```

```
        for g in G:

                if all(g[i] == '?' or g[i] == x[i] for i in range(len(x))):



            G_temp += specialize_G(x, [g], attributes)
        else:
            G_temp.append(g)
    # Remove more specific hypotheses
    G = []
    for h in G_temp:
        if not any(more_general(h2, h) and h != h2 for h2 in G_temp):
            G.append(h)

print("Final Specific Hypothesis (S):", S)
print("Final General Hypotheses (G):", G)
return S, G


if __name__ == "__main__":
    data = get_input_data()
    final_hypothesis = find_s_algorithm(data)
    S_final, G_final = candidate_elimination_algorithm(data)

    print("\n=== Results ===")
    print("Find-S Final Hypothesis:", final_hypothesis)
    print("Candidate Elimination Version Space:")
    print("Most Specific Hypothesis:", S_final)
    print("Most General Hypotheses:")
    for g in G_final:
        print(g)
```

**OUTPUT**

```
Enter the dataset row-by-row as comma-separated values (e.g., Sunny,Warm,Normal,Strong,Warm,Sam e,Yes).
Enter an empty line to finish.

Enter example:  Sunny,Warm,Normal,Strong,Warm,Same,Yes Sunny,Warm,High,Strong,Warm,Same,Yes Rai
ny,Cold,High,Strong,Warm,Change,No Sunny,Warm,High,Strong,Cool,Change,Yes
Enter example:
```

=== Find-S Algorithm ===
Final Hypothesis (Find-S): ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes Sunny', '
Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change ', 'No Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change']

=== Candidate Elimination Algorithm ===
Final Specific Hypothesis (S): ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes Sunny ', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Ch ange', 'No Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change']
Final General Hypotheses (G): [['?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?'
, '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?']]

=== Results ===
Find-S Final Hypothesis: ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes Sunny', 'Wa rm', 'High', 'Strong', 'Warm', 'Same', 'Yes Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change']
Candidate Elimination Version Space:

Most Specific Hypothesis: ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes Sunny', 'W arm', 'High', 'Strong', 'Warm', 'Same', 'Yes Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change'
, 'No Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change'] Most
General Hypotheses:
['?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?']

## PROGRAM 2

**Read a dataset from the user and i. Use the Find-S algorithm to find the most specific hypothesis that is consistent with the positive examples. Ii. What is the final hypothesis after processing all the positive examples? Using the same dataset, apply the Candidate Elimination algorithm. Determine the final version space after processing all examples (both positive and negative). What are the most specific and most general hypotheses in the version space?**

Soln)

```
import pandas as pd
import Orange

# Load a dataset (e.g., Iris dataset or any CSV with categorical target)
def load_dataset(csv_path):
    df = pd.read_csv(csv_path)

    attributes = []
    for col in df.columns[:-1]:  # All columns except the target
        if df[col].dtype == 'object':
            # Use the unique values as categories
            values = list(map(str, df[col].unique()))
            attributes.append(Orange.data.DiscreteVariable(col, values))
        else:
            attributes.append(Orange.data.ContinuousVariable(col))

    # Target variable (last column)
    class_col = df.columns[-1]
    class_values = list(map(str, df[class_col].unique()))
    class_var = Orange.data.DiscreteVariable(class_col, class_values)

    domain = Orange.data.Domain(attributes, class_var)

    # Convert all values to strings (because Orange expects matching categories)
    data_as_str = df.astype(str).values.tolist()
    table = Orange.data.Table.from_list(domain, data_as_str)

    return table

# CN2 Rule Learner (example-based rule learner)
```



Pe

```
def apply_cn2_learner(table):

learner = Orange.classification.rules.CN2Learner()


    classifier = learner(table)
    return classifier

# FOIL-like learner (CN2SDUnorderedLearner uses similar principles)
def apply_foil_like_learner(table):
    learner = Orange.classification.rules.CN2SDUnorderedLearner()
    classifier = learner(table)
    return classifier

# Display the rules
def display_rules(classifier):
    print("\nLearned Rules:\n")
    for rule in classifier.rule_list:
        print(rule)

# Main function
def main():
    csv_path = "your_dataset.csv"  # Replace with your dataset path
    table = load_dataset(csv_path)

    print("=== CN2 RULES ===")
    cn2_classifier = apply_cn2_learner(table)
    display_rules(cn2_classifier)

    print("\n=== FOIL-LIKE RULES ===")
    foil_classifier = apply_foil_like_learner(table)
    display_rules(foil_classifier)

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

**OUTPUT**

```
=== CN2 RULES ===

Learned Rules:

IF Outlook==Overcast THEN
Play=Yes IF Temperature==Hot
THEN Play=No
IF Humidity!=High AND Outlook!=Rain THEN
```



Pe

```
Play=Yes IF Outlook==Sunny THEN Play=No
IF Wind==Weak THEN
Play=Yes IF Outlook==Rain

THEN Play=No IF TRUE
THEN Play=Yes

=== FOIL-LIKE RULES ===

Learned Rules:

IF Humidity==High AND Outlook!=Overcast THEN
Play=No IF Outlook==Sunny AND Humidity==High THEN
Play=No
IF Outlook!=Overcast AND Wind!=Weak THEN
Play=No IF Wind!=Weak AND Outlook==Rain THEN
Play=No
IF Outlook!=Overcast THEN Play=No
IF  Outlook!=Sunny  AND  Wind==Weak  THEN
Play=Yes IF Humidity!=High AND Outlook!=Rain
THEN Play=Yes IF Outlook==Overcast THEN
Play=Yes
IF Humidity!=High THEN Play=Yes
IF Outlook!=Sunny AND Wind==Weak AND Temperature!=Cool THEN
Play=Yes IF Humidity!=High AND Wind==Weak THEN Play=Yes
IF Outlook!=Sunny THEN Play=Yes
IF Temperature!=Hot AND Outlook!=Rain THEN
Play=Yes IF Outlook!=Sunny AND Temperature!=Cool
THEN Play=Yes IF Temperature!=Hot AND
Humidity!=High THEN Play=Yes
IF TRUE THEN Play=Yes
```

## PROGRAM 3

**Read a supervised dataset and use bagging and boosting technique to classify the dataset. Indicate the performance of the model.**

**Soln)**

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Step 1: Load dataset (Iris)
iris = load_iris()
X, y = iris.data, iris.target
```



Pe

```
# Step 2: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 3: Define base model
base_model = DecisionTreeClassifier(random_state=42)

# Step 4: Bagging Classifier
bagging = BaggingClassifier(estimator=base_model, n_estimators=50, random_state=42)
bagging.fit(X_train, y_train)
y_pred_bag = bagging.predict(X_test)

# Step 5: Boosting Classifier (AdaBoost)
boosting = AdaBoostClassifier(estimator=base_model, n_estimators=50, random_state=42)
boosting.fit(X_train, y_train)
y_pred_boost = boosting.predict(X_test)

# Step 6: Performance Evaluation
print("=== Bagging Classifier ===")
print("Accuracy:", accuracy_score(y_test, y_pred_bag))
print("Classification Report:\n", classification_report(y_test, y_pred_bag))

print("=== Boosting Classifier (AdaBoost) ===")
print("Accuracy:", accuracy_score(y_test, y_pred_boost))
print("Classification Report:\n", classification_report(y_test, y_pred_boost))
```

**OUTPUT**

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

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       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

=== Boosting Classifier (AdaBoost) Accuracy: 1.0 ===
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       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
```



Pe

## PROGRAM 4

**Read an unsupervised dataset and group the dataset based on similarity based on k-means clustering**

.

**Slno)**

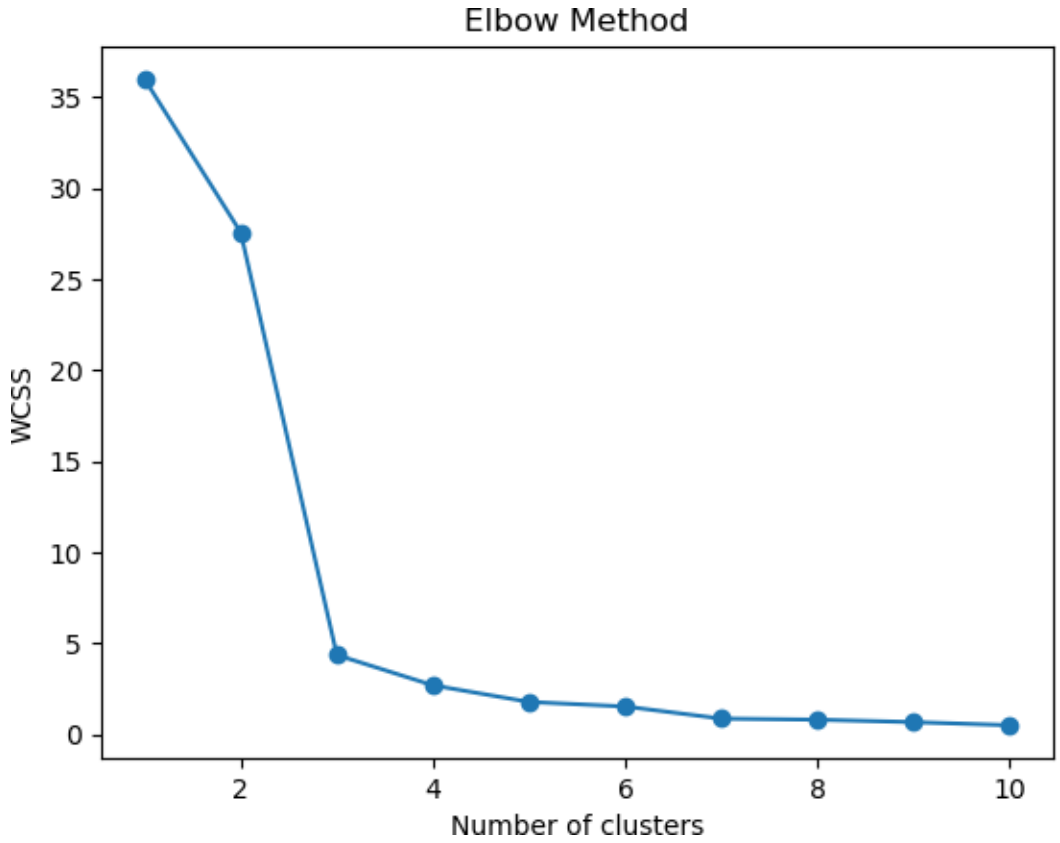
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Step 1: Load Dataset
def load_dataset(path):
    try:
        data = pd.read_csv(path)
        print("Dataset loaded successfully.")
        return data
    except Exception as e:
        print(f"Error loading dataset: {e}")
        return None

# Step 2: Preprocess Dataset (drop non-numeric columns)
def preprocess_data(data):
    numeric_data = data.select_dtypes(include=['float64', 'int64'])
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(numeric_data)
    return scaled_data

# Step 3: Determine Optimal Number of Clusters using Elbow Method
def plot_elbow_curve(data):
    wcss = []  # Within-cluster sum of squares
    for i in range(1, 11):
        kmeans = KMeans(n_clusters=i, random_state=42)
        kmeans.fit(data)
        wcss.append(kmeans.inertia_)

    plt.plot(range(1, 11), wcss, marker='o')
    plt.title('Elbow Method')
    plt.xlabel('Number of clusters')
    plt.ylabel('WCSS')
    plt.show()

# Step 4: Apply KMeans Clustering
def apply_kmeans(data, n_clusters):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    clusters = kmeans.fit_predict(data)
```

```
    return clusters

    # Main Function
    def main():
        file_path = input("Enter path to the CSV dataset: ")
        dataset = load_dataset(file_path)

        if dataset is None:
            return

        print("\nFirst 5 rows of the dataset:")
        print(dataset.head())

        scaled_data = preprocess_data(dataset)

        print("\nPlotting Elbow Curve to determine optimal number of clusters...")
        plot_elbow_curve(scaled_data)

        k = int(input("Enter the number of clusters (k): "))
        clusters = apply_kmeans(scaled_data, k)

        dataset['Cluster'] = clusters
        print("\nClustered Dataset (with 'Cluster' column added):")
        print(dataset.head())

        # Optional: Save the clustered dataset to a new CSV file
        dataset.to_csv("clustered_output.csv", index=False)
        print("\nClustered dataset saved as 'clustered_output.csv'.")

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

**OUTPUT**

```
Enter path to the CSV dataset:  sample_data.csv Dataset
loaded successfully.

First 5 rows of the dataset:
    Feature1  Feature2
0        1.0        2.0
1        1.5        1.8
2        5.0        8.0
3        8.0        8.0
4        1.0        0.6
```

Plotting Elbow Curve to determine optimal number of clusters...




Enter the number of clusters (k): 2

Clustered Dataset (with 'Cluster' column added): Feature1
Feature2 Cluster
0 1.0 2.0 1
1 1.5 1.8 1

| | | | |
|---|---|---|---|
| 2 | 5.0 | 8.0 | |
| 3 | 8.0 | 8.0 | 1 |
| 4 | 1.0 | 0.6 | 1 |

Clustered dataset saved as 'clustered_output.csv'.

## PROGRAM 5

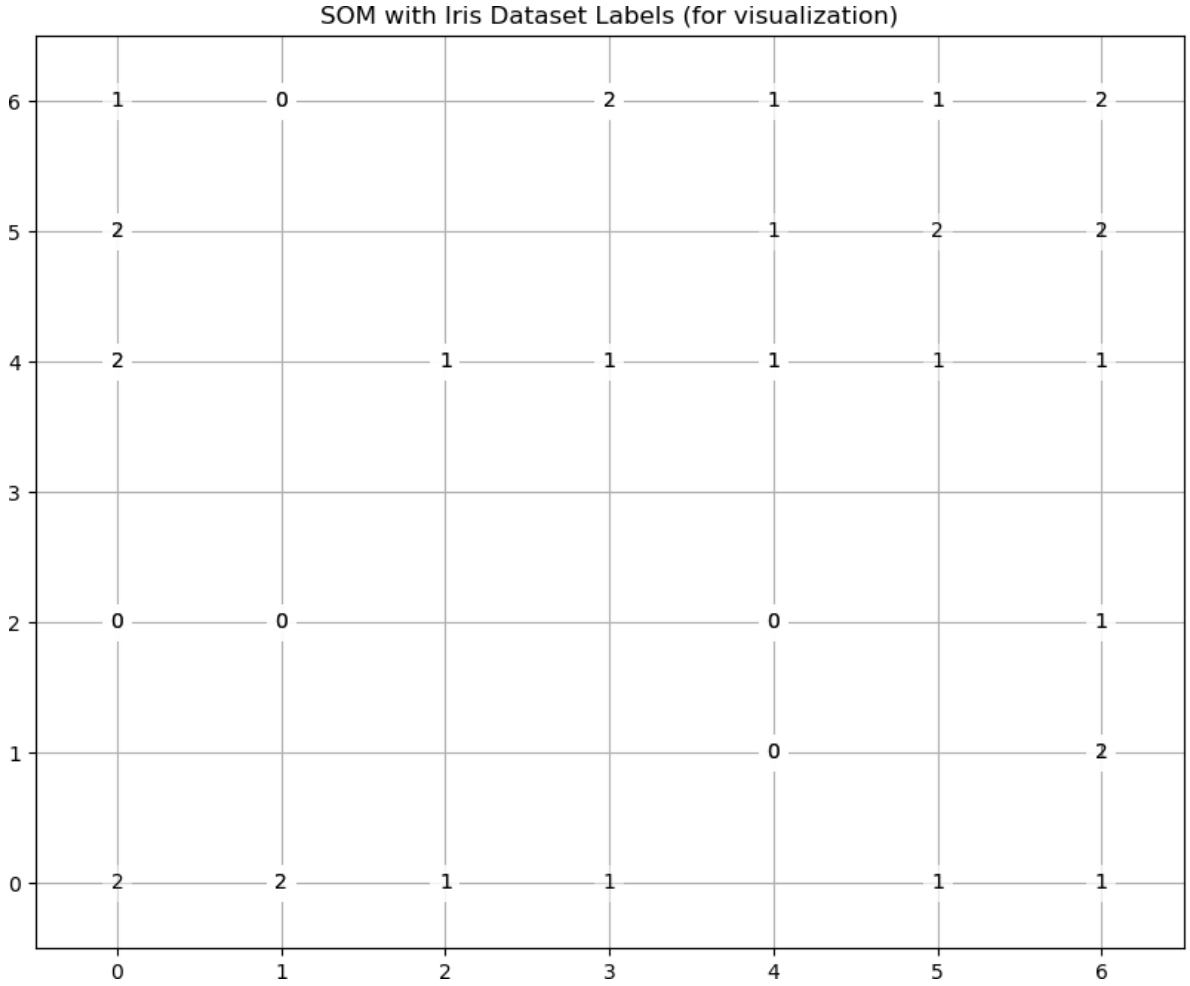
**Read a dataset and perform unsupervised learning using SOM algorithm.**

Soln)

```
# Install the minisom library
# pip install minisom

import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from minisom import MiniSom
import matplotlib.pyplot as plt

# Step 1: Load dataset
# Example: Load the Iris dataset
from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target  # Not used in unsupervised learning, just for visualization

# Step 2: Normalize the data
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Initialize and train SOM
# Parameters: 7x7 SOM, input_len = number of features
som = MiniSom(x=7, y=7, input_len=X.shape[1], sigma=1.0, learning_rate=0.5)
som.random_weights_init(X_scaled)
som.train_random(data=X_scaled, num_iteration=100)

# Step 4: Plotting the SOM
```

```
plt.figure(figsize=(10, 8))
frequencies = np.zeros((7, 7))


for x in X_scaled:
    w = som.winner(x)
    frequencies[w[0], w[1]] += 1

# Plot the frequency map (how many samples are mapped to each neuron)
plt.pcolor(frequencies.T, cmap='Blues')
plt.colorbar(label='Number of Mappings')
plt.title("Self-Organizing Map - Mapping Frequencies")
plt.show()

# Optional: Visualize class mappings (if labels are known, for illustration only)
plt.figure(figsize=(10, 8))
for i, x in enumerate(X_scaled):
    w = som.winner(x)
    plt.text(w[0], w[1], str(y[i]),
             ha='center', va='center',
             bbox=dict(facecolor='white', alpha=0.5, lw=0))

plt.title("SOM with Iris Dataset Labels (for visualization)")
plt.xlim(-0.5, 6.5)
plt.ylim(-0.5, 6.5)
plt.grid()
plt.show()
```

output




## Program 6

Write a function to generate **uniform random numbers** in the interval [0, 1]. Use this function to generate 10 random samples and evaluate f(x) for each sample. What are the sampled function values?
Using the samples generated in the previous step, estimate the integral I using the **Monte Carlo method**.\
Soln)

```
import random

# Function to generate a uniform random number in [0,1]
def generate_uniform():
    return random.uniform(0, 1)
```

```
# Define the function f(x)
def f(x):
    return x**2

# Generate 10 random samples and evaluate f(x)
samples = [generate_uniform() for _ in range(10)]
f_values = [f(x) for x in samples]

# Display samples and function values
for i, (x, fx) in enumerate(zip(samples, f_values), 1):
    print(f"Sample {i}: x = {x:.4f}, f(x) = {fx:.4f}")

# Monte Carlo integration estimate of the integral from 0 to 1
# Integral ≈ average of f(x) over samples * (b - a), here b - a = 1 - 0 = 1
I_estimate = sum(f_values) / len(f_values)

print(f"\nEstimated integral I using Monte Carlo method: {I_estimate:.4f}")
```

output

```
Sample 1: x = 0.2450, f(x) = 0.0600
Sample 2: x = 0.7183, f(x) = 0.5159
Sample 3: x = 0.8600, f(x) = 0.7396
Sample 4: x = 0.9426, f(x) = 0.8885
Sample 5: x = 0.9421, f(x) = 0.8876
Sample 6: x = 0.5063, f(x) = 0.2564
Sample 7: x = 0.0047, f(x) = 0.0000
Sample 8: x = 0.1515, f(x) = 0.0230
Sample 9: x = 0.8020, f(x) = 0.6432
Sample 10: x = 0.1976, f(x) = 0.0390

Estimated integral I using Monte Carlo method: 0.4053
```

## Program 7

**Read a dataset and indicate the likelihood of an event occurring using Bayesian Networks**

Soln)

```
import pandas as pd
from pgmpy.models import DiscreteBayesianNetwork
```

```
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
# Sample dataset (you can replace this with your own CSV file)
data = pd.DataFrame([
    ['Sunny', 'Hot', 'High', 'False', 'No'],
    ['Sunny', 'Hot', 'High', 'True', 'No'],
    ['Overcast', 'Hot', 'High', 'False', 'Yes'],
    ['Rain', 'Mild', 'High', 'False', 'Yes'],
    ['Rain', 'Cool', 'Normal', 'False', 'Yes'],
    ['Rain', 'Cool', 'Normal', 'True', 'No'],
    ['Overcast', 'Cool', 'Normal', 'True', 'Yes'],
    ['Sunny', 'Mild', 'High', 'False', 'No'],
    ['Sunny', 'Cool', 'Normal', 'False', 'Yes'],
    ['Rain', 'Mild', 'Normal', 'False', 'Yes'],
    ['Sunny', 'Mild', 'Normal', 'True', 'Yes'],
    ['Overcast', 'Mild', 'High', 'True', 'Yes'],
    ['Overcast', 'Hot', 'Normal', 'False', 'Yes'],
    ['Rain', 'Mild', 'High', 'True', 'No']
], columns=['Outlook', 'Temperature', 'Humidity', 'Windy', 'Play'])


# Define the structure of the Bayesian Network
model = DiscreteBayesianNetwork([
    ('Outlook', 'Play'),
    ('Temperature', 'Play'),
    ('Humidity', 'Play'),
    ('Windy', 'Play')
])

# Learn CPDs from the dataset
model.fit(data, estimator=MaximumLikelihoodEstimator)

# Print CPDs
print("\nLearned CPDs:")
for cpd in model.get_cpds():
    print(cpd)

# Perform inference
infer = VariableElimination(model)
```

```
# Example query: Probability of playing given Outlook=Sunny and Humidity=High
query_result = infer.query(
    variables=['Play'],
    evidence={'Outlook': 'Sunny', 'Humidity': 'High'}
)

print("\nProbability of Play given Outlook='Sunny' and Humidity='High':")
print(query_result)
```

output

```
Learned CPDs:
+------------------+----------+
| Outlook(Overcast) | 0.285714 |
+------------------+----------+
| Outlook(Rain)     | 0.357143 |
+------------------+----------+
| Outlook(Sunny)    | 0.357143 |
+------------------+----------+
+-------------+------------------+----------+------------------+
| Humidity    | Humidity(High)   | ... | Humidity(Normal)  |
+-------------+------------------+----------+------------------+
| Outlook     | Outlook(Overcast) | ... | Outlook(Sunny)     |
+-------------+------------------+----------+------------------+
| Temperature | Temperature(Cool) | ... | Temperature(Mild) |
+-------------+------------------+----------+------------------+
| Windy       | Windy(False)     | ... | Windy(True)        |
+-------------+------------------+----------+------------------+
| Play(No)    | 0.5              | ... | 0.0                |
+-------------+------------------+----------+------------------+
| Play(Yes)   | 0.5              | ... | 1.0                |
+-------------+------------------+----------+------------------+
+------------------+----------+
| Temperature(Cool) | 0.285714 |
+------------------+----------+
| Temperature(Hot)  | 0.285714 |
+------------------+----------+
| Temperature(Mild) | 0.428571 |
```

```
+------------------+---------+
+-----------------+-----+
| Humidity(High)       | 0.5 |
+-----------------+-----+
| Humidity(Normal) | 0.5 |
+-----------------+-----+
+-------------+---------+
| Windy(False) | 0.571429 |
+-------------+---------+
| Windy(True)  | 0.428571 |
+-------------+---------+
```

Probability of Play given Outlook='Sunny' and Humidity='High':

```
+----------+------------+
| Play           |      phi(Play) |
+===========+=============+
| Play(No)  |            0.7653 |
+----------+------------+
| Play(Yes) |            0.2347 |
+----------+------------+
```

## Program 8

**Refer to the dataset in question 7 and indicate inferences based on the sequence of steps** .

**Soln)**

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
import matplotlib.pyplot as plt

# Step 1: Load the dataset
data = pd.DataFrame([
    ['Sunny', 'Hot', 'High', 'False', 'No'],
    ['Sunny', 'Hot', 'High', 'True', 'No'],
    ['Overcast', 'Hot', 'High', 'False', 'Yes'],
    ['Rain', 'Mild', 'High', 'False', 'Yes'],
    ['Rain', 'Cool', 'Normal', 'False', 'Yes'],
    ['Rain', 'Cool', 'Normal', 'True', 'No'],
    ['Overcast', 'Cool', 'Normal', 'True', 'Yes'],
    ['Sunny', 'Mild', 'High', 'False', 'No'],
    ['Sunny', 'Cool', 'Normal', 'False', 'Yes'],
    ['Rain', 'Mild', 'Normal', 'False', 'Yes'],
    ['Sunny', 'Mild', 'Normal', 'True', 'Yes'],
    ['Overcast', 'Mild', 'High', 'True', 'Yes'],
    ['Overcast', 'Hot', 'Normal', 'False', 'Yes'],
    ['Rain', 'Mild', 'High', 'True', 'No']
], columns=['Outlook', 'Temperature', 'Humidity', 'Windy', 'Play'])

# Step 2: Encode categorical features
label_encoders = {}
for column in data.columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    label_encoders[column] = le

# Step 3: Split features and target
X = data.drop('Play', axis=1)
y = data['Play']

# Step 4: Train Decision Tree Classifier
clf = DecisionTreeClassifier(criterion='entropy')  # using ID3-like entropy
clf.fit(X, y)

# Step 5: Visualize the tree (text-based)
feature_names = X.columns
tree_rules = export_text(clf, feature_names=list(feature_names))
print(tree_rules)
```

```
# Step 6: Visualize the tree graphically
plt.figure(figsize=(12, 6))
plot_tree(clf, feature_names=feature_names, class_names=label_encoders['Play'].classes_, filled=True)
plt.title("Decision Tree")
   plt.show()
```

output

```
|--- Outlook <= 0.50
|      |--- class: 1
|--- Outlook >  0.50
|      |--- Humidity <= 0.50
|      |      |--- Outlook <= 1.50
|      |      |      |--- Windy <= 0.50
|      |      |      |      |--- class: 1
|      |      |      |--- Windy >  0.50
|      |      |      |      |--- class: 0
|      |      |--- Outlook >  1.50
|      |      |      |--- class: 0
|      |--- Humidity >  0.50
|      |      |--- Windy <= 0.50
|      |      |      |--- class: 1
|      |      |--- Windy >  0.50
```

```
|   |   |   |--- Outlook <=  1.50
|   |   |   |    |--- class:  0
|   |   |   |--- Outlook >  1.50
|   |   |   |    |--- class:  1
```