Lab 04: Introduction to Scikit-learn and its Built-in Modules for Traditional Machine Learning

Lab 04: Introduction to Scikit-learn and its Built-in Modules for Traditional Machine Learning

1. Introduction

This lab introduces the **Scikit-learn (sklearn)** library, one of the most powerful and widely used machine learning frameworks in Python.

Students will explore its **modular structure**, covering:

- Data preprocessing and splitting
- Training and evaluating traditional ML models
- Understanding pipelines and model persistence

This lab emphasizes hands-on exposure to Scikit-learn's capabilities for **supervised learning**, including regression and classification tasks.

2. Objectives

By the end of this lab, students will be able to:

- Understand the architecture and purpose of Scikit-learn.
- Install and import Scikit-learn modules and consult its documentation.
- Load and split datasets using `train_test_split`.
- Implement and evaluate **Linear Regression** and **Logistic Regression** models.
- Compute model performance metrics using `accuracy_score`, `r2_score`, and `confusion_matrix`.
- Build a simple **machine learning pipeline** combining preprocessing and modeling steps.

3. Justification of Libraries and Tools

Library	Area	Justification
NumPy	Numerical Computing	Backbone of Scikit-learn; used for efficient numerical operations and arrays.
pandas	Data Manipulation	Used to handle tabular data structures (DataFrames) for model input/output.
matplotlib & seaborn	Visualization	Used to visualize dataset patterns and model results.
scikit-learn	Machine Learning	Provides standardized APIs for preprocessing, model training, evaluation, and pipelines.

4. Lab Procedure

4.1 Setup and Installation

Before using Scikit-learn, ensure that the required libraries are installed.

```
# Installation of required libraries
!pip install numpy pandas matplotlib seaborn scikit-learn
```

Import the core modules:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import r2_score, accuracy_score, confusion_matrix,
classification_report
```

4.2 Exploring Scikit-learn Documentation

Scikit-learn has an extensive documentation hub:

- **Main site:** https://scikit-learn.org/stable/
- **User guide:** https://scikit-learn.org/stable/user_guide.html
- **API reference:** https://scikit-learn.org/stable/modules/classes.html

Students are encouraged to explore sections on:

- `model_selection`
- `preprocessing`
- `metrics`
- `linear_model`

4.3 Step 1: Loading a Sample Dataset

Scikit-learn provides built-in datasets for quick experimentation.
We will use:

- **California Housing** for regression
- **Iris Dataset** for classification

```
from sklearn.datasets import fetch_california_housing, load_iris

# Load regression dataset
```

```
housing = fetch_california_housing(as_frame=True)
df_reg = housing.frame

# Load classification dataset
iris = load_iris(as_frame=True)
df_cls = iris.frame

print("Regression dataset shape:", df_reg.shape)
print("Classification dataset shape:", df_cls.shape)
```

4.4 Step 2: Data Preprocessing and Splitting

Split the data into **training and testing** subsets to avoid overfitting.

```
# Regression example
X_reg = df_reg.drop(columns=['MedHouseVal'])
y_reg = df_reg['MedHouseVal']

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg, y_reg, test_size=0.2, random_state=42
)

# Classification example
X_cls = df_cls.drop(columns=['target'])
y_cls = df_cls['target']

X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(
    X_cls, y_cls, test_size=0.25, random_state=42
)

print("Training samples (Regression):", X_train_reg.shape[0])
print("Training samples (Classification):", X_train_cls.shape[0])
```

4.5 Step 3: Model Training and Evaluation

A. Linear Regression Model

```
from sklearn.linear_model import LinearRegression

reg_model = LinearRegression()
reg_model.fit(X_train_reg, y_train_reg)

# Predictions
y_pred_reg = reg_model.predict(X_test_reg)

# Evaluation
print("R² Score:", r2_score(y_test_reg, y_pred_reg))
```

B. Logistic Regression Model

```
log_model = LogisticRegression(max_iter=200)
log_model.fit(X_train_cls, y_train_cls)
```

```
# Predictions
y_pred_cls = log_model.predict(X_test_cls)

# Evaluation
print("Accuracy:", accuracy_score(y_test_cls, y_pred_cls))
print("\nConfusion Matrix:\n", confusion_matrix(y_test_cls,
y_pred_cls))
print("\nClassification Report:\n", classification_report(y_test_cls,
y_pred_cls))
```

4.6 Step 4: Standardization and Pipelines

Scikit-learn pipelines help chain preprocessing and modeling steps.

```
from sklearn.pipeline import Pipeline

# Build a pipeline for regression
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', LinearRegression())
])

pipeline.fit(X_train_reg, y_train_reg)
y_pred_pipe = pipeline.predict(X_test_reg)

print("Pipeline R² Score:", r2_score(y_test_reg, y_pred_pipe))
```

4.7 Step 5: Model Persistence

You can save and reload trained models using `joblib`.

```
import joblib

# Save model
joblib.dump(reg_model, 'linear_model.pkl')

# Load model
loaded_model = joblib.load('linear_model.pkl')
print("Loaded Model R²:", r2_score(y_test_reg,
loaded_model.predict(X_test_reg)))
```

4.8 Visualization of Predictions

```
plt.figure(figsize=(8,5))
sns.scatterplot(x=y_test_reg, y=y_pred_reg)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Linear Regression: Actual vs Predicted")
plt.show()
```

5. Expected Outcomes

After completing this lab, students will:

- Be proficient in using Scikit-learn for regression and classification tasks.
- Understand the importance of preprocessing, model evaluation, and pipelines.
- Know how to explore official documentation for new model types and utilities.