Lab 9: Introduction to Machine Learning – Supervised Learning

Objectives:

- Understand the basics of supervised learning and its applications.
- Learn how to implement a machine learning model using **Logistic Regression**.
- Understand the process of data preprocessing, model training, and evaluation.

Dataset: In this lab, we will use the **Iris Dataset**, which contains features about three species of Iris flowers. However, for this lab, we will focus on classifying **two species**: Setosa and Versicolor.

image.png

1. Importing the libraries

Think of this like getting your kitchen ready before cooking.

```
# Importing necessary libraries
# 'load_iris' to load the iris dataset from sklearn
# 'pandas' for creating and manipulating DataFrames

from sklearn.datasets import load_iris
import pandas as pd
```

2. Load and inspect the data

```
# get the dataset object (features + labels + names)
# iris.data contains the features and iris.feature_names provides column names
iris = load_iris()
```

```
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
    - sepal length in cm
    - sepal width in cm
    - petal length in cm
    - petal width in cm
    - class:
            - Iris-Setosa
            - Iris-Versicolour
            - Iris-Virginica

:Summary Statistics:

============== ==== ==== ======= ===== ====================
                Min  Max   Mean    SD   Class Correlation
============== ==== ==== ======= ===== ====================
sepal length:   4.3  7.9   5.84   0.83    0.7826
sepal width:    2.0  4.4   3.05   0.43   -0.4194
petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
============== ==== ==== ======= ===== ====================

:Missing Attribute Values: None
:Class Distribution: 33.3% for each of 3 classes.
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.
```

```
This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. dropdown:: References

  - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
    Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
    Mathematical Statistics" (John Wiley, NY, 1950).
  - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
    (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
  - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
    Structure and Classification Rule for Recognition in Partially Exposed
    Environments".  IEEE Transactions on Pattern Analysis and Machine
    Intelligence, Vol. PAMI-2, No. 1, 67-71.
```

```python
# check feature names
print(iris.feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

```python
# check dataset shape
print(iris.data.shape)
```

```
(150, 4)
```

```python
# check target names
print(iris.target_names)
```

```
['setosa' 'versicolor' 'virginica']
```

```python
iris.data[:5]  # display first 5 rows of feature data
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

```
# Create a DataFrame from the iris dataset for easier manipulation
# iris.data contains feature data, iris.feature_names contains column names
iris_data = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_data.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
import numpy as np
# check unique target labels
print(np.unique(iris.target))
# iris.target
```

```
[0 1 2]
```

```
# Add the target labels to the DataFrame ('species')
# iris.target contains the numeric representation of the species (Setosa, Versicolor, Virginica)
iris_data['species'] = iris.target
iris_data.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
# Mapping the target numbers to actual species names
# 0 -> Setosa, 1 -> Versicolor, 2 -> Virginica
iris_data['species'] = iris_data['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})

# Display the first few rows of the dataset to understand its structure
iris_data.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

3. Choose the learning problem (binary classification)

```
# Filter the data to only include Setosa and Versicolor species
# This is done to simplify the problem to a binary classification problem
iris_data = iris_data[iris_data['species'].isin(['setosa', 'versicolor'])]
```

```
# Split the dataset into features (X) and target labels (y)
# X will contain the feature columns, and y will contain the target column (species)
X = iris_data[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']]
y = iris_data['species']
```

```
print(X.head())
print(X.count())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
sepal length (cm)    100
sepal width (cm)     100
petal length (cm)    100
petal width (cm)     100
dtype: int64
```

```
print(y.head())
print(y.count())
```

```
0    setosa
1    setosa
2    setosa
3    setosa
4    setosa
Name: species, dtype: object
100
```

Why:

- Binary classification is the simplest intro to Logistic Regression.
- X = inputs (predictors), y = the label we're trying to predict.

```
# Import the train_test_split function from sklearn to split the dataset
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets (80% for training and 20% for testing)
# random_state ensures the split is reproducible
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Why:

- Prevents “cheating” (overfitting).
- test_size=0.2 keeps 20% for a final unbiased check.
- random_state makes results reproducible.

5. Pick and train a model (Logistic Regression)

```
# Import LogisticRegression from sklearn
# This is the main class for performing logistic regression

from sklearn.linear_model import LogisticRegression

# Create an instance of the Logistic Regression model
# This model will be used for binary classification
model = LogisticRegression()

# Train the model using the training data (X_train, y_train)
# The model learns from the input features and the target labels
model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

Why:

.fit() adjusts internal weights so predictions match labels as well as possible.

Analogy: A coach watches training and tunes strategies (weights) to win matches.

6. Make predictions

Ask the trained model to guess labels for unseen test data.

```
# Use the trained model to make predictions on the test set (X_test)
# y_pred will contain the predicted species for the test set
y_pred = model.predict(X_test)
print(y_pred)
```

```
['versicolor' 'versicolor' 'versicolor' 'setosa' 'setosa' 'setosa'
 'setosa' 'versicolor' 'setosa' 'setosa' 'setosa' 'setosa' 'versicolor'
 'setosa' 'versicolor' 'setosa' 'versicolor' 'versicolor' 'setosa'
 'setosa']
```

7. Evaluate the model

We’ll quantify how good those guesses are.

```
# Import evaluation metrics from sklearn
# We will use accuracy, precision, and recall to assess the model's performance
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Calculate accuracy, precision, and recall based on the test set predictions
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label='setosa')  # Precision for 'setosa' class
recall = recall_score(y_test, y_pred, pos_label='setosa')  # Recall for 'setosa' class

# Print the evaluation metrics
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
```

- Accuracy: 1.0: out of all test samples, how many did we get right?

The model made 100% correct predictions. Every instance was classified correctly, so the overall accuracy is perfect.

- Precision: 1.0: (for 'setosa'): when we said "setosa", how often were we correct? (avoid false alarms)

When the model predicted Versicolor, it was always correct. There were no false positives (no Setosa was predicted as Versicolor).

- Recall: 1.0: (for 'setosa'): out of all real "setosa", how many did we catch? (avoid misses)

The model identified all actual Versicolor instances correctly. There were no false negatives (no Versicolor was missed and incorrectly classified as Setosa).

In short, the model performed perfectly in this task: it made no mistakes in both predicting Versicolor and identifying all Versicolor samples.

8. Confusion matrix (error breakdown)

See exactly where mistakes happen.

```
# Import the confusion_matrix function from sklearn
# This will help us understand how well the model classifies different species
from sklearn.metrics import confusion_matrix

# Print the confusion matrix to see how many correct and incorrect predictions were made
# The confusion matrix shows true positives, true negatives, false positives, and false negatives
print(confusion_matrix(y_test, y_pred))
```

```
[[12  0]
 [ 0  8]]
```

This matrix represents the predictions made by your logistic regression model, and it shows how well the model classified the species as Setosa (Class 0) and Versicolor (Class 1). Let's interpret it:

Setosa = 0, Versicolor = 1 [[TP_setosa, FN_setosa], [FP_setosa, TP_versicolor]]

- Top-left = setosa correctly predicted setosa (true positives for setosa)
- Top-right = setosa predicted as versicolor (missed setosa = false negatives)
- Bottom-left = versicolor predicted as setosa (false positives for setosa)
- Bottom-right = versicolor correctly predicted versicolor

  0 1

0 [[12 0] 1 [0 8]]

True Positive (TP): The model correctly predicted Versicolor as Versicolor. This is the value in the bottom-right corner of the matrix: 8.

True Negative (TN): The model correctly predicted Setosa as Setosa. This is the value in the top-left corner of the matrix: 12.

False Positive (FP): The model incorrectly predicted Versicolor as Setosa. This is the value in the top-right corner of the matrix: 0.

False Negative (FN): The model incorrectly predicted Setosa as Versicolor. This is the value in the bottom-left corner of the matrix: 0.

Discussion and Conclusion

- **Accuracy**: This is the proportion of correctly predicted instances over the total number of instances.

- **Precision and Recall**: These metrics help us understand how well the model is performing with respect to a specific class (e.g., Setosa). Precision is the fraction of relevant instances retrieved, while recall is the fraction of relevant instances that were retrieved.

Further Learning:

- Explore how different hyperparameters affect the performance of Logistic Regression.
- Try using other classification algorithms (e.g., KNN, SVM) and compare their performance with Logistic Regression.