Effat University
Department of Computer Science
Energy and Information Technology Research Center

LAB 2
Machine Learning with Scikit-Learn Basics

CS4082 – Machine Learning

A Hands-On Introduction to Building ML Models in Python

Prepared by: Dr. Naila Marir
Semester: Spring 2026

Contents

1 Lab Overview

In this lab, you will learn how to use **scikit-learn** (`sklearn`), the most popular Python library for Machine Learning. By the end, you will be able to load data, train a model, make predictions, and evaluate performance – all in just a few lines of code.

1.1 What You Will Learn

- How scikit-learn organizes ML workflows
- Loading and exploring built-in datasets
- Splitting data into training and testing sets
- Training a classifier (Decision Tree & K-Nearest Neighbors)
- Making predictions and evaluating accuracy
- Visualizing results with a confusion matrix
- Loading real-world CSV data and applying the full workflow

1.2 Prerequisites

- Basic Python knowledge (variables, functions, loops)
- Understanding of what classification means in ML
- Google Colab account (recommended) or local Python 3.8+ installation

Why Scikit-Learn?

Scikit-learn provides a consistent, clean API for dozens of ML algorithms. Once you learn the pattern (`fit` → `predict` → `evaluate`), you can apply it to almost any algorithm!

2 Part 1: Setting Up Your Environment

We will use Google Colab for this lab. No installation is needed – scikit-learn comes pre-installed!

2.1 Step 1: Open Google Colab

1. Go to `https://colab.research.google.com`
2. Click **“New Notebook”**
3. Rename it to: `Lab2_ML_Sklearn`

2.2 Step 2: Verify Installation

Run this cell to confirm everything is working:

```
import sklearn
print(f'scikit-learn version: {sklearn.__version__}')

import numpy as np
import matplotlib.pyplot as plt
print('All libraries loaded successfully!')
```

Expected Output

You should see the sklearn version number (e.g., 1.3.x or higher) and the success message. If you get an error, run: `!pip install scikit-learn`

3 Part 2: Loading and Exploring Data

Scikit-learn comes with several built-in toy datasets that are perfect for learning. We will use the famous **Iris dataset** – a classic dataset that contains measurements of 150 flowers from 3 species.

3.1 The Iris Dataset at a Glance

Property	Details
Samples	150 flowers (50 per species)
Features	4 measurements: sepal length, sepal width, petal length, petal width
Target Classes	0 = Setosa, 1 = Versicolor, 2 = Virginica
Task	Classify a flower into its species based on measurements

3.2 Step 1: Load the Dataset

```
from sklearn.datasets import load_iris

# Load the dataset
iris = load_iris()

# Features (X) and Labels (y)
X = iris.data          # Shape: (150, 4)
y = iris.target        # Shape: (150,)

# Let's see what we have
print(f'Feature names: {iris.feature_names}')
print(f'Target names:  {iris.target_names}')
print(f'Data shape:    {X.shape}')
print(f'First 3 rows:\n{X[:3]}')
```

3.3 Step 2: Quick Visualization

Let's plot the data to see if the classes are separable:

```
plt.figure(figsize=(8, 5))
colors = ['red', 'green', 'blue']

for i, name in enumerate(iris.target_names):
    mask = y == i
    plt.scatter(X[mask, 0], X[mask, 1],
                color=colors[i], label=name, alpha=0.7)

plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title('Iris Dataset - Sepal Features')
plt.legend()
plt.grid(True, alpha=0.3)
```

```
plt.show()
```

What to Notice

Setosa (red) is clearly separated from the other two. Versicolor and Virginica overlap slightly – this is what makes classification interesting!

Task 1: Explore the Data

- Print the first 10 rows of `X` and `y` side by side.
- Use `np.unique(y, return_counts=True)` to check the class distribution.
- Create another scatter plot using **petal length** (column 2) vs. **petal width** (column 3). Which features seem better for separating the classes?

4 Part 3: Splitting Data (Train/Test)

Before training, we must split our data into two parts: one for **training** the model and one for **testing** it. This is crucial to avoid overfitting – we need to know if the model can generalize to new, unseen data.

4.1 The Golden Rule

Never evaluate your model on the same data you used to train it. This is like a student writing the exam questions and then taking the same exam – the score would be meaningless!

4.2 Splitting with Scikit-Learn

```
from sklearn.model_selection import train_test_split

# Split: 80% training, 20% testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,          # 20% for testing
    random_state=42,        # For reproducibility
    stratify=y              # Keep class proportions balanced
)

print(f'Training set: {X_train.shape[0]} samples')
print(f'Testing set:  {X_test.shape[0]} samples')
```

4.3 Understanding the Parameters

Parameter	Meaning
`test_size=0.2`	Use 20% of data for testing (common choices: 0.2 or 0.3)
`random_state=42`	Fixes the random split so results are reproducible every time
`stratify=y`	Ensures each class has proportional representation in both sets

Task 2: Verify the Split

- Print the shape of `X_train`, `X_test`, `y_train`, and `y_test`.
- Use `np.unique(y_train, return_counts=True)` to confirm the classes are balanced.
- What happens if you remove `stratify=y`? Try it and compare the distributions.

5 Part 4: Training Your First Model

Now comes the exciting part! Scikit-learn uses a beautifully simple **3-step pattern** for every algorithm:

5.0.1 The Universal Scikit-Learn Pattern

```
# Step 1: Create the model
model = SomeAlgorithm()

# Step 2: Train it (fit)
model.fit(X_train, y_train)

# Step 3: Predict
predictions = model.predict(X_test)
```

Key Insight

This same pattern works for Decision Trees, KNN, SVM, Random Forest, Logistic Regression, and many more. Learn it once, use it everywhere!

5.1 Model A: Decision Tree Classifier

A Decision Tree learns a series of *if-then rules* from the data. Think of it as a flowchart that asks questions about features to reach a classification.

```
from sklearn.tree import DecisionTreeClassifier

# Step 1: Create
dt_model = DecisionTreeClassifier(random_state=42)

# Step 2: Train
dt_model.fit(X_train, y_train)

# Step 3: Predict
dt_predictions = dt_model.predict(X_test)

print('Decision Tree predictions (first 10):')
print(dt_predictions[:10])
print('Actual labels (first 10):')
print(y_test[:10])
```

5.2 Model B: K-Nearest Neighbors (KNN)

KNN classifies a new sample by looking at its **K closest neighbors** in the training data and taking a majority vote.

```
from sklearn.neighbors import KNeighborsClassifier

# Step 1: Create (k=5 neighbors)
knn_model = KNeighborsClassifier(n_neighbors=5)
```

```
# Step 2: Train
knn_model.fit(X_train, y_train)

# Step 3: Predict
knn_predictions = knn_model.predict(X_test)

print('KNN predictions (first 10):')
print(knn_predictions[:10])
```

Task 3: Train the Models

- Run both code blocks and compare the first 10 predictions. Are they different?
- Try changing `n_neighbors` to 3 and then to 10. Does the output change?

6 Part 5: Evaluating Model Performance

Making predictions is only half the story. We need to **measure how good** those predictions are. Scikit-learn provides several metrics for this purpose.

6.1 Accuracy Score

The simplest metric – what percentage of predictions were correct?

```
from sklearn.metrics import accuracy_score

dt_accuracy = accuracy_score(y_test, dt_predictions)
knn_accuracy = accuracy_score(y_test, knn_predictions)

print(f'Decision Tree Accuracy: {dt_accuracy:.2%}')
print(f'KNN Accuracy:           {knn_accuracy:.2%}')
```

6.2 Classification Report

A more detailed view showing precision, recall, and F1-score per class:

```
from sklearn.metrics import classification_report

print('=== Decision Tree Report ===')
print(classification_report(y_test, dt_predictions,
      target_names=iris.target_names))

print('=== KNN Report ===')
print(classification_report(y_test, knn_predictions,
      target_names=iris.target_names))
```

6.3 Quick Metric Definitions

Metric	What It Tells You
Precision	Of all samples predicted as class X, how many were actually X?
Recall	Of all actual class X samples, how many did the model find?
F1-Score	Harmonic mean of precision and recall – a balanced single metric.
Accuracy	Overall percentage of correct predictions across all classes.

6.4 Confusion Matrix (Visual)

The confusion matrix shows exactly where the model gets confused between classes:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

fig, axes = plt.subplots(1, 2, figsize=(12, 5))
```

```
# Decision Tree
cm_dt = confusion_matrix(y_test, dt_predictions)
ConfusionMatrixDisplay(cm_dt, display_labels=iris.target_names)\
    .plot(ax=axes[0], cmap='Blues')
axes[0].set_title('Decision Tree')

# KNN
cm_knn = confusion_matrix(y_test, knn_predictions)
ConfusionMatrixDisplay(cm_knn, display_labels=iris.target_names)\
    .plot(ax=axes[1], cmap='Greens')
axes[1].set_title('KNN')

plt.tight_layout()
plt.show()
```

Reading a Confusion Matrix

Diagonal values = correct predictions. Off-diagonal = errors. A perfect model has numbers only on the diagonal!

Task 4: Compare the Models

- Which model achieved higher accuracy?
- Look at the confusion matrices – which class was hardest to classify? Why?
- Which metric (precision or recall) would be more important in a **medical diagnosis** scenario? Explain briefly.

7 Part 6: Predicting New Samples

Now let's use our trained model to predict the species of a completely new flower that wasn't in our dataset:

```
import numpy as np

# A new flower measurement (sepal_l, sepal_w, petal_l, petal_w)
new_flower = np.array([[5.1, 3.5, 1.4, 0.2]])

# Predict with both models
dt_pred = dt_model.predict(new_flower)
knn_pred = knn_model.predict(new_flower)

print(f'Decision Tree says: {iris.target_names[dt_pred[0]]}')
print(f'KNN says:           {iris.target_names[knn_pred[0]]}')
```

Important Note

The input must be a **2D array** (notice the double brackets `[[...]]`). Scikit-learn expects the shape `(n_samples, n_features)`, even for a single sample.

Task 5: Predict New Flowers

- Try predicting these flowers and record the results:
  - Flower A: `[6.7, 3.0, 5.2, 2.3]`
  - Flower B: `[5.8, 2.7, 4.1, 1.0]`
  - Flower C: `[4.9, 3.1, 1.5, 0.1]`
- Do both models agree on all three? If not, which one do you trust more and why?

8 Part 7: Working with Your Own CSV Data

So far we used a built-in dataset. In real projects, your data will usually come as a **CSV file**. Let's learn how to load a CSV and apply the same scikit-learn workflow.

8.1 Step 1: Create a Sample CSV

First, let's create a small CSV file to work with. Run this code to generate one in Colab:

```
import pandas as pd
import numpy as np

# Create a simple student performance dataset
np.random.seed(42)
n = 100

data = {
    'study_hours':    np.round(np.random.uniform(1, 10, n), 1),
    'attendance_pct': np.round(np.random.uniform(40, 100, n), 1),
    'assignments':    np.random.randint(3, 10, n),
    'passed':         np.random.choice([0, 1], n, p=[0.35, 0.65])
}

df = pd.DataFrame(data)
df.to_csv('students.csv', index=False)
print('CSV saved! First 5 rows:')
print(df.head())
```

8.2 Step 2: Load the CSV with Pandas

Pandas is the go-to library for reading tabular data in Python:

```
import pandas as pd

# Read the CSV file
df = pd.read_csv('students.csv')

# Quick exploration
print(f'Shape: {df.shape}')
print(f'\nColumn types:\n{df.dtypes}')
print(f'\nBasic stats:\n{df.describe()}')
```

Useful Pandas Commands

`df.head()` shows first 5 rows, `df.info()` shows column types and null counts, `df.isnull().sum()` checks for missing values. Always explore before modeling!

8.3 Step 3: Prepare Features and Target

We need to separate the features (X) from the target label (y), just like we did with the Iris dataset:

```
# Features = all columns except 'passed'
X = df[['study_hours', 'attendance_pct', 'assignments']].values

# Target = the 'passed' column
y = df['passed'].values

print(f'Features shape: {X.shape}')
print(f'Target shape:   {y.shape}')
print(f'Class counts:   {np.unique(y, return_counts=True)}')
```

8.4 Step 4: Apply the Full Workflow

Now apply everything you learned – the same pattern works with any data source:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# Train
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Predict & Evaluate
predictions = model.predict(X_test)
print(f'Accuracy: {accuracy_score(y_test, predictions):.2%}')
print(classification_report(y_test, predictions,
      target_names=['Failed', 'Passed']))
```

Real-World Tip

Real CSV files often have missing values, text columns, and messy formatting. You would need to handle those with `df.dropna()`, `df.fillna()`, or encoding techniques before feeding data to scikit-learn.

Task 6: CSV Challenge

- Load the `students.csv` file and explore it with `df.describe()` and `df.info()`.
- Train both a Decision Tree and KNN model on this data. Which performs better?
- Try adding a new column to the CSV (e.g., `quiz_score`) and retrain. Does accuracy improve?
- **Bonus:** Upload your own CSV file to Colab and apply the same workflow!

9 Part 8: Summary and Key Takeaways

9.1 The Scikit-Learn Workflow

Every ML project in scikit-learn follows the same clean pattern:

Step	Action	Code
1	**Load Data**	`load_iris()`, `pd.read_csv()`, or your own data
	Split Data	`train_test_split(X, y, ...)`
3	**Create Model**	`model = Algorithm()`
	Train	`model.fit(X_train, y_train)`
5	**Predict**	`model.predict(X_test)`
	Evaluate	`accuracy_score()`, `classification_report()`

9.2 What to Explore Next

- **Try other algorithms:** Replace the classifier with `SVC()` from `sklearn.svm` or `RandomForestClassifier()` and compare results.
- **Try a different dataset:** Use `load_wine()` or `load_digits()` with the exact same workflow.
- **Feature scaling:** Learn about `StandardScaler` – some algorithms (like KNN) work better with scaled features.

10 Submission Requirements

10.1 What to Submit

1. Your completed Colab notebook (`.ipynb`) with all code cells executed.
2. A short paragraph (5–7 sentences) comparing Decision Tree vs. KNN performance and explaining which model you would choose for this task and why.

10.2 Grading Rubric

Criterion	Points	Weight
Data loading and exploration (Tasks 1–2)	15	15%
Model training and predictions (Tasks 3 & 5)	25	25%
Evaluation and analysis (Task 4)	20	20%
CSV data exercise (Task 6)	20	20%
Written comparison and reflection	20	20%
Total	**100**	**100%**

Final Tip

Machine Learning is learned by doing! Don't just copy the code – try changing parameters, using different datasets, and breaking things. That's how real understanding develops.