Sri Sivasubramaniya Nadar College of Engineering, Chennai

(An autonomous Institution affiliated to Anna University)

Degree & Branch: B.E. Computer Science & Engineering
Semester: V
Subject Code & Name: ICS1512 - Machine Learning Algorithms Laboratory
Academic Year: 2025–2026 (Odd)
Batch: 2023–2028

Experiment 1: Working with Python Packages - Numpy, Scipy, Scikit-Learn, Matplotlib

- Explore the various functions and methods available in the following Python libraries:
  - **Numpy**
  - **Pandas**
  - **Scipy**
  - **Scikit-learn**
  - **Matplotlib**
- Understand the key operations such as:
  - Array manipulations
  - Data preprocessing
  - Mathematical computing
  - Machine learning workflows
  - Data visualization
- **Course Outcome:** CO1
  Knowledge Level: K2

Helpful Resources and Library Usage

1. NumPy

Use: Core package for numerical computation. Used in all ML pipelines for handling arrays, matrices, and fast operations on them.

Applications: Feature engineering, mathematical computation, matrix ops.

Basic Commands:

```
import numpy as np
a = np.array([1, 2, 3])
b = np.zeros((2, 2))
c = np.ones((3, 1))
d = np.arange(0, 10, 2)
e = np.reshape(a, (3, 1))
```

2. Pandas

Use: High-level data manipulation tool built on NumPy. Used for reading datasets, cleaning, transforming, and exploring data.

Applications: Preprocessing, handling CSV files, data wrangling.

Basic Commands:

```
import pandas as pd
df = pd.read_csv("data.csv")
print(df.head())
print(df.describe())
df["Age"] = df["Age"].fillna(df["Age"].mean())
```

3. SciPy

Use: Built on NumPy, includes modules for optimization, integration, interpolation, signal processing, and linear algebra.

Applications: Scientific computing, optimization in ML models.

4. Scikit-learn

Use: The most widely used ML library in Python. Offers tools for classification, regression, clustering, dimensionality reduction, model selection, and preprocessing.

Applications: Model training, evaluation, and pipeline creation. **Example: Logistic Regression on Iris Dataset**

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
```

```
iris = load_iris()
X = iris.data
y = iris.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
   test_size=0.2)

# Train model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Predict & evaluate
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
```

5. Matplotlib

Use: Visualization library for plotting 2D graphs, histograms, pie charts, scatter plots.
Applications: Data visualization and model result interpretation. **Basic Commands:**
Example: Plotting a Simple Line Graph

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]

plt.plot(x, y, marker='o')
plt.title("Simple Line Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.show()
```

Helpful Resource Links

- **NumPy:** Official Tutorials, W3Schools Guide
- **Pandas:** Official 10-Min Guide, W3Schools Guide
- **SciPy:** Official Special Functions, W3Schools Graphs
- **Scikit-learn:** Official Site, Tutorials

- **Matplotlib:** Official Tutorials, W3Schools Intro

Array Manipulations using NumPy

Explanation: Arrays are fundamental in ML. NumPy helps with fast manipulation such as reshaping, slicing, and broadcasting.

```
import numpy as np

# Create a 2D array
arr = np.array([[1, 2, 3], [4, 5, 6]])

# Transpose it
transposed = arr.T

# Reshape into a 3x2
reshaped = arr.reshape(3, 2)

# Broadcasting operation
result = arr + 10
```

Data Preprocessing using Pandas

Explanation: Real-world datasets often have missing or inconsistent data. Pandas helps in cleaning and transforming it.

```
import pandas as pd

# Load CSV data
df = pd.read_csv('students.csv')

# Fill missing values
df['Age'] = df['Age'].fillna(df['Age'].mean())

# Convert categorical to numeric
df['Passed'] = df['Result'].map({'Yes': 1, 'No': 0})

# Normalize a column
df['Score'] = (df['Score'] - df['Score'].min()) / (df['Score'].
   max() - df['Score'].min())
```

Mathematical Computing using SciPy

Explanation: SciPy enables advanced computations like integration, differentiation, and linear algebra used in ML theory.

```python
from scipy import integrate
import numpy as np

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

# Compute the definite integral of f from 0 to 5
area, _ = integrate.quad(f, 0, 5)
print("Area under curve:", area)
```

Machine Learning Workflow using Scikit-learn

Explanation: Scikit-learn simplifies the end-to-end ML pipeline—loading data, splitting, training, predicting, evaluating.

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load dataset
X, y = load_iris(return_X_y=True)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3)

# Train classifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Predict and evaluate
predictions = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, predictions))
```

Data Visualization using Matplotlib

Explanation: Matplotlib is used to visualize data trends, feature distributions, and ML model outputs.

```python
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [10, 20, 15, 25, 30]

# Plot with labels
plt.plot(x, y, marker='o', linestyle='--', color='green')
plt.title("Performance Over Time")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```

Experiment 2: Dataset Exploration and Model Identification

Task:
Explore public repositories such as the UCI Machine Learning Repository and Kaggle Datasets. Download the following datasets and identify the appropriate machine learning model to be used (e.g., Supervised, Unsupervised, Semi-supervised, Regression, Classification).

Course Outcome: CO1
Knowledge Level: K3

1. **Loan Amount Prediction**
   - **Type:** Supervised Learning
   - **Model Type:** Regression
   - **Reason:** The target variable (loan amount) is continuous. We are predicting numeric values based on features like income, credit history, etc.
2. **Handwritten Character Recognition**
   - **Type:** Supervised Learning

- **Model Type:** Classification
- **Reason:** Each image represents a labeled character (A–Z, 0–9). The task is to classify the image into one of the possible classes.

3. **Classification of Email Spam and MNIST Data**
   - **Type:** Supervised Learning
   - **Model Type:** Classification
   - **Reason:**
     - **Email Spam:** Binary classification (Spam or Not Spam).
     - **MNIST:** Multiclass classification of handwritten digits (0–9).

4. **Predicting Diabetes**
   - **Type:** Supervised Learning
   - **Model Type:** Classification
   - **Reason:** The outcome is binary (Diabetic or Non-diabetic). Based on input health parameters, we classify the patient status.

5. **Iris Dataset**
   - **Type:** Supervised Learning
   - **Model Type:** Classification
   - **Reason:** The dataset has labeled classes (Setosa, Versicolor, Virginica). Based on features like petal and sepal measurements, we classify the species.

Experiment 3: Machine Learning Workflow and Task Identification

Task:
Identify the type of ML task associated with each dataset. Explore the steps involved in the machine learning workflow.

Course Outcome: CO1
Knowledge Level: K2

Steps in a Machine Learning Workflow

1. **Loading the Dataset**
   Use libraries like `pandas` or `sklearn.datasets` to import datasets from local or public sources (e.g., UCI, Kaggle).

2. **Exploratory Data Analysis (EDA) and Visualization**
   Analyze data using:
   - Summary statistics: `describe()`, `info()`
   - Visualizations: Histograms, bar charts, scatter plots, heatmaps, box plots
3. **Data Preprocessing**
   Handle:
   - Missing values: fillna(), dropna()
   - Irrelevant features: drop()
   - Categorical encoding: LabelEncoder, OneHotEncoder
   - Scaling: MinMaxScaler, StandardScaler
4. **Feature Selection**
   Use statistical tests like:
   - `SelectKBest`
   - Chi-Square Test
   - ANOVA (f-classif)
5. **Data Splitting**
   Divide the dataset into:
   - Training set
   - Validation set (optional)
   - Testing set

   Using: `train_test_split()`
6. **Performance Evaluation**
   Metrics:
   - For Classification: Accuracy, Precision, Recall, F1-Score
   - For Regression: RMSE, MAE, R^2 score

   Use Confusion Matrix, ROC curve, residual plots for interpretation.

Identify the type of task performed for the above datasets and tabulate the details in the below table.

Dataset	**Type of ML Task**	**Feature Selection Technique**	**Suitable ML Algorithm**
Iris Dataset	Supervised (Classification)	ANOVA, SelectKBest	KNN, Logistic regression , Decision Tree, SVM
Loan Amount Prediction	Supervised (Regression)	Correlation, Mutual Info	Linear Regression, XGBoost Regressor
Predicting Diabetes	Supervised (Classification)	Chi-Square, f-classif	Logistic Regression, Random Forest
Classification of Email Spam	Supervised (Binary Classification)	Chi-Square, TF-IDF selection	Naive Bayes, SVM
Handwritten Character Recognition / MNIST	Supervised (Multiclass Classification)	PCA, Variance Threshold	CNN, Random Forest, SVM

Justification for Dataset Model Selection

1. **Iris Dataset**
   The Iris dataset contains labeled flower species based on numeric features like petal length, sepal width, etc. Since the labels are known, this is a **supervised classification** problem. Techniques like ANOVA or SelectKBest are useful for identifying the most significant features. Algorithms such as Logistic regression , K-Nearest Neighbors (KNN), Decision Tree, and Support Vector Machines (SVM) are suitable due to their effectiveness with low-dimensional, well-labeled datasets.

2. **Loan Amount Prediction**
   In this task, we aim to predict a continuous numeric value — the loan amount — based on applicant data (e.g., income, credit history). This makes it a **supervised regression** problem. Feature selection can be done using correlation or mutual information to find features that impact the target. Algorithms like Linear Regression or XGBoost Regressor perform well with numerical prediction tasks.

3. **Predicting Diabetes**
   The diabetes dataset typically contains health metrics like BMI, glucose level, etc., and aims to classify patients as diabetic or not. Since labels are available, it's a **supervised binary classification** task. Chi-Square test or f-classif can be used for selecting impactful features. Logistic Regression and Random Forest classifiers are commonly used for medical diagnosis problems due to their interpretability and accuracy.

4. **Classification of Email Spam**
   Spam detection is a binary classification problem where emails are labeled as ”spam” or ”not spam”. Text features (like word counts or TF-IDF scores) are extracted. This is **supervised learning**. Chi-Square test helps select the most relevant terms. Naive Bayes is commonly used for text classification due to its efficiency, while SVM is powerful for high-dimensional sparse data like text.

5. **Handwritten Character Recognition / MNIST**
   The MNIST dataset involves classifying handwritten digits (0–9), making it a **supervised multiclass classification** problem. Due to the high-dimensional image data, techniques like PCA (Principal Component Analysis) or Variance Threshold are useful to reduce dimensionality. Convolutional Neural Networks (CNNs) are ideal due to their superior performance in image recognition. Traditional models like Random Forest and SVM can also perform well with flattened pixel inputs.

Understanding CNN for Handwritten Character Recognition

Introduction:
Convolutional Neural Networks (CNNs) are deep learning models that are highly effective for image classification tasks, including handwritten character recognition. CNNs process image data in a hierarchical structure that allows them to learn spatial hierarchies of features.

Working of CNN in Handwriting Recognition:

(a) **Input Layer:**
The input to the CNN is a grayscale image (usually 28x28 pixels in MNIST). Each pixel represents an intensity value between 0 (black) and 255 (white), normalized to [0,1].

(b) **Convolutional Layer:**
Filters (kernels) slide over the image, performing element-wise multiplications and summing the result to produce a feature map. These filters detect edges, corners, curves, etc.

Example: A 3x3 filter scanning the 28x28 image will generate a 26x26 feature map (depending on padding).

(c) **Activation Function (ReLU):**
After convolution, the ReLU (Rectified Linear Unit) is applied to introduce non-linearity, making the network capable of learning complex patterns.

(d) **Pooling Layer:**
Pooling reduces the spatial size of the feature maps. Max pooling takes the maximum value from a 2x2 window, helping to reduce computation and control overfitting.

(e) **Flattening:**
After several convolution and pooling layers, the 2D feature maps are flattened into a 1D vector so it can be fed into the fully connected layers.

(f) **Fully Connected Layer:**
This layer acts like a traditional neural network where every node is connected to the next layer. It combines the learned features into a classification decision.

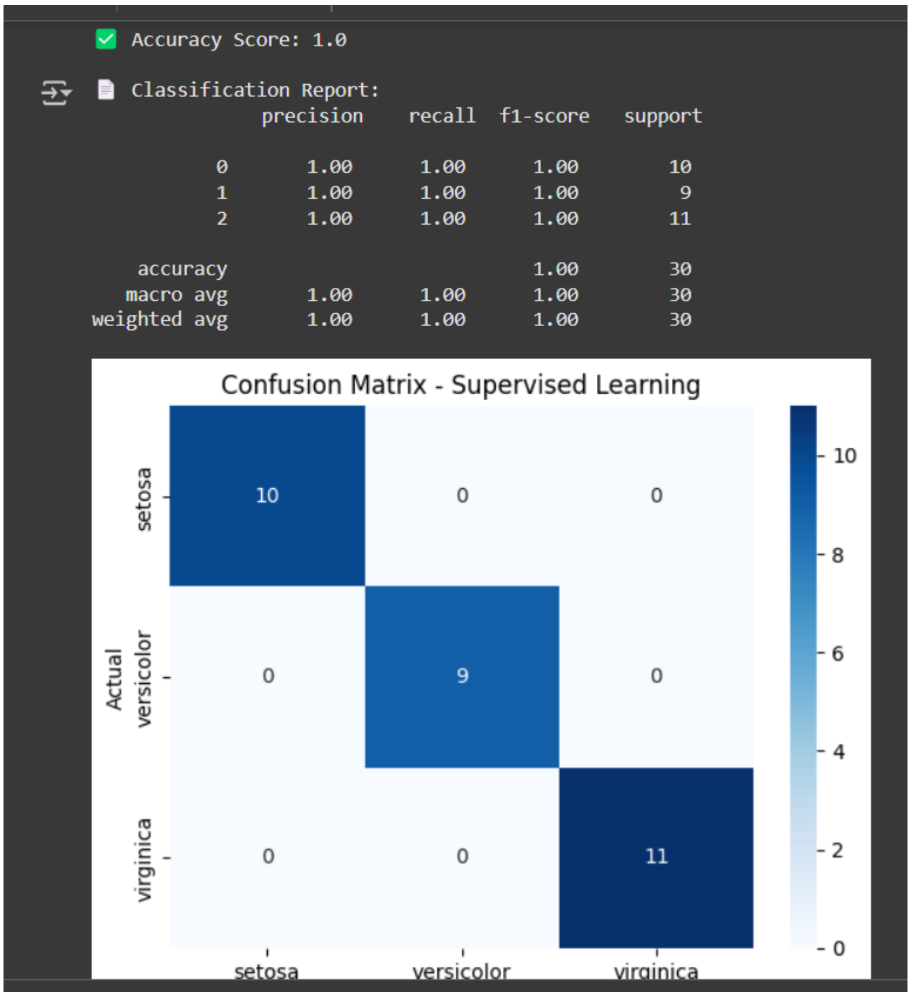
(g) **Output Layer (Softmax):**
The final layer uses the softmax activation function to output probability scores for each of the 10 digit classes (0–9).

Captured Output and Analysis

Google Colab Notebook Link

The full model implementation, preprocessing, visualization, and evaluation can be accessed through the following Colab notebook:

`Click here to view the notebook`



```
✅ Accuracy Score: 1.0

📄 Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Figure 1: Confusion Matrix - Supervised Classification (Iris Dataset)

Captured Output Screenshots

Inference and Summary Table

Metric/Observation	Value	Inference
Accuracy Score	1.0 (100%)	Model performed perfectly on test set; all predictions were correct.
Precision, Recall, F1-Score	All = 1.0	Each class (Setosa, Versicolor, Virginica) was predicted without error. Indicates balanced data and effective model.
Confusion Matrix	No misclassifications	All diagonal values are high and off-diagonals are zero. No false positives or negatives.
Strong Correlation	Petal Length & Width with Species 12	Feature correlation heatmap shows petal-based features are most influential in classification.
Low Correlation	Sepal Width with Species	Sepal width has negative/low correlation with target. Less impor-

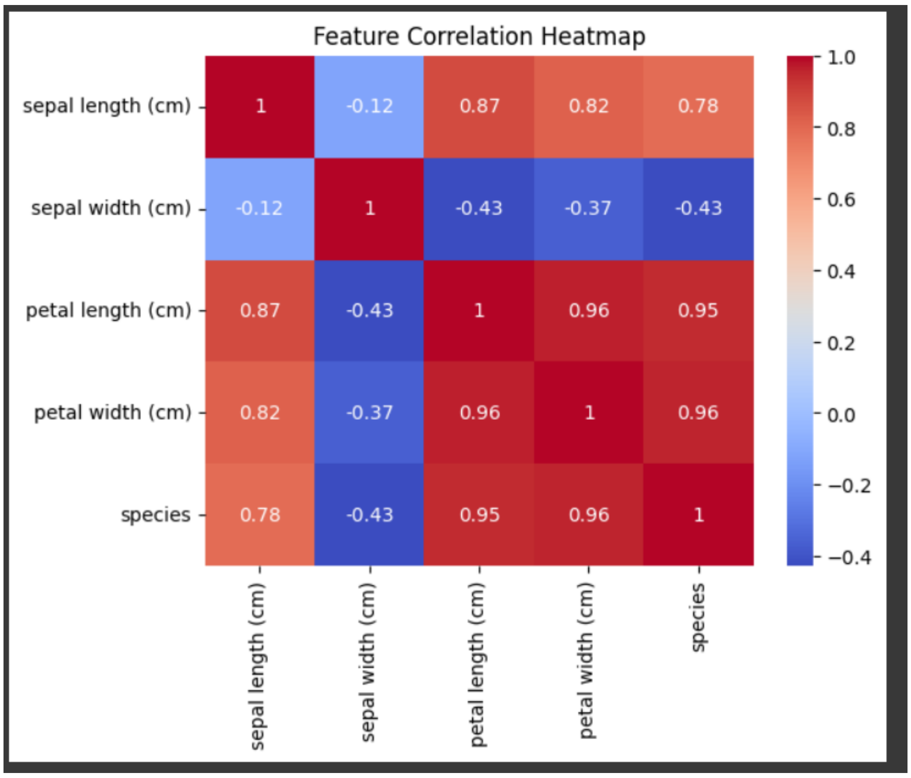


Figure 2: Feature Correlation Heatmap - Iris Dataset

Conclusion: The Decision Tree Classifier performed with 100% accuracy on the Iris dataset. The confusion matrix confirms perfect classification across all three classes. Feature correlation indicates that `petal length` and `petal width` are highly predictive of the species, making them key features for classification models.

Learning Outcomes

Experiment 1: Working with Python Packages (NumPy, SciPy, Scikit-learn, Matplotlib)

- Gained hands-on experience with key Python libraries for scientific computing and machine learning.
- Learned how to perform array manipulations, data preprocessing, and mathematical computations using NumPy and SciPy.
- Developed visualization skills using Matplotlib and understood ML workflows using Scikit-learn.

Experiment 2: Dataset Exploration and Model Selection

- Understood how to explore public repositories like UCI and Kaggle.

- Practiced identifying ML task types (classification, regression, etc.) for various real-world datasets.
- Learned how to choose suitable algorithms and feature selection methods based on dataset properties.

Experiment 3: ML Workflow with Evaluation Metrics

- Implemented end-to-end machine learning workflow including data loading, preprocessing, feature selection, model training, and evaluation.
- Interpreted performance using metrics such as accuracy, precision, recall, and confusion matrix.
- Visualized correlation and model predictions to gain insights into dataset structure.

Optional Task: Introduction to Deep Learning Frameworks

Optional Task: When and Where to Use TensorFlow, Keras, and PyTorch

Overview: TensorFlow, Keras, and PyTorch are three popular frameworks used in deep learning. Each has strengths suited to specific tasks or user preferences.

TensorFlow

- Developed by Google, TensorFlow is a powerful and scalable deep learning framework.
- Suitable for production-grade systems and deployment on cloud/mobile devices.
- Supports both low-level operations and high-level APIs (via Keras).
- Ideal for building complex models and serving them using TensorFlow Serving or TensorFlow Lite.
- Integrates well with TensorBoard for visualizing training metrics.

Basic Use Case:

- Use TensorFlow when:
  - You are working in Google's ecosystem (TPUs, TensorBoard, etc.)
  - You need strong deployment support
  - You want both flexibility and performance

Keras

- Keras is a high-level neural network API written in Python and runs on top of TensorFlow.
- It is user-friendly, modular, and designed for fast experimentation.
- Great for beginners due to its simple syntax and intuitive structure.
- Supports rapid prototyping and building models with minimal code.

Basic Use Case:

- Use Keras when:
  - You are new to deep learning and want quick results.
  - You prefer clean, readable code.
  - You need a TensorFlow-compatible model builder.

PyTorch

- Developed by Facebook, PyTorch is known for its flexibility and Pythonic nature.
- Widely used in academic research and experimentation.
- Offers dynamic computation graphs (eager execution), making debugging and custom logic easier.
- Excellent for building custom architectures and performing low-level operations.

Basic Use Case:

- Use PyTorch when:
  - You need more control over the model or training loop.
  - You're working on research or want flexibility over performance.
  - You prefer writing native Python code for deep learning.

Summary Comparison Table:

Framework	**Best For**	**Main Features**
TensorFlow	Production deployment, scalability	Scalable, mobile-ready, TensorBoard integration
Keras	Beginners and fast prototyping	Simple API, built on TensorFlow, modular
PyTorch	Research and experimentation	Dynamic graphs, native Python, flexible