Lokmanya Tilak Jankalyan Shikshan Sanstha's
PRIYADARSHINI COLLEGE OF ENGINEERING, NAGPUR
(An Autonomous Institute affiliated to Rashtrasant Tukadoji Maharaj Nagpur University)

DEPARTMENT OF INFORMATION TECHNOLOGY

MACHINE LEARNING - LAB MANUAL

Subject code: 25UIT632P

Semester: VI

INDEX

LOKMANYA TILAK JANKALYAN SHIKSHAN SANSTHA'S

PRIYADARSHINI COLLEGE OF ENGINEERING

An Autonomous Institute Affilliated to R.T.M. Nagpur University, Nagpur
Accredited with Grade 'A+' by NAAC
Near C.R.P.F. Campus, Hingna Road, Nagpur - 440 019 (Maharashtra) India
Phone : 07104 - 299648, Fax : 07104-299681
E-mail : principal.pce.ngp@gmail.com • Website: www.pcenagpur.edu.in
AICTE ID No. 1-5435581; DTE CODE No. 4123,
UNIVERSITY CODE No. : 278

Vision

To become one of the India's leading Engineering Institutes in both education and research. We are committed to provide quality and state-of-the-art technical education to our students so that they become Technologically competent and in turn contribute for creating a great society.

Mission

1. Fostering a dynamic learning environment that equips students with Technical expertise, problem-solving skills and a deep commitment to ethical practices.
2. To cultivate a culture of innovation, incubation, research and entrepreneurship that drives technological advancements.
3. To uphold the spirit of mutual excellence while interacting with stake holders of our Institutional ecosystem.
4. Promoting lifelong learning, professional growth and ensuring holistic development of students and the well being of society.

PRIYADARSHINI COLLEGE OF ENGINEERING PCE NAGPUR

Principal
Priyadarshini College of Engg.
Nagpur.

LOKMANYA TILAK JANKALYAN SHIKSHAN SANSTHA'S
PRIYADARSHINI COLLEGE OF ENGINEERING
(Recognised by A.I.C.T.E. New Delhi & Govt. of Maharashtra Affiliated to R.T.M. Nagpur University)
Near C.R.P.F. Campus, Hingna Road, Nagpur - 440 019 (Maharashtra) India
Phone : 07104 - 299648, Fax : 07104-299681
E-mail : principal.pce.ngp@gmail.com • Website: www.pcenagpur.edu.in
AICTE ID No. 5435581; DTE CODE No. 4123; UNIVERSITY CODE No. 278
Accredited with Grade A+by NAAC

Department of Information Technology

Date: 16/09/2023

Vision

To nurture the students with latest technology and research, helping them to become valuable contributors who can address the changing needs of the IT industry and society.

Mission

1. To create a holistic learning environment which empower the emerging graduates with advanced technical and value based education.
2. To attain professional excellence in the field of Information Technology and related areas.
3. To encourage transformative learning for research addressing the changing needs of society and fostering lifelong learning.

H.O.D.
Department of Information Tech
Priyadarshini College of Engg
Nagpur-19

PROGRAM EDUCATIONAL OBJECTIVES :

PEOs	Program Educational Objectives Statements
PEO1	Demonstrate strong technical expertise and uphold ethical values to excel in their professional careers.
PEO2	Apply advanced skills and knowledge in Information Technology to address real-world challenges and contribute to industry success.
PEO3	Pursue lifelong learning and engage in research and innovation to address societal needs and advance the field of Information Technology.

PROGRAM OUTCOMES:

Engineering Graduates will Able to:

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

PROGRAM SPECIFIC OUTCOMES :

PSOs	Program Specific Outcomes
PSO1	An ability to apply mathematical foundations, algorithmic principles and computer science theory in the modeling and design of software systems of varying complexity.
PSO2	An ability to work with Open-Source Software and use off the shelf utilities for program integration.

COURSE OUTCOMES :

Course Outcomes	Statement
CO1	Implement Core Python Libraries for Data Analysis
CO2	Apply Dimensionality Reduction Techniques
CO3	Build and Evaluate a K-Nearest Neighbors (KNN) Classifier

CO-PO & PSO MAPPING:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	3	2	2	2	3	-	-	-	-	-	-	2	3	3
CO2	3	3	2	3	2	-	-	-	-	-	-	2	3	2
CO3	3	3	3	3	3	-	-	-	-	-	-	2	3	2

List of Practical's:

Sr. No.	Name of Practical
1	Implementation of Python Basic Libraries such as Statistics, Math, Numpy and Scipy
2	Implementation of Python Libraries for ML application such as Pandas and Matplotlib.
3	Creation and Loading different types of datasets in Python using the required libraries and perform EDA
4	Implement Dimensionality reduction using Principle component Analysis method on a dataset iris
5	Write a program to demonstrate the working of the decision tree based ID3 algorithm by considering a dataset
6	Consider a dataset, use Random Forest to predict the output class. Vary the number of trees as follows and compare the results: i.20 ii.50 iii.100 iv.200 v.500
7	Write a Python program to implement Simple Linear Regression and plot the graph
8	Write a Python program to implement Logistic Regression for iris using sklearn and plot the confusion matrix.
9	Build KNN Classification model for a given dataset. Vary the number of k values as follows and compare the results: i. 1 ii. 3 iii. 5 iv. 7 v. 11
10	Implement Support Vector Machine for a dataset and compare the accuracy by applying the following kernel functions: i. Linear ii. Polynomial iii. RBF
I1	Case Study/ Mini Project.

Practical 01

Aim:

To implement and understand the usage of Python basic libraries such as Math, Statistics, NumPy, and SciPy for performing mathematical, statistical, and scientific computations.

Tools used:

Python 3.x

Jupyter Notebook / VS Code / PyCharm

Python Libraries: math, statistics, numpy, scipy

Theory:

Python provides several built-in and external libraries to support numerical and scientific computations. The Math library offers mathematical functions such as logarithms, square roots, and factorials. The Statistics library is used to compute descriptive statistical measures like mean, median, variance, and standard deviation. NumPy enables efficient numerical computations using arrays and matrix operations. SciPy, built on NumPy, provides advanced scientific and statistical functions widely used in data analysis and machine learning applications.

Algorithm:

Import the required Python libraries.

Perform basic mathematical operations using the Math library.

Compute statistical measures using the Statistics library.

Create arrays and perform numerical operations using NumPy.

Apply advanced statistical computations using SciPy.

Display the results.

Program:

```
# Import required libraries
import math
import statistics
import numpy as np
from scipy import stats

# Math library operations
print("Math Library Operations")
print("Square root of 25:", math.sqrt(25))
print("Factorial of 5:", math.factorial(5))
```

```
print("Logarithm of 100 (base 10):", math.log10(100))
print("Value of Pi:", math.pi)

# Statistics library operations
data = [10, 20, 30, 40, 50]
print("\nStatistics Library Operations")
print("Mean:", statistics.mean(data))
print("Median:", statistics.median(data))
print("Mode:", statistics.mode(data))
print("Variance:", statistics.variance(data))
print("Standard Deviation:", statistics.stdev(data))

# NumPy operations
array = np.array([1, 2, 3, 4, 5])
print("\nNumPy Operations")
print("Array:", array)
print("Sum:", np.sum(array))
print("Mean:", np.mean(array))
print("Standard Deviation:", np.std(array))
print("Square of elements:", np.square(array))

# SciPy operations
print("\nSciPy Operations")
print("Mean using SciPy:", stats.tmean(data))
print("Geometric Mean:", stats.gmean(data))
print("Harmonic Mean:", stats.hmean(data))
```

Output:

```
Math Library Operations
Square root of 25: 5.0
Factorial of 5: 120
Logarithm of 100 (base 10): 2.0
Value of Pi: 3.141592653589793

Statistics Library Operations
Mean: 30
Median: 30
Mode: 10
Variance: 250
Standard Deviation: 15.811388300841896

NumPy Operations
Array: [1 2 3 4 5]
Sum: 15
Mean: 3.0
Standard Deviation: 1.4142135623730951
Square of elements: [ 1  4  9 16 25 ]
```

SciPy Operations
Mean using SciPy: 30.0
Geometric Mean: 26.051710846973528
Harmonic Mean: 21.8978102189781

Conclusion:

The experiment successfully demonstrated the use of Python's Math, Statistics, NumPy, and SciPy libraries for mathematical and statistical computations. These libraries form the foundation for scientific computing and data analysis in Python.

Viva Questions and Answers

Q1. What is the use of the NumPy library in Python?
Answer: NumPy is used for efficient numerical computations using multi-dimensional arrays and vectorized operations.

Q2. How is the Statistics library different from NumPy?
Answer: The Statistics library is used for basic statistical calculations on small datasets, whereas NumPy is optimized for large datasets and array-based computations.

Q3. What types of functions are available in the Math library?
Answer: The Math library provides functions such as square root, logarithmic, trigonometric, exponential, factorial functions, and mathematical constants.

Q4. What is the purpose of the SciPy library?
Answer: SciPy is used for advanced scientific and statistical computations such as probability distributions, hypothesis testing, optimization, and numerical analysis.

Q5. Why is SciPy built on top of NumPy?
Answer: SciPy uses NumPy's efficient array structure and numerical capabilities to provide higher-level scientific and statistical functions.

Practical 02

Aim:

To implement Pandas and Matplotlib libraries for loading, analyzing, and visualizing a real-world CSV dataset for machine learning applications.

Tools used:

Python 3.x

Jupyter Notebook / VS Code / PyCharm

Python Libraries: pandas, matplotlib

Theory:

Pandas is a Python library used for data manipulation and analysis. It provides the DataFrame data structure, which allows efficient handling of structured data such as CSV files, including operations like filtering, aggregation, and statistical analysis.
Matplotlib is a data visualization library used to create plots and charts that help in understanding data distributions, trends, and relationships. In machine learning, these libraries are essential for exploratory data analysis (EDA) and preprocessing before model building.

Algorithm:

Import Pandas and Matplotlib libraries.

Load the Iris CSV dataset into a Pandas DataFrame.

Display the dataset and check its dimensions.

Generate descriptive statistics using Pandas.

Check for missing values in the dataset.

Visualize feature distributions and relationships using Matplotlib.

Program:

```
# Import required libraries
import pandas as pd
import matplotlib.pyplot as plt

# Load the Iris dataset
df = pd.read_csv("Iris.csv")

# Display first five records
print("Dataset Head:")
```

```
print(df.head())

# Display dataset shape
print("\nDataset Shape:", df.shape)

# Descriptive statistics
print("\nStatistical Summary:")
print(df.describe())

# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())

# Histogram of numerical features
df.hist(figsize=(8, 6))
plt.suptitle("Feature Distribution")
plt.show()

# Scatter plot between Sepal Length and Sepal Width
plt.figure()
plt.scatter(df["SepalLengthCm"], df["SepalWidthCm"])
plt.xlabel("Sepal Length (cm)")
plt.ylabel("Sepal Width (cm)")
plt.title("Sepal Length vs Sepal Width")
plt.show()
```

Output:

Dataset Head:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Dataset Shape: (150, 6)

Statistical Summary:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
max	7.900000	4.400000	6.900000	2.500000

Missing Values:

Id	0
SepalLengthCm	0

SepalWidthCm 0
PetalLengthCm 0
PetalWidthCm 0
Species 0

(Histograms and scatter plot are displayed graphically.)

Conclusion:

The Iris CSV dataset was successfully analyzed and visualized using Pandas and Matplotlib. These libraries are essential for exploratory data analysis and preprocessing in machine learning workflows.

Viva Questions and Answers

Q1. What is the role of Pandas in machine learning?
Answer: Pandas is used for data loading, cleaning, manipulation, and exploratory data analysis before applying machine learning algorithms.

Q2. What is a DataFrame?
Answer: A DataFrame is a two-dimensional, labeled data structure in Pandas used to store and manipulate tabular data.

Q3. Why is data visualization important in machine learning?
Answer: It helps in understanding data distribution, detecting outliers, and identifying relationships between features.

Q4. What does the describe() function do?
Answer: It provides descriptive statistics such as mean, standard deviation, minimum, and maximum values for numerical columns.

Q5. What types of plots can be created using Matplotlib?
Answer: Line plots, scatter plots, histograms, bar charts, and box plots.

Practical 3

Aim:

To create and load different types of datasets in Python and perform Exploratory Data Analysis (EDA) with extensive data visualization using Python libraries.

Tools used:

Python 3.x

Jupyter Notebook / VS Code / PyCharm

Python Libraries: pandas, numpy, matplotlib

Theory:

Datasets used in machine learning can be manually created or loaded from external sources such as CSV files. Exploratory Data Analysis (EDA) involves summarizing datasets using statistical methods and visualizations to understand data distribution, relationships, and patterns.
The Titanic dataset is a real-world dataset commonly used for data analysis and classification problems. Pandas is used for data handling, NumPy for numerical operations, and Matplotlib for visualizing trends, distributions, and categorical relationships in the data.

Algorithm:

Import the required Python libraries.

Create a dataset using NumPy and Python dictionaries.

Load the Titanic CSV dataset using Pandas.

Display dataset structure and basic information.

Perform statistical analysis and check missing values.

Visualize numerical and categorical data using various plots.

Program:

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# --------------------------------
# Creating dataset using NumPy
# --------------------------------
np_data = np.array([5, 10, 15, 20, 25])
print("NumPy Dataset:")
```

```
print(np_data)

# ------------------------------
# Creating dataset using Dictionary
# ------------------------------
dict_data = {
   "Experience": [1, 2, 3, 4, 5],
   "Salary": [30000, 40000, 50000, 60000, 70000]
}
df_manual = pd.DataFrame(dict_data)
print("\nCreated DataFrame:")
print(df_manual)

# ------------------------------
# Loading Titanic CSV Dataset
# ------------------------------
df = pd.read_csv("train.csv")

print("\nDataset Head:")
print(df.head())

print("\nDataset Shape:", df.shape)
print("\nDataset Information:")
print(df.info())

# ------------------------------
# Exploratory Data Analysis
# ------------------------------
print("\nStatistical Summary:")
print(df.describe())

print("\nMissing Values:")
print(df.isnull().sum())

# ------------------------------
# Visualization
# ------------------------------

# 1. Histogram of Age
plt.figure()
df["Age"].hist(bins=20)
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.title("Age Distribution of Passengers")
plt.show()

# 2. Bar chart of Survival Count
plt.figure()
```

```
df["Survived"].value_counts().plot(kind="bar")
plt.xlabel("Survived")
plt.ylabel("Count")
plt.title("Survival Count")
plt.show()

# 3. Bar chart of Passenger Class
plt.figure()
df["Pclass"].value_counts().plot(kind="bar")
plt.xlabel("Passenger Class")
plt.ylabel("Count")
plt.title("Passenger Class Distribution")
plt.show()

# 4. Scatter plot: Age vs Fare
plt.figure()
plt.scatter(df["Age"], df["Fare"])
plt.xlabel("Age")
plt.ylabel("Fare")
plt.title("Age vs Fare")
plt.show()
```

Output:

```
NumPy Dataset:
[ 5 10 15 20 25]

Created DataFrame:
   Experience  Salary
0           1   30000
1           2   40000
2           3   50000
3           4   60000
4           5   70000

Dataset Head:
   PassengerId  Survived  Pclass  ...   Fare Cabin  Embarked
0            1         0       3  ...   7.25   NaN         S
1            2         1       1  ...  71.28   C85         C
2            3         1       3  ...   7.92   NaN         S
...

Dataset Shape: (891, 12)

Statistical Summary:
              Age        Fare
count  714.000000  891.000000
mean    29.699118   32.204208
std     14.526497   49.693429
```

min 0.420000 0.000000
max 80.000000 512.329200

Missing Values:
Age 177
Cabin 687
Embarked 2

(Graphs such as histogram, bar charts, and scatter plot are displayed.)

Conclusion:

The Titanic dataset was successfully loaded and analyzed using Pandas, and multiple visualizations were created using Matplotlib. Exploratory Data Analysis helped in understanding passenger demographics, survival patterns, and data quality.

Viva Questions and Answers

Q1. What is the purpose of Exploratory Data Analysis (EDA)?
Answer: EDA is used to understand data distribution, detect missing values, identify patterns, and prepare data for machine learning.

Q2. Why is the Titanic dataset widely used in data science?
Answer: It is a real-world dataset with both numerical and categorical features suitable for classification and EDA tasks.

Q3. Which plot is suitable for showing data distribution?
Answer: Histogram is suitable for showing data distribution.

Q4. How can categorical data be visualized?
Answer: Using bar charts or count plots.

Q5. Why is visualization important before applying machine learning algorithms?
Answer: Visualization helps in understanding relationships, detecting outliers, and choosing appropriate preprocessing techniques.

Practical 04

Aim:

To implement **Principal Component Analysis (PCA)** for dimensionality reduction on the Iris dataset and visualize the transformed data.

Tools used:

Python 3.x

Jupyter Notebook / VS Code / PyCharm

Python Libraries: pandas, numpy, matplotlib, scikit-learn

Theory:

Dimensionality reduction is a technique used to reduce the number of input features while retaining most of the important information. **Principal Component Analysis (PCA)** is a statistical method that transforms the original correlated features into a smaller set of uncorrelated variables called **principal components**. These components capture the maximum variance present in the data.
PCA helps in reducing computational complexity, removing redundancy, and visualizing high-dimensional data in lower dimensions, which is especially useful in machine learning and data analysis.

Algorithm:

Import the required Python libraries.

Load the Iris dataset into a Pandas DataFrame.

Separate features and target labels.

Standardize the feature values.

Apply PCA to reduce dimensions to two components.

Visualize the reduced dataset using a scatter plot.

Program:

```
# Import required librariesimport pandas as pdimport numpy as npimport
matplotlib.pyplot as pltfrom sklearn.preprocessing import StandardScalerfrom
sklearn.decomposition import PCA
# Load Iris dataset
df = pd.read_csv("Iris.csv")
```

```
# Select features
features = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']
X = df[features]
y = df['Species']

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=2)
principal_components = pca.fit_transform(X_scaled)

# Create DataFrame of principal components
pca_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
pca_df['Species'] = y

# Explained varianceprint("Explained Variance Ratio:",
pca.explained_variance_ratio_)

# Visualization
plt.figure()for species in pca_df['Species'].unique():
    subset = pca_df[pca_df['Species'] == species]
    plt.scatter(subset['PC1'], subset['PC2'], label=species)

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA on Iris Dataset')
plt.legend()
plt.show()
```

Output:

```
Explained Variance Ratio: [0.72770452 0.23030523]
```

(A scatter plot showing Iris species distributed across two principal components is displayed.)

Conclusion:

Principal Component Analysis successfully reduced the Iris dataset from four features to two principal components while retaining most of the variance. PCA simplifies data visualization and improves efficiency in machine learning applications.

Viva Questions and Answers

Q1. What is Principal Component Analysis (PCA)?
Answer: PCA is a dimensionality reduction technique that transforms correlated features into uncorrelated principal components.

Q2. Why is standardization required before applying PCA?
Answer: Standardization ensures that all features contribute equally to the analysis by bringing them to the same scale.

Q3. What does explained variance ratio indicate?
Answer: It shows the proportion of total variance captured by each principal component.

Q4. How does PCA help in machine learning?
Answer: PCA reduces dimensionality, removes redundancy, improves computation efficiency, and helps in data visualization.

Q5. Is PCA a supervised or unsupervised technique?
Answer: PCA is an unsupervised learning technique.

Practical 05

Aim:

To write a Python program to demonstrate the working of the Decision Tree based ID3 algorithm using a given CSV dataset.

Tools used:

Python 3.x

Jupyter Notebook / VS Code / PyCharm

Python Libraries: pandas, numpy, math

Theory:

The ID3 (Iterative Dichotomiser 3) algorithm is a decision tree learning algorithm used for classification. It builds the decision tree by selecting the attribute that maximizes Information Gain at each step. Information Gain is calculated using Entropy, which measures the impurity or randomness in the dataset.
ID3 is a top-down, greedy algorithm and works well for categorical data. Decision trees are easy to interpret and widely used in machine learning for classification problems.

Algorithm:

Load the CSV dataset using Pandas.

Calculate the entropy of the target attribute.

Calculate information gain for each feature.

Select the attribute with maximum information gain as the root node.

Recursively repeat the process for each subset until the tree is formed.

Display the decision tree structure.

Program:

```
# Import required libraries
import pandas as pd
import math

# Load dataset
data = pd.read_csv("play_tennis.csv")

# Function to calculate entropy
def calculate_entropy(column):
```

```
    values = column.value_counts()
    entropy = 0
    for count in values:
        probability = count / len(column)
        entropy -= probability * math.log2(probability)
    return entropy

# Function to calculate information gain
def information_gain(data, feature, target):
    total_entropy = calculate_entropy(data[target])

    values = data[feature].unique()
    weighted_entropy = 0

    for value in values:
        subset = data[data[feature] == value]
        weighted_entropy += (len(subset) / len(data)) * calculate_entropy(subset[target])

    return total_entropy - weighted_entropy

# Function to build ID3 decision tree
def id3(data, features, target):

    # If all target values are same
    if len(data[target].unique()) == 1:
        return data[target].iloc[0]

    # If no features left
    if len(features) == 0:
        return data[target].mode()[0]

    # Select feature with maximum information gain
    gains = {}
    for feature in features:
        gains[feature] = information_gain(data, feature, target)

    best_feature = max(gains, key=gains.get)
    tree = {best_feature: {}}

    # Build subtrees
    for value in data[best_feature].unique():
        subset = data[data[best_feature] == value]
        remaining_features = features.copy()
        remaining_features.remove(best_feature)
        tree[best_feature][value] = id3(subset, remaining_features, target)

    return tree
```

```
# Define features and target
features = list(data.columns)
features.remove("PlayTennis")

# Build decision tree
decision_tree = id3(data, features, "PlayTennis")

# Display result
print("Decision Tree using ID3 Algorithm:")
print(decision_tree)
```

Output:

```
Decision Tree using ID3 Algorithm:
{'Outlook':
 {'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}},
  'Overcast': 'Yes',
  'Rain': {'Wind': {'Weak': 'Yes', 'Strong': 'No'}}}
}
```

Conclusion:

The ID3 algorithm was successfully implemented using a CSV dataset to construct a decision tree. The model effectively selected attributes based on information gain to perform classification.

Viva Questions and Answers

Q1. What is the ID3 algorithm?
Answer: ID3 is a decision tree algorithm that uses information gain to select the best attribute for splitting the data.

Q2. What is entropy in decision trees?
Answer: Entropy measures the impurity or randomness in a dataset.

Q3. What is information gain?
Answer: Information gain is the reduction in entropy after splitting the dataset on an attribute.

Q4. Is ID3 a supervised learning algorithm?
Answer: Yes, ID3 is a supervised learning algorithm used for classification.

Q5. What type of data is suitable for ID3?
Answer: ID3 works best with categorical data.

Practical 06

Aim:

To build a Random Forest classification model using a Kaggle dataset and compare predictive performance by varying the number of trees in the forest.

Tools used:

Python 3.x

Jupyter Notebook / VS Code / PyCharm

Python Libraries: pandas, numpy, matplotlib, scikit-learn

Theory:

Random Forest is an ensemble learning algorithm used for classification and regression tasks. It builds multiple decision trees during training and outputs the mode of the classes (classification) of individual trees. The number of trees in the forest (n_estimators) influences model stability and performance:

Low number of trees: faster but may underfit

High number of trees: slower but usually more accurate

This practical evaluates model performance for different values of trees: 20, 50, 100, 200, and 500, and compares metrics such as classification accuracy.

Algorithm:

Import the required libraries (pandas, sklearn).

Load the Heart Disease UCI dataset into a Pandas DataFrame.

Perform data preprocessing (handle missing values, encode categorical features if any).

Split the dataset into training and testing sets.

For each value of number of trees (n_estimators):

Build a Random Forest classifier.

Train the classifier on the training data.

Evaluate accuracy on the test data.

Store and compare results.

Plot accuracy vs number of trees.

Program:

```
# Import required libraries
import pandas as pd             # For data loading and manipulation
import numpy as np              # For numerical operations
import matplotlib.pyplot as plt  # For plotting graphs

# Import machine learning utilities from scikit-learn
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the dataset (Heart Disease dataset from Kaggle)
df = pd.read_csv("heart.csv")

# Separate input features (X) and target class (y)
X = df.drop("target", axis=1)   # All columns except target
y = df["target"]                # Target column

# Split dataset into training and testing sets
# 70% training data and 30% testing data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# List of number of trees to test in Random Forest
tree_counts = [20, 50, 100, 200, 500]

# List to store accuracy results
accuracies = []

# Loop through different values of number of trees
for n in tree_counts:

    # Create Random Forest classifier with n trees
    rf = RandomForestClassifier(n_estimators=n, random_state=42)

    # Train the model using training data
    rf.fit(X_train, y_train)

    # Predict output class for test data
    y_pred = rf.predict(X_test)

    # Calculate accuracy of the model
    acc = accuracy_score(y_test, y_pred)

    # Store accuracy result
```

```
    accuracies.append(acc)

    # Print number of trees and corresponding accuracy
    print(f"Number of Trees: {n}, Accuracy: {acc:.4f}")

# Plot accuracy vs number of trees
plt.figure()
plt.plot(tree_counts, accuracies, marker='o')
plt.xlabel("Number of Trees")
plt.ylabel("Accuracy")
plt.title("Random Forest Accuracy vs Number of Trees")
plt.show()
```

Output:

```
Number of Trees: 20, Accuracy: 0.8033
Number of Trees: 50, Accuracy: 0.8361
Number of Trees: 100, Accuracy: 0.8443
Number of Trees: 200, Accuracy: 0.8484

Number of Trees: 500, Accuracy: 0.8484
```

Conclusion:
Random Forest classification was successfully implemented to predict heart disease. As the number of trees increased, model accuracy generally improved, stabilizing around 200–500 trees. Random Forest offers robust performance compared to a single decision tree.

Viva Questions and Answers

Q1. What is a Random Forest classifier?
Answer: Random Forest is an ensemble algorithm that combines multiple decision trees and predicts the class by majority voting.

Q2. How does increasing the number of trees affect performance?
Answer: Increasing the number of trees reduces variance, often improves accuracy, and stabilizes the model, but increases computational cost.

Q3. Why do we split the dataset into training and testing sets?
Answer: To evaluate model performance on unseen data and avoid overfitting.

Q4. What metric is used to evaluate classification performance in this practical?
Answer: Classification accuracy was used.

Q5. Why do we use random_state in model building?
Answer: To ensure reproducibility of results by fixing the randomness in train-test split and model training

Practical 07

Aim:

To implement **Simple Linear Regression** using Python and visualize the relationship between independent and dependent variables using a regression line.

Tools used:

Python 3.x

Jupyter Notebook / VS Code / PyCharm

Python Libraries: pandas, numpy, matplotlib, scikit-learn

Theory:

Simple Linear Regression is a supervised learning algorithm used to model the linear relationship between one independent variable (X) and one dependent variable (Y). The model fits a straight line defined by the equation:

$$Y=mX+c$$

where m is the slope and c is the intercept.
The objective of linear regression is to minimize the error between actual and predicted values, usually using the **least squares method**. It is widely used for prediction and trend analysis.

Algorithm:

Import the required Python libraries.

Load the dataset into a Pandas DataFrame.

Separate independent and dependent variables.

Split the dataset into training and testing sets.

Create a Simple Linear Regression model.

Train the model using training data.

Predict output values.

Plot the regression line along with the actual data points.

Program:

```
# Import required libraries
import pandas as pd             # For handling datasets
import matplotlib.pyplot as plt  # For plotting graphs
from sklearn.linear_model import LinearRegression

# Step 1: Create a simple dataset
# X is the independent variable
# Y is the dependent variable
data = {
    'X': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Y': [2, 4, 5, 4, 5, 7, 8, 9, 10, 12]
}

# Convert dictionary to DataFrame
df = pd.DataFrame(data)

# Step 2: Separate input (X) and output (Y)
X = df[['X']]   # Independent variable must be in 2D format
y = df['Y']     # Dependent variable

# Step 3: Create Linear Regression model
model = LinearRegression()

# Step 4: Train the model using the dataset
model.fit(X, y)

# Step 5: Predict output values
y_pred = model.predict(X)

# Step 6: Plot the original data points
plt.scatter(X, y)

# Step 7: Plot the regression line
plt.plot(X, y_pred)

# Add labels and title
plt.xlabel("Independent Variable (X)")
plt.ylabel("Dependent Variable (Y)")
plt.title("Simple Linear Regression")

# Display the graph
plt.show()
```

Output:

A scatter plot showing actual data points

A straight regression line representing predicted values

(The regression line best fits the given data.)

Conclusion:

Simple Linear Regression was successfully implemented to model the relationship between two variables. The regression line effectively represents the trend in the data and can be used for prediction.

Viva Questions and Answers

Q1. What is Simple Linear Regression?
Answer: It is a supervised learning algorithm that models the linear relationship between one independent and one dependent variable.

Q2. What is the equation of a straight line in linear regression?
Answer: Y=mX+cY = mX + cY=mX+c

Q3. What does the slope represent in linear regression?
Answer: The slope indicates the rate of change of the dependent variable with respect to the independent variable.

Q4. Why do we split data into training and testing sets?
Answer: To evaluate model performance on unseen data and avoid overfitting.

Q5. Which library is commonly used to implement linear regression in Python?
Answer: The `scikit-learn` library.

Practical 08

Aim:

To implement **Logistic Regression** using Python on the Iris dataset and evaluate the model performance by plotting the **confusion matrix**.

Tools used:

Python 3.x

Jupyter Notebook / VS Code / PyCharm

Python Libraries: pandas, numpy, matplotlib, scikit-learn

Theory:

Logistic Regression is a supervised machine learning algorithm used for **classification problems**. It predicts the probability of a data point belonging to a particular class using a **sigmoid (logistic) function**.
Although called regression, it is primarily used for **binary and multi-class classification**. In multi-class problems like the Iris dataset, Logistic Regression uses the **one-vs-rest (OvR)** strategy.
A **confusion matrix** is used to evaluate classification models by showing the number of correct and incorrect predictions for each class.

Algorithm:

Import required Python libraries.

Load the Iris CSV dataset into a Pandas DataFrame.

Separate input features and output class labels.

Split the dataset into training and testing sets.

Create and train a Logistic Regression model.

Predict class labels for test data.

Generate and plot the confusion matrix.

Program:

Import required libraries

```
import pandas as pd                    # For loading and handling dataset
import matplotlib.pyplot as plt        # For plotting
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay


# Step 1: Load the Iris dataset from CSV file
df = pd.read_csv("Iris.csv")


# Step 2: Select input features (independent variables)
X = df[['SepalLengthCm', 'SepalWidthCm',
        'PetalLengthCm', 'PetalWidthCm']]


# Step 3: Select output class (dependent variable)
y = df['Species']


# Step 4: Split dataset into training and testing sets
# 70% training data and 30% testing data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)


# Step 5: Create Logistic Regression model
# max_iter is increased to ensure proper training
```

```
model = LogisticRegression(max_iter=200)


# Step 6: Train the model using training data
model.fit(X_train, y_train)


# Step 7: Predict class labels for test data
y_pred = model.predict(X_test)


# Step 8: Create confusion matrix
cm = confusion_matrix(y_test, y_pred)


# Step 9: Display confusion matrix graphically
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=model.classes_)
disp.plot()


# Add title
plt.title("Confusion Matrix - Logistic Regression (Iris Dataset)")
# Show the plot
plt.show()
```

Output:

A **confusion matrix plot** is displayed

Diagonal elements represent **correct predictions**

Non-diagonal elements represent **misclassifications**

(The model correctly classifies most Iris samples.)

Conclusion:

Logistic Regression was successfully implemented on the Iris dataset. The confusion matrix shows that the model achieves high classification accuracy with minimal misclassification.

Viva Questions and Answers

Q1. What is Logistic Regression used for?
Answer: Logistic Regression is used for classification problems to predict categorical output classes.

Q2. Why is Logistic Regression suitable for the Iris dataset?
Answer: The Iris dataset is a multi-class classification problem, which Logistic Regression can handle using the one-vs-rest approach.

Q3. What is a confusion matrix?
Answer: A confusion matrix is a table that compares actual class labels with predicted class labels.

Q4. What does the diagonal of the confusion matrix represent?
Answer: Correctly classified instances.

Q5. Why is `max_iter` used in Logistic Regression?
Answer: To increase the number of iterations for model convergence and avoid training errors.

Practical 09

Aim:

To build a **K-Nearest Neighbors (KNN)** classification model using Python and compare the classification accuracy by varying the number of neighbors **k = 1, 3, 5, 7, 11**.

Tools used:

Python 3.x

Jupyter Notebook / VS Code / PyCharm

Python Libraries: `pandas`, `matplotlib`, `scikit-learn`

Theory:

K-**Nearest Neighbors (KNN)** is a supervised machine learning algorithm used for classification. It classifies a new data point based on the **majority class of its nearest neighbors**.
The value of **k** represents the number of neighbors considered:

Small k → sensitive to noise

Large k → smoother decision boundary
Choosing an optimal k value improves classification performance.

Algorithm:

Import required Python libraries.

Load the Iris dataset from a CSV file.

Separate input features and output class labels.

Split the dataset into training and testing sets.

Train KNN models with different values of k.

Predict output classes for test data.

Calculate and compare accuracy for each k value.

Program:

```
# Import required libraries
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Step 1: Load the Iris dataset
df = pd.read_csv("Iris.csv")

# Step 2: Select input features
X = df[['SepalLengthCm', 'SepalWidthCm',
        'PetalLengthCm', 'PetalWidthCm']]

# Step 3: Select output class
y = df['Species']

# Step 4: Split 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
)

# Step 5: Define different k values
k_values = [1, 3, 5, 7, 11]

# Step 6: Train and test KNN for each k value
for k in k_values:

    # Create KNN classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    # Train the model
    knn.fit(X_train, y_train)

    # Predict class labels
    y_pred = knn.predict(X_test)

    # Calculate accuracy
    acc = accuracy_score(y_test, y_pred)

    # Display result
    print(f"K value: {k}, Accuracy: {acc:.4f}")
```

Output:

```
K value: 1, Accuracy: 0.9556K value: 3, Accuracy: 0.9778K value: 5, Accuracy:
0.9778K value: 7, Accuracy: 0.9556K value: 11, Accuracy: 0.9333
```

(Accuracy values may slightly vary based on data split.)

Conclusion:

The KNN classifier was successfully implemented on the Iris dataset. The results show that **moderate k values (3 and 5)** provide better classification accuracy compared to very small or large k values.

Viva Questions and Answers

Q1. What is K-Nearest Neighbors (KNN)?
Answer: KNN is a supervised learning algorithm that classifies data based on the majority class of its nearest neighbors.

Q2. What does the value of k represent?
Answer: k represents the number of nearest neighbors used for classification.

Q3. What happens if k is too small?
Answer: The model becomes sensitive to noise and may overfit.

Q4. What happens if k is too large?
Answer: The model may underfit and lose important local patterns.

Q5. Why is the Iris dataset suitable for KNN?
Answer: It is small, well-labeled, and has numerical features suitable for distance-based algorithms.

Practical 10

Aim:

To implement a **Support Vector Machine (SVM)** classifier using Python and compare classification accuracy by applying different kernel functions:
i. Linear
ii. Polynomial
iii. Radial Basis Function (RBF)

Tools used:

Python 3.x

Jupyter Notebook / VS Code / PyCharm

Python Libraries: `pandas`, `scikit-learn`

Theory:

Support Vector Machine (SVM) is a supervised learning algorithm used for classification. It finds an optimal hyperplane that best separates data points of different classes.
Kernel functions allow SVM to perform **non-linear classification** by transforming data into higher-dimensional space.
Common kernels include:

Linear kernel: used when data is linearly separable

Polynomial kernel: models complex relationships

RBF kernel: handles highly non-linear data

Algorithm:

Import required Python libraries.

Load the dataset from a CSV file.

Separate input features and output class labels.

Split the dataset into training and testing sets.

Train SVM models using different kernel functions.

Predict class labels and calculate accuracy.

Compare results for each kernel.

Program:

```
# Import required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


# Step 1: Load the Iris dataset from CSV file
df = pd.read_csv("Iris.csv")


# Step 2: Select input features (independent variables)
X = df[['SepalLengthCm', 'SepalWidthCm',
        'PetalLengthCm', 'PetalWidthCm']]


# Step 3: Select output class (dependent variable)
y = df['Species']


# Step 4: Split dataset into training and testing sets
# 70% data for training and 30% for testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)


# Step 5: List of kernel functions to compare
```

```
kernels = ['linear', 'poly', 'rbf']


# Step 6: Train and test SVM model for each kernel
for kernel in kernels:

    # Create SVM classifier using selected kernel
    svm_model = SVC(kernel=kernel)

    # Train the model using training data
    svm_model.fit(X_train, y_train)

    # Predict output classes for test data
    y_pred = svm_model.predict(X_test)

    # Calculate classification accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Print kernel type and corresponding accuracy
    print(f"Kernel: {kernel}, Accuracy: {accuracy:.4f}")
```

Output:

```
Kernel: linear, Accuracy: 0.9778Kernel: poly, Accuracy: 0.9556Kernel: rbf,
Accuracy: 0.9778
```

(Accuracy values may vary slightly depending on data split.)

Conclusion:

The Support Vector Machine classifier was successfully implemented using different kernel functions. The **Linear and RBF kernels** provided better accuracy on the Iris dataset compared to the Polynomial kernel.

Viva Questions and Answers

Q1. What is Support Vector Machine (SVM)?
Answer: SVM is a supervised learning algorithm that finds the optimal hyperplane to separate different classes.

Q2. What is the role of a kernel function in SVM?
Answer: Kernel functions transform data into higher dimensions to handle non-linear classification.

Q3. When is the linear kernel preferred?
Answer: When the data is linearly separable.

Q4. What is the advantage of the RBF kernel?
Answer: It can handle complex and non-linear data effectively.

Q5. Why is the Iris dataset suitable for SVM?
Answer: It is small, clean, and has numerical features, making it suitable for SVM classification.