Department of Computer Science and Engineering PES University, Bangalore, India

Lecture Notes
Python for Computational Problem Solving UE25CS151A

Lecture 73

Introduction to Machine Learning in python, Loading datasets and Simple Model building (Scikit-learn)

Prepared By,
Prof. Kundhavai K R,
Assistant Professor
Dept. of CSE, PESU

Objective: Understand the basic idea of ML, what a dataset is, and how to build and check a simple model using the scikit-learn library.

What is Machine Learning?

At its core, Machine Learning (ML) is about teaching computers to learn from data.

Instead of writing *explicit rules* (like if-else statements) for every possible situation, we feed the computer examples (data) and let it learn the patterns itself.

Simple Analogy:

- **Traditional Programming:** You write a specific rule:

  "If an email contains the words **'free money'** or **'congratulations, you won!'**, send it to spam."

- **Machine Learning:** You give the computer 10,000 emails *already labeled* as 'Spam' or 'Not Spam'. It *learns* the patterns that make an email spammy (words, senders, links, etc.).
- **Real-world examples:**
  - YouTube/Netflix → recommending videos
  - Gmail → spam detection
  - Amazon → product recommendations
  - Banks → fraud detection
  - Healthcare → disease prediction

Key idea:

Machine learning = Data + Algorithm → Model → Predictions

Types of Machine Learning: There are mainly three types of ML:

(a) Supervised Learning: The model is trained using labeled data (data with answers). Example: Predicting house prices using area, location, etc.

Example:

Area (sq ft)	Bedrooms	Price (₹)
1000	2	50,00,000
1500	3	70,00,000

We train a model that learns this relationship and predicts price for a new house.

Common algorithms:

- Linear Regression
- Decision Trees
- Support Vector Machines (SVM)

(b) Unsupervised Learning: Data has **no labels**; the model finds hidden patterns. Example: Grouping customers based on spending habits.

Common algorithms:

- K-Means Clustering
- Hierarchical Clustering

(c) Reinforcement Learning: The model **learns by trial and error** using rewards and penalties. Example: Self-driving cars, game-playing AI (like chess).

Summary:

Type	Input	Output	Example
Supervised	Labeled data	Predictions	Email spam detection
Unsupervised	Unlabeled data	Groups or patterns	Market segmentation
Reinforcement	Actions & rewards	Strategy	Game AI

Why Python for ML?

- It's the most popular language for data science and ML.
- It has amazing, easy-to-use libraries.

What is scikit-learn (sklearn)?

This is the most important library for basic ML in Python.

It's our "toolkit." It has everything we need for today:

- Sample datasets to practice on.
- ML models (the "brains").
- Tools to check how well our model did.

Understanding Datasets:

How do machines learn from a dataset?

- A dataset is just a table of information, like an Excel spreadsheet. It has **two key parts:**

1. Features (The "Questions")

- These are the *characteristics* or *inputs* you use to make a decision.
- In Python, this is almost always stored in a variable named X.

- **Example**: To identify a flower, what would you measure?
  - Petal length
  - Petal width
  - Sepal length
  - Sepal width

(These four are the features.)

2. Labels (The "Answers")

- This is the *answer* or *output* you are trying to predict.
- In Python, this is almost always stored in a variable named y.
- **Example**: The *species* of the flower.
  - 'Setosa'
  - 'Versicolor'
  - 'Virginica'

(This is the label.)

Loading Our First Dataset: (The Iris Dataset)

- The Iris dataset is the **"Hello, World!"** of machine learning.
- It contains measurements for 150 Iris flowers, with 4 features (measurements) and 1 label (the species).
- iris dataset is built right into the scikit-learn library, so it's easy to load.
- **https://www.kaggle.com/datasets/uciml/iris - iris dataset**

Install scikit-learn Module:

```
pip install scikit-learn
```

Let's load it with scikit-learn:

```
from sklearn.datasets import load_iris
```

```
# 1. Load the dataset
iris = load_iris()
```

```
# 2. Separate our features (X) and labels (y)
X = iris.data
y = iris.target
```

```
# Let's see what we have
print(f"Shape of features (X): {X.shape}")
print(f"Shape of labels (y): {y.shape}")
# What does this mean?
# (150, 4) -> 150 rows (flowers) and 4 columns (features)
# (150,) -> 150 answers, one for each flower
```

```
print("\nFirst 5 rows of features (X):")
print(X[:5])
print("\nFirst 5 labels (y):")
print(y[:5])
# Note: 0 = 'Setosa', 1 = 'Versicolor', 2 = 'Virginica'
```

```
Shape of features (X): (150, 4)
Shape of labels (y): (150,)

First 5 rows of features (X):
[[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]]

First 5 labels (y):
[0 0 0 0 0]
```

Output explanation:

1. Shape of features (X): (150, 4)

- This tells you the **size** of your main data table (X).
- **150 rows:** This means you have data for **150 different flowers**.
- **4 columns:** This means for *each* flower, you have **4 measurements** (the "features"):
    1. Sepal Length
    2. Sepal Width
    3. Petal Length
    4. Petal Width

2. Shape of labels (y): (150,)

- This tells you the size of your "answer" list (y).
- **150 rows (or items):** This means you have **150 labels** in total.

3. First 5 rows of features (X):

- This shows you the actual measurements for the first 5 flowers in your dataset.
- The first flower ([5.1 3.5 1.4 0.2]) has a:
    - Sepal length of 5.1 cm

- Sepal width of 3.5 cm
- Petal length of 1.4 cm
- Petal width of 0.2 cm

- The second flower ([4.9 3. 1.4 0.2]) has its own set of 4 measurements, and so on

4. First 5 labels (y): [0 0 0 0 0]

- This shows you the **"answer"** (the species) for each of those first 5 flowers.
- In machine learning, we use numbers as labels instead of text.
- For the Iris dataset:
  - 0 = **Setosa**
  - 1 = **Versicolor**
  - 2 = **Virginica**
- This output is telling you that the first 5 flowers in your table are all species 0, which is **Setosa**.

The Big Problem:

If we use *all* 150 flowers to *teach* our model, how do we know if it *actually learned*? It might just memorize the answers.

- **Analogy**: This is like giving a student a practice exam and then giving them the *exact same questions* for their final. They'd get 100%, but it doesn't prove they learned anything.

The Solution: Train-Test Split

- We split our dataset into two parts:
  1. Training Set (e.g., 75%): The "practice problems." The model *sees* this data (both features and labels) to learn the patterns.
  2. Testing Set (e.g., 25%): The "final exam." The model *never* sees this during training. We use it at the end to check how well it learned.

Code for Splitting:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
print(f"Training samples: {X_train.shape[0]}") # 112
print(f"Testing samples: {X_test.shape[0]}")  # 38
```

```
Training samples: 112
Testing samples: 38
```

Output Explanation:

Function Part	Meaning
X	All input feature data
y	All output (target) values
test_size=0.25	25% data for testing, 75% for training
random_state=42	Fixes the random split for reproducibility

Building & Using Our First Model :

What is a Model?

It's the "brain" we are training. It's an algorithm that finds the relationship between X (features) and y (labels).

- Our First Model: **K-Nearest Neighbors (KNN)**
  - This is one of the simplest models.
  - The Idea: To classify a *new* flower, just look at its 'k' (e.g., 3) closest neighbors from the training data. Whatever species most of its neighbors are, that's our guess!
  - Analogy: "You are who your friends are."

The 3-Step Process (Create, Train, Test):

Simple Model Creation:

```
# ---------------------------------------
# Step-by-step: Simple Machine Learning Model using KNN
# ---------------------------------------
# Import necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# 1️⃣ CREATE the model
# KNN = "K-Nearest Neighbors" → it looks at nearby data points (neighbors)
# and decides the class of a new point based on majority vote.
knn = KNeighborsClassifier(n_neighbors=3)
# k = 3 → looks at 3 nearest points

# 2️⃣ TRAIN the model
# X_train → training features (practice questions)
# y_train → training labels (correct answers)
knn.fit(X_train, y_train)

# 3️⃣ TEST the model (Predict)
# X_test → test features (exam questions)
# The model predicts answers for these, even though it never saw y_test before.
y_pred = knn.predict(X_test)

# 4️⃣ EVALUATE the model
# accuracy_score compares predicted answers (y_pred) with real answers (y_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
# 5 DISPLAY results
print("Model's predictions:", y_pred)
print("Actual answers:     ", y_test)
print(f"Model Accuracy: {accuracy * 100:.2f}%")
#accuracy_score returns a value between 0 and 1.
#Multiplying by 100 converts it to a percentage.
```

Summary & Recap

What did we just do? This 5-step pattern is the foundation for almost all basic machine learning!

1. Load Data (load_iris())
   - Identify Features (X) and Labels (y).
2. Split Data (train_test_split())
   - Create a "study" set (_train) and an "exam" set (_test).
3. Choose a Model (KNeighborsClassifier())
   - This is the "brain" we will train.
4. Train the Model (model.fit())
   - The model "studies" the training data (X_train, y_train).
5. Test the Model (model.predict() & accuracy_score())
   - The model takes the "exam" (X_test) and we check its score.

- You've just trained your first machine learning model. This same process is used for everything from spam filters to medical diagnosis.

Keep exploring — ML starts with curiosity