Genba Sopanrao Moze Trust's
Parvatibai Genba Moze College of Engineering

Department of MCA

LABORATORY MANUAL

For the Academic Year 2024 - 2025

SYMCA- Semester III

Teaching Scheme:

PR: 04Hours/Week

Scheme: TW:25Marks PR:

50 Marks

PROGRAM OUTCOMES	
PO NO	**Program Outcome Description**
Po 1	Generate a proper 2-D data set of N points. Split the data set into Training Data set and Test Dataset.
Po 2	Download the open source software like WEKA or R or rJava. Document the distinct features and functionality of the software platform.
Po 3	Implement Naïve Bayes Classifier and K-Nearest Neighbor Classifier on Data set of your choice. Test and Compare for Accuracy and Precision.
Po 4	Implement K-Means Clustering and Hierarchical clustering on the proper data set of your choice. Compare their Convergence
Po 5	Design and implement SVM for classification with the proper data set of your choice. Comment on Design and Implementation for Linearly non separable Dataset.

OBJECTIVE:

- To study fundamentals of machine learning
- To acquaint with various machine learning algorithms.
- To become aware of various logic based and algebraic models in machine learning.
- To study trends in machine learning

Course Outcomes	
CO 1	Understand basic concepts of Machine Learning
CO 2	Understand classification concepts.
CO 3	Apply different regression and generalization techniques.
CO 4	Apply various logic Based and algebraic algorithms for real world applications.
CO 5	Use probabilistic models for machine learning
CO 6	Understand trends In Machine Learning

Guidelines for Student Journal:

The laboratory assignments are to be submitted by student in the form of journal. Journal consists of prologue, Certificate, table of contents, and **handwritten write-up** of each assignment (Title,

Objectives, Problem Statement, Outcomes, software & Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory- Concept in brief, algorithm, flowchart, Design, source code, tools, conclusion/analysis.

Program codes with sample output of all performed assignments are to be submitted as softcopy. As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers as part of write-ups and program listing to journal may be avoided. Use of DVD containing students programs maintained by lab In-charge is highly encouraged. For reference one or two journals may be maintained with program prints at Laboratory.

ASSIGNMENT NO: 1

Title: Generate a proper 2-D data set of N points. Split the data set into Training Data set and Test Dataset.

AIM: Generate a proper 2-D data set of N points. Split the data set into Training Data set and Test Dataset.

Explanation:

We use NumPy to generate a 2-D dataset with N points following a multivariate normal distribution.

The train_test_split function from scikit-learn is used to split the dataset into training and test sets.

The generated dataset is then visualized using matplotlib.

You can adjust the parameters, such as the mean, covariance matrix, and test size, based on your requirements. The key is to use the train_test_split function to split the dataset into training and test sets randomly.

Source Code:

```
import numpy as np

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt


# Set a random seed for reproducibility

np.random.seed(42)
```

```
# Generate a 2-D dataset with N points
N = 100
mean = [2, 3]  # Mean of the distribution
covariance = [[1, 0.5], [0.5, 1]]  # Covariance matrix

data = np.random.multivariate_normal(mean, covariance, N)

# Split the dataset into training and test sets
test_size = 0.2  # 80% training, 20% testing
train_data, test_data = train_test_split(data, test_size=test_size, random_state=42)

# Plot the generated dataset
plt.scatter(data[:, 0], data[:, 1], label='Original Data')
plt.title('Generated 2-D Dataset')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()

# Print the shapes of the training and test sets
print("Shape of Training Data:", train_data.shape)
print("Shape of Test Data:", test_data.shape)
```

ASSIGNMENT No: 2

Title: Download the open source software like WEKA or R or rJava. Document the distinct features and functionality of the software platform.

AIM: Download the open source software like WEKA or R or rJava. Document the distinct features and functionality of the software platform.

Steps to follow :

1. WEKA:

Overview:

- WEKA (Waikato Environment for Knowledge Analysis) is a collection of machine learning algorithms for data mining tasks.
- Developed at the University of Waikato, New Zealand, it is written in Java and provides a graphical user interface (GUI) for easy access to its functionalities.

Distinct Features:

1. **Diverse Algorithms:**
    - WEKA includes a wide range of machine learning algorithms for tasks like classification, regression, clustering, and association rule mining.
    - It supports popular algorithms such as decision trees, support vector machines, k-means clustering, and more.
2. **Data Preprocessing:**
    - Robust data preprocessing capabilities, including filtering, normalization, and handling missing values.
    - Interactive tools for exploring datasets and understanding data characteristics.
3. **Graphical User Interface (GUI):**
    - Provides a user-friendly GUI for both novice and experienced users.

- Supports interactive visualizations and model evaluations.

4. **Integration with Java Code:**
    - Can be used programmatically through Java code, allowing integration with other Java-based applications.
    - Provides an API for developers to incorporate WEKA functionalities into custom applications.
5. **Open Source:**
    - Released under the GNU General Public License (GPL), making it free and open-source.

2. R:

Overview:

- R is a programming language and environment specifically designed for statistical computing and graphics.
- It provides a comprehensive set of tools for data analysis, machine learning, and visualization.

Distinct Features:

1. **Statistical Analysis:**
    - R is widely used for statistical analysis, hypothesis testing, and data exploration.
    - A vast number of statistical packages and libraries are available.
2. **Data Visualization:**
    - Powerful plotting and visualization capabilities with libraries like ggplot2.
    - Supports the creation of static and interactive visualizations.
3. **Extensibility:**
    - A rich ecosystem of packages and libraries contributed by the community.
    - Users can easily create and share packages, making R highly extensible.
4. **Data Manipulation:**
    - Efficient data manipulation using tools like data frames and functions like

dplyr.

- Handles large datasets and provides data reshaping capabilities.

5. **Integration with Other Languages:**

    - R interfaces with other languages, enabling integration with libraries and tools from different ecosystems.

6. **Open Source:**

    - R is open-source software released under the GNU General Public License (GPL).

3. rJava:

Overview:

- rJava is an R package that provides a bridge between R and Java, allowing the use of Java code and libraries within R.

Distinct Features:

1. **Java Integration:**

    - Enables R to interact with Java classes and methods.
    - Allows calling Java code from R and vice versa.

2. **Access to Java Libraries:**

    - Facilitates the use of Java libraries and tools directly within R scripts.
    - Useful for leveraging existing Java functionality.

3. **Bidirectional Communication:**

    - Supports bidirectional communication between R and Java.
    - R functions can be called from Java, and Java methods can be invoked from R.

4. **Interoperability:**

    - Enhances interoperability between R and Java-based applications.
    - Useful for scenarios where specific functionality is available in Java but not in R.

5. **Open Source:**

    - rJava is an open-source package, available on CRAN (Comprehensive R Archive Network).

It's important to note that these software platforms serve different purposes, and the choice between them depends on the specific needs and requirements of your project or analysis.

ASSIGNMENT No: 3

TITLE: Implement Naïve Bayes Classifier and K-Nearest Neighbor Classifier on Data set of your choice. Test and Compare for Accuracy and Precision.

AIM: Implement Naïve Bayes Classifier and K-Nearest Neighbor Classifier on Data set of your

choice. Test and Compare for Accuracy and Precision.

Explanation:

Load the Iris dataset and split it into training and testing sets.

Train a Naïve Bayes classifier (GaussianNB) and a k-Nearest Neighbors classifier

(KNeighbors Classifier) on the training data.

Make predictions on the test set.

Evaluate and compare the accuracy and precision of both classifiers.

This script calculates and prints the accuracy and precision for both classifiers on the Iris

dataset. Adjust the dataset and classifiers as needed for your specific scenario.

Source Code:

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
from sklearn.metrics import accuracy_score, precision_score

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

# Naïve Bayes Classifier
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
nb_predictions = nb_classifier.predict(X_test)

# k-Nearest Neighbors Classifier
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, y_train)
knn_predictions = knn_classifier.predict(X_test)

# Evaluate and compare classifiers
def evaluate_classifier(predictions, y_test, classifier_name):
    accuracy = accuracy_score(y_test, predictions)
    precision = precision_score(y_test, predictions, average='weighted')
    print(f"{classifier_name} Classifier:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}\n")

# Evaluate Naïve Bayes Classifier
evaluate_classifier(nb_predictions, y_test, "Naïve Bayes")

# Evaluate k-Nearest Neighbors Classifier
evaluate_classifier(knn_predictions, y_test, "k-NN")
```

ASSIGNMENT No: 4

TITLE: Implement K-Means Clustering and Hierarchical clustering on the proper data set of your choice. Compare their Convergence

AIM: Implement K-Means Clustering and Hierarchical clustering on the proper data set of your choice. Compare their Convergence

Explanation:

Load the Iris dataset and standardize the data (important for K-Means).

Implement a function to plot the convergence of K-Means for a specified number of clusters.

Implement a function to plot the dendrogram for Hierarchical clustering.

Plot the convergence of K-Means with K=3.

Plot the dendrogram for Hierarchical clustering using complete linkage.

Feel free to experiment with different parameters and datasets. Keep in mind that the convergence behavior may vary based on the dataset and the characteristics of the data.

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import pairwise_distances_argmin_min
from scipy.cluster.hierarchy import dendrogram, linkage

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

# Standardize the data (important for K-Means)
X_standardized = (X - X.mean(axis=0)) / X.std(axis=0)

# Function to plot the convergence of K-Means
def plot_kmeans_convergence(X, n_clusters):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=1,
max_iter=10)
    kmeans.fit(X)
    labels = kmeans.labels_
    centers = kmeans.cluster_centers_

    # Plot data points
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', edgecolor='k')

    # Plot cluster centers
    plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='X', s=200, alpha=0.75,
label='Centers')
```

```
    plt.title(f'K-Means Clustering Convergence (K={n_clusters})')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.show()

# Plot the convergence of K-Means with K=3
plot_kmeans_convergence(X_standardized[:, :2], n_clusters=3)

# Function to plot the dendrogram for Hierarchical clustering
def plot_hierarchical_dendrogram(X, method='complete'):
    Z = linkage(X, method)
    plt.figure(figsize=(10, 6))
    dendrogram(Z)
    plt.title(f'Hierarchical Clustering Dendrogram ({method.capitalize()} Linkage)')
    plt.xlabel('Sample Index')
    plt.ylabel('Distance')
    plt.show()

# Plot the dendrogram for Hierarchical clustering with complete linkage
plot_hierarchical_dendrogram(X_standardized[:, :2], method='complete')
```

ASSIGNMENT No: 5

TITLE: Design and implement SVM for classification with the proper data set of your choice. Comment on Design and Implementation for Linearly non separable Dataset

AIM: Design and implement SVM for classification with the proper data set of your choice. Comment on Design and Implementation for Linearly non separable Dataset

Explanation:

Dataset Choice: I used the Iris dataset for simplicity, although it is inherently linearly separable.

The dataset is transformed to make it linearly non-separable by considering only classes 1 and 2.

Kernel Selection: For linearly non-separable datasets, using a non-linear kernel (e.g., radial basis function - RBF) is common.

In this example, I used a linear kernel (kernel='linear'), which assumes linear separability for simplicity. For non-linear separability, consider other kernel options.

Standardization: Features are standardized using Standard Scaler to ensure that all features have the same scale.

Model Evaluation: The accuracy of the SVM model is evaluated on the test set.

Visualization: Decision boundaries are visualized to illustrate the impact of the linear kernel on linearly non-separable data.

Note: When dealing with linearly non-separable datasets, consider using non-linear kernels (e.g., RBF) or exploring more advanced techniques such as kernel trick or nonlinear SVMs. The choice of the kernel and hyperparameters depends on the characteristics of the data. Additionally, model evaluation metrics beyond accuracy may be crucial for assessing performance on non-separable datasets.

Source Code:

```
import numpy as np

import matplotlib.pyplot as plt
```

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Iris dataset (for demonstration purposes)
iris = datasets.load_iris()
X = iris.data[:, :2]  # Using only the first two features for visualization
y = iris.target

# Make the dataset linearly non-separable
mask = y != 0  # Consider only classes 1 and 2
X = X[mask]
y = y[mask]

# 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)

# Standardize the features
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)

# Design and implement SVM
svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)
svm_classifier.fit(X_train_std, y_train)
```

```
# Predictions on the test set
y_pred = svm_classifier.predict(X_test_std)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Visualize the decision boundary
def plot_decision_boundary(X, y, model, title):
    h = .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 = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)


    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k', marker='o', s=100)
    plt.title(title)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()

# Visualize the decision boundary
plot_decision_boundary(X_train_std, y_train, svm_classifier, 'SVM Decision Boundary (Training Set)')
plot_decision_boundary(X_test_std, y_test, svm_classifier, 'SVM Decision Boundary (Test Set)')
```