Co-funded by
the European Union

EcoMinds

Topic: Introduction to Machine Learning

Lesson Description:

This introductory lesson provides a basic overview of machine learning (ML) and its role in solving environmental and ecological problems. Here, we introduced the main ML tasks (classification, regression, clustering) and learning strategies (supervised and unsupervised learning). The focus is on understanding key concepts, including data preparation, feature engineering, model training, and evaluation, through clear, visual examples. The problem of underfitting and overfitting, and their consequences, are explained in depth. Environmental applications include air quality level classification and habitat detection.

Learning Outcomes:

After completing this unit, students will be able to:

1. Explain what ML is and describe its main applications in environmental science.
2. Differentiate between supervised and unsupervised learning.
3. Recognize basic ML tasks such as regression, classification, and clustering.
4. Understand the basic workflow of an ML project (data preparation, training, evaluation).
5. Recognize and explain common problems in ML model learning, such as overfitting, underfitting, and data quality issues.
6. Train and evaluate a simple ML model using environmental data.

Introduction

Machine learning (ML) enables computers to learn from data without requiring explicit programming. In recent years, ML has found wide application in solving various environmental problems, complementing or, in some cases, replacing traditional environmental modeling approaches that are primarily grounded in formal physical and chemical equations, as well as their combination with other formalisms in biology, geology, and ecology. With the increase in the volume and variety of environmental data (sensors, IoT devices, large-scale monitoring systems), the focus has shifted toward data-driven methods that learn directly from collected data.

Machine Learning is a data-driven discipline and one of the most important fields in Artificial Intelligence. Instead of prescribing a fixed model structure, ML algorithms infer underlying relationships from example thus allowing scientists and practitioners to model systems where explicit physical laws are unknown, incomplete, or too complex.

Environmental phenomena (e.g., climate processes, ecosystem degradation, and pollution dynamics) are naturally complex, nonlinear, and multidimensional problems. The field of Machine Learning offers various tools and approaches that can address such complex issues. Moreover, because rich data accompany these problems, they can be addressed more effectively with ML than traditional regression or simulation-based models often can.

Some examples of applications of ML in environmental sciences:

- Climate and weather prediction: ML models identify patterns in long-term meteorological records and satellite observations to predict temperature anomalies or precipitation extremes [1].
- Air and water quality modeling: Predictive models learn correlations between emissions, meteorological variables, and pollutant concentrations [2].
- Remote sensing and land-use classification: Image-based ML models automatically classify land-cover types from multispectral satellite data [3].
- Biodiversity and ecosystem monitoring: ML aids in species distribution modeling, acoustic classification, and ecological anomaly detection [4].

We have already mentioned that Machine Learning is a system that learns from data without the need for explicit programming for the task at hand.

> *“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.” [5]*

This formulation underlines three critical components:

- Data (Experience E): The foundation from which the model learns patterns.
- Task (T): What the model is expected to accomplish—such as classifying, predicting, or clustering.
- Performance (P): The quantitative measure of how well the model performs (accuracy, RMSE, F1-score, etc.).

Example: We want to develop an application that can detect cars in images. If we want to hardcode a set of rules (IF-THEN) that describes a car, for example:

IF an image shows 4 wheels, is rectangular, and has a windscreen, it is a CAR; otherwise, it is NOT a CAR.

This approach quickly fails because cars look different from various angles, lighting affects recognition, and some objects resemble cars. So, for this approach, we said that it is too rigid and does not produce good results.

We heard from multiple sources that a machine learning approach can solve this type of problem, given the relatively large amount of data.
So, we decided to implement that strategy: we collected a decent number of images of cars in different situations, and an equal number of images that do not contain cars. For that task, we read that a particular ML model for image recognition is recommended. All images are defined with appropriate and correct labels (CAR/NO CAR). Here, the ML model's task is to predict the correct label from an image. That task is learned from images (data), or we can say that the model learns from experience (by presenting images). At the beginning, the model will fail in many cases, but over time (showing more data - experience), it will learn to predict labels correctly in most cases. That percentage of correctly predicted image labels is the performance measure that indicates how well the model learned the task.

The model is said to **learn** because its **performance** in detecting cars in images **improves** as it **gains more experience with data**!

A typical ML pipeline involves:

1. Data collection and preprocessing
2. Feature engineering and representation
3. Model selection and training
4. Evaluation and validation
5. Interpretation and deployment

Besides its reliable predictive performance in environmental science[6], ML provides the core for high-quality decision-making systems under uncertainty [7], offering interpretable insights into feature importance [8], spatial dependencies, and system sensitivity [9]. As environmental data continues to grow exponentially, ML literacy becomes an indispensable component of modern environmental data management [10].

Under the hood of ML fundamentals

Environmental science, in essence, is rich with a large amount of data from heterogeneous sources, and it is natural to exploit the usefulness of ML models in that field. ML complements conventional scientific approaches by identifying hidden patterns and inferring correlations. In this introductory lecture, our focus is not on detailed explanations of models for various ML tasks, which will be covered in subsequent lectures, but rather on fundamental principles that apply to all tasks. The main goal is conceptual understanding, not the mathematical formalism underlying an ML model's workings, which will be covered in lectures dedicated to specific ML tasks.

Supervised and Unsupervised Learning

From the definition of Machine Learning, we can see that ML models can learn to perform a task by learning from data. There are two basic types of learning: supervised and unsupervised. Supervised learning is a learning strategy in which a model is trained on data with known input values and corresponding outputs. For example, in **classifying** satellite images into eight land-cover categories: built-up areas, roads, water bodies, agricultural land, shrubland, forest, grassland, and others [11]. Category labels are obtained from domain experts, and we know in advance where the annotation is for each image in the dataset; however, the model does not. By employing an appropriate performance measure (e.g., absolute accuracy), the model can learn to predict those areas in labeled images, and, more importantly, in new, previously unseen images. Here, the name supervised refers to annotations (labels) obtained from supervisors, the domain experts who serve as ground truth for learning a task. In addition to the **classification task** mentioned, another typical supervised learning task is **regression**, in which an ML model learns from data using input values with known real-valued outputs. Predicting the concentration of Nitrate-Nitrogen, and Chloride from the set of independent variables (average daily surface flow, 24h precipitation prior to sampling time, location of sampling site, crop cover, temperature, pH, dissolved oxygen, oxidation-reduction potential, and electrical conductivity). Dependent variables are indicators of surface water quality in agricultural watersheds [12]. In the regression tasks, the supervisor is indirect, expressed through the collected independent and dependent data.

On the other hand, unsupervised learning does not have a well-defined, known output in advance for observed inputs. In fact, only inputs are known. The ML model needs to find regularities in the input data and highlight meaningful patterns with concrete meaning in the domain of interest. Here, we cannot see any influence from the supervisor that could define the ground truth; for that reason, this type of learning in the ML field is known as unsupervised learning.

A typical task in the unsupervised learning paradigm is **clustering**, in which the goal is to group similar data instances into the same group and ensure that every dissimilar instance belongs to a different group (this is so called **crisp clustering** approach), while in cases when the same instance can belong to more than one group (cluster) with associated membership value we are talking on **soft clustering** approaches. In environmental science, clustering plays an important role and can be applied to assess groundwater quality [13].

There are many ML models that could be applied for the mentioned tasks:

- Classification (SVM – Support Vector Machines, Decision Trees, RIPPER, K-NN, Artificial Neural Networks, Logistic Regression, Random Forest, Naive Bayes, AdaBoost, Bagging, Stacking, etc.)
- Regression (Linear regression, SVM for regression, Decision trees for regression, Multilayer Perceptron, etc.)
- Clustering (K-means clustering, Expectation-maximization, DBSCAN, HDBSCAN, Cobweb, Hierarchical clustering, Fuzzy C-means, etc.)

Most of these ML models you will learn in more depth in the next lectures, so here we will provide just basic insight into a model needed for our further exposition of basic concepts in ML, such as data preparation, feature engineering, training, testing, performance evaluation, and overfitting/underfitting phenomena. Additionally, a brief introduction to Google Colab will be presented.

Logistic Regression – Fundamental Insight

Although it contains the word *regression*, **logistic regression is a classification model**, most used for **binary classification problems**. This type of classification, as the name suggests, models a two-class problem, typically denoted by 0 and 1.

Its goal is not to predict a continuous value, but to estimate the **probability that an example belongs to a given class** (usually class 1).

Formally, it models:

$P(y=1;x,\theta)$ Logically, when we know this probability, the other is also known:

$$P(y=0;x,\theta)=1-P(y=1;x,\theta)$$

Where is:

- **x** is a feature vector
- y (0 or 1) is the class label
- θ are the model parameters (weights)

At the core of logistic regression is the **sigmoid (logistic) function**, which maps any real-valued input to the interval **(0, 1)**:

$$h(x)=\frac{1}{1+e^{-\theta^T \cdot x}}$$

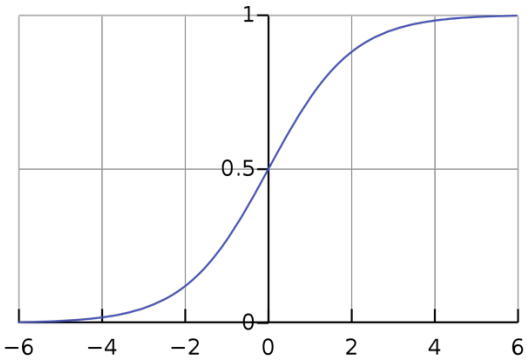


Figure 1 Sigmoidal (logistic) function (Source: https://commons.wikimedia.org/wiki/File:Logistic-curve.svg)

This is crucial because:

- Probabilities must lie between 0 and 1,
- The sigmoid smoothly converts a linear combination of features ($\theta^T \cdot x$) into a probability.

Interpretation is now strongly related to probabilities, not just labels. The logistic regression produces a numerical output:

- $P(y=1 \mid x)$ — probability of the positive class
- $P(y=0 \mid x)=1-P(y=1 \mid x)$ — probability of the negative class

This makes logistic regression especially useful when:

- classes are imbalanced,
- decision thresholds need to be adjusted,
- ranking or confidence estimation matters

Two major question arises:

- What is the principle to determine the weights (θ_i)?
- How does the decision boundary of this model look?

$$h(x)=¿\ \frac{1}{1+e^{-(\theta_0+\theta_1 \cdot x_1+\theta_2 \cdot x_2)}}$$

$$h(x)=¿\ \frac{1}{1+e^{-(\theta_0+\theta_1 \cdot x_1^2+\theta_2 \cdot x_2^2)}}$$

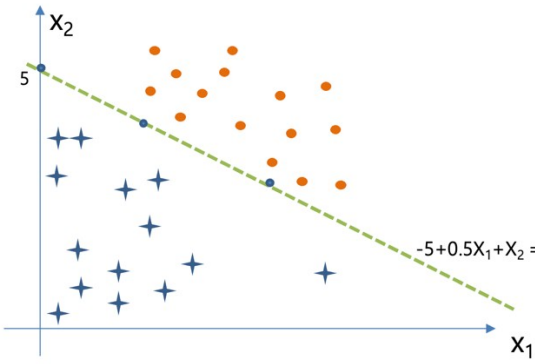


Figure 2 Linear decision boundary

Decission → all above line is assigned to class 1, otherwise to class 0

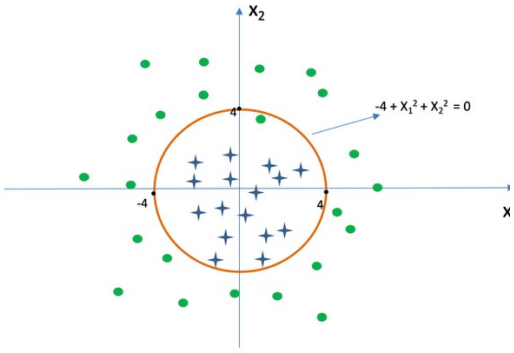


Figure 3 Nonlinear decision boundary

Decission → all out of circle is assigned to class 1, otherwise to class 0

The first case depicted in Figure 2 shows logistic regression that yields a linear decision boundary in the original feature space. In the second case, a nonlinear decision boundary is obtained (Figure 3), but only after applying feature engineering that transforms the original input features (e.g., by adding squared terms), while the model remains linear in parameters. The weights θ are learned from input data by optimizing a chosen performance measure.

In this lecture, there is no emphasis on performance metrics or a detailed explanation, as these will be covered in subsequent lectures.

Accuracy

Accuracy measures the proportion of correct predictions among all predictions.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy answers the question: “How often is the model correct overall?”

Precision

Precision measures how many of the instances predicted as positive are actually positive.

$$P = \frac{TP}{TP + FP}$$

Precision answers the question: “When the model predicts a positive case, how often is it correct?”

Recall (Sensitivity)

Recall measures how many of the actual positive instances are correctly detected by the model.

$$R = \frac{TP}{TP + FN}$$

High recall is crucial when missing an environmental event (such as pollution exceedance or habitat presence) carries significant risk.

F1-score

The F1-score combines precision and recall into a single measure using their harmonic mean.

$$F1 = \frac{2 \cdot P \cdot R}{P + R}$$

The F1-score is useful when both false positives and false negatives are important and when the dataset is imbalanced, which is common in environmental applications.

ROC Curve

The Receiver Operating Characteristic (ROC) curve plots the True Positive Rate (Recall) against the False Positive Rate for all possible classification thresholds.

$$FPR = \frac{FP}{FP + TN}$$

The ROC curve illustrates the trade-off between the detection of true environmental events and the incidence of false alarms.

ROC-AUC

ROC-AUC (Area Under the ROC Curve) summarizes the ROC curve into a single value between 0 and 1.

Interpretation:

- 0.5 indicates random guessing
- 1.0 indicates perfect discrimination

ROC-AUC reflects how well the model can rank positive cases higher than negative ones, independently of a specific decision threshold.

Precision–Recall (PR) Curve

The Precision–Recall (PR) curve plots precision against recall for different classification thresholds.

The PR curve directly shows how precision changes as recall increases, highlighting the trade-off between false alarms and missed detections.

PR-AUC (Average Precision)

PR-AUC summarizes the Precision–Recall curve into a single value. PR-AUC is particularly informative for environmental datasets where positive cases are rare, such as pollution exceedances or specific habitat types. It is more sensitive to false positives than ROC-AUC.

Key Takeaway

No single metric is universally best. In environmental machine learning, metric choice should reflect the application's priorities:

- Use recall when missing true events is risky
- Use precision when false alarms are costly
- Use F1-score to balance both
- Prefer PR-AUC and PR curves for rare-event detection
- Use ROC-AUC for general model comparison

Google Colab

Google Colab is a cloud-based platform that provides free, browser-based access to Google's computing power, including CPUs, GPUs, and TPUs. Built on the Jupyter Notebook framework, it allows users to write and execute Python code without any local setup. While it comes preloaded with essential machine learning libraries and integrates seamlessly with Google Drive, it requires a Google account and operates entirely within the Google ecosystem.

The Google Colab service is one of the simplest ways for students to run Python scripts without creating a working environment with all the necessary libraries. Or, more simply, it allows you to bring an environment for running even complex Python code that meets all requirements and dependencies in a user-friendly way. The environment includes a basic Python interpreter and additional libraries and modules (prebuilt code developed and tested by reputable sources to simplify and accelerate development and improve reliability). These libraries support tasks such as efficient numerical computation

(NumPy), robust data structures and analysis (Pandas), visualization (Matplotlib), and state-of-the-art machine learning implementations (Scikit-learn, TensorFlow, PyTorch).

To use Colab, you need to have a Google account. If you don't have one, you can create one here. Once you have an account, you can access Colab here. You should see a screen like this:

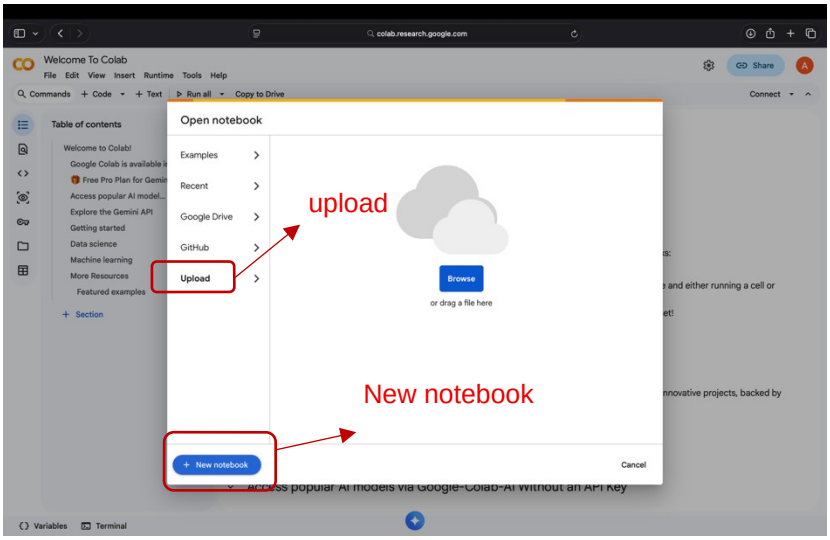


Figure 4 Google Colab start window

From this window, you can open a new Notebook, upload a prepared one, or even open an existing Notebook. The same could be done from the File menu. For now, open a new notebook.

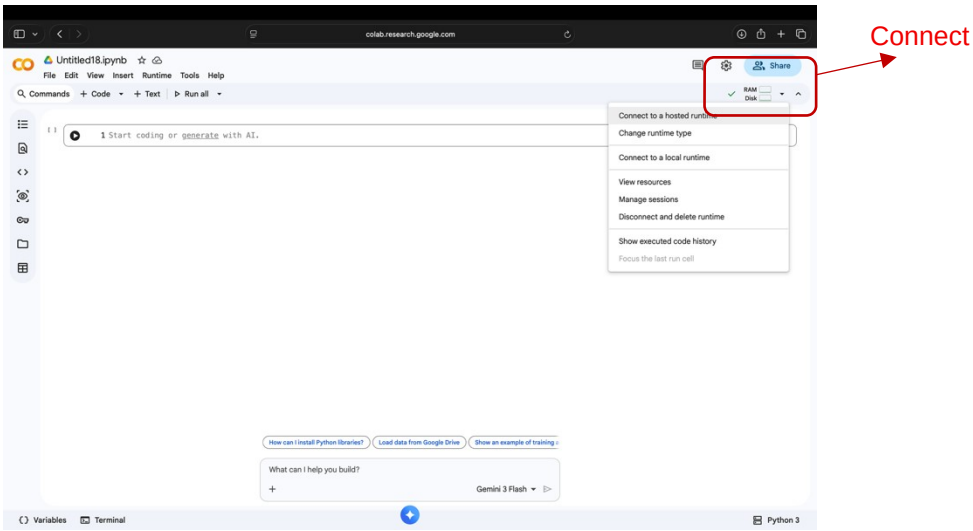


Figure 5 Running a new Google Colab notebook

In Figure 5, in the right upper corner, click on the arrow and choose Connect to a hosted runtime – after the connection is established, the indicator will show you RAM, CPU, and a green checkmark. Two types of cells are available – code cells and text cells (markdown

cells). To run Python code, code cells should be used. To execute a cell, you can either click on the ‘play’ button to the left of the cell or press ‘Shift+Enter’ on your keyboard, which will automatically run the selected cell and insert a new one. Working with text cells is simple – just insert a text cell and double-click to enter edit mode and start writing. On the right side, the preview will be visible; after clicking outside that cell, the result will appear. Look at the markdown tutorial to learn some basics for formatting text in those cells. Renaming a notebook is simple (Figure 6) by changing the name in the upper left corner (where the extension is ipynb).

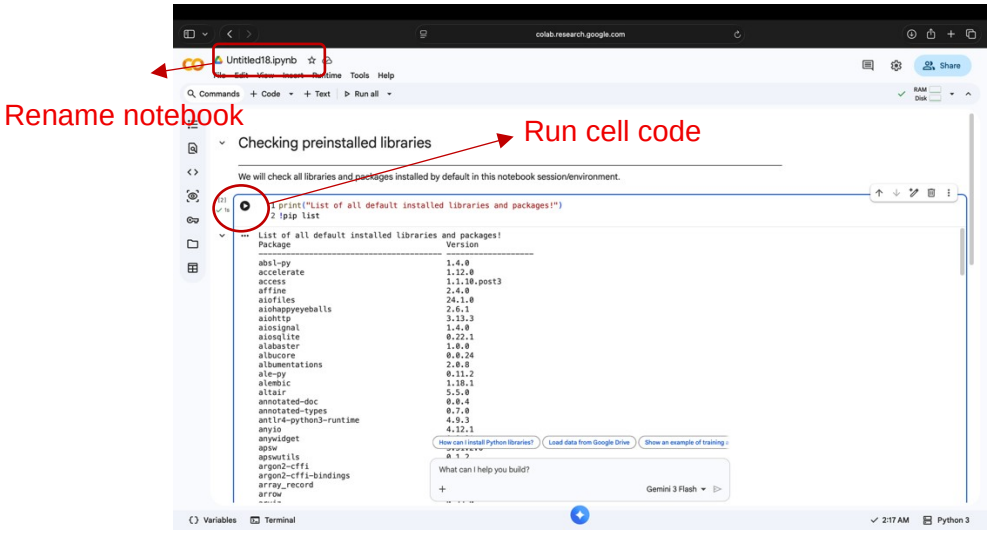


Figure 6 Running Python code in Colab / Renaming Notebook

In addition to saving the Colab notebook to Google Drive or a GitHub repository, it is possible to share the notebook (in the upper-right corner, select Share) or download it as a .py or .ipynb file (from the File menu, choose Download option - Figure 7.

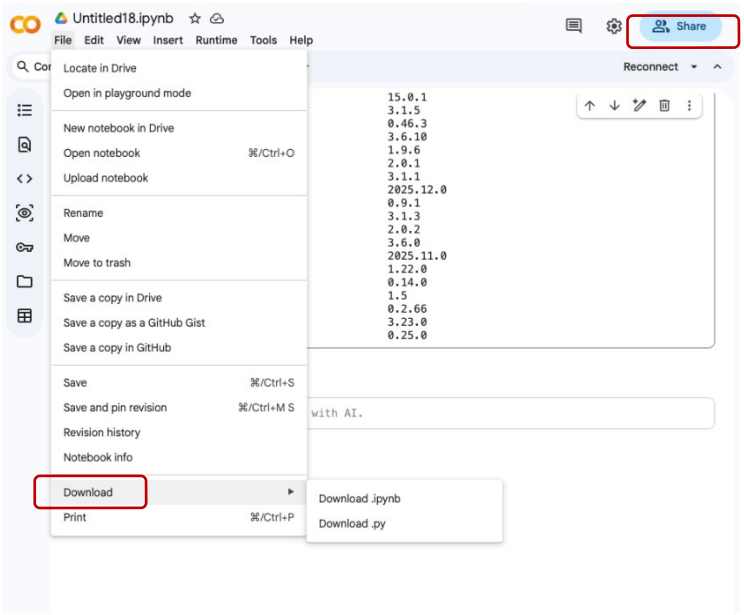


Figure 7 Download notebook / Share notebook

Predicting NO_2 Exceedance using Air Quality Sensor Data

Air quality monitoring plays a key role in protecting public health and supporting environmental policy. Modern monitoring stations continuously collect large amounts of data from chemical sensors and meteorological instruments. This makes air quality an excellent domain for demonstrating how machine learning can support real-world environmental decision-making.

In practice, environmental decisions are rarely based on predicting an exact pollution value. Instead, the key questions are usually very simple:

- Is a legal air-quality limit exceeded?
- Should an alert be issued?
- Do mitigation measures need to be activated?

This use case focuses on answering questions of this type by predicting **NO_2 exceedance events** using machine learning.

DATASET for Case Study

In this lecture, we use the Air Quality dataset from the **UCI Machine Learning Repository**, a well-known public repository of machine learning datasets.

- **Time period:** March 2004 to February 2005
- **Sampling frequency:** One measurement per hour
- **Location:** Urban monitoring station in an Italian city

- **Number of observations:** Approximately 9,300 hourly records

Each record represents the environmental conditions measured during a single hour.

The dataset combines three types of measurements:

Ground-truth pollutant concentrations

These values come from reference analyzers and represent reliable chemical measurements:

- Nitrogen Dioxide – NO_2(GT) (µg/m³)
- Carbon Monoxide – CO(GT) (mg/m³)
- Nitrogen Oxides – NOx(GT) (ppb)
- Benzene – C_6H_6(GT) (µg/m³)

In the description of the original dataset, these variables are intended as targets for regression models.

Sensor signals

The dataset also contains readings from metal oxide sensors. These sensors are cheaper and noisier than reference analyzers, and their signals are often nonlinear and cross-sensitive. In this use case, these sensor readings form the **input features** for our models.

Meteorological variables

Environmental conditions that affect both pollution levels and sensor behavior are included:

- Temperature
- Relative humidity
- Absolute humidity

Comments on the time-series nature of the data

All measurements are time-stamped and collected sequentially. This means that neighboring observations are related in time and not completely independent.

For this lecture, however, each measurement is treated as a **snapshot of the environment at a given hour**. To avoid methodological issues, we split the data **chronologically** for training and testing, mirroring how such systems would be used in practice.

Remarks to the original problem definition

According to the UCI repository, this dataset is designed for **regression**. The goal of the original problem was to predict continuous pollutant concentrations (such as NO_2) from sensor signals and meteorological data.

This formulation makes sense when the objective is precise numerical prediction.

Reformulating the problem for classification

In environmental monitoring, exact pollutant values are often less important than whether a critical threshold is exceeded. Regulatory systems and public warnings are usually triggered when pollution crosses predefined limits.

For this reason, we reformulate the original regression problem as a **binary classification task**. Instead of predicting the exact NO_2 concentration, we predict whether an **exceedance event** occurs.

Regulatory threshold used

The threshold used in this case study comes from official European air-quality legislation:

Directive 2008/50/EC of the European Parliament and of the Council of 21 May 2008 on ambient air quality and cleaner air for Europe. From this directive we have:

- The **hourly NO_2 limit value** is **200 µg/m³**
- This limit should not be exceeded more than 18 times per year.

The official legislative text (EUR-Lex) for this direction list mentions limits in ANNEX XI (Limit Values for the protection of Human health) under section B.

Using this threshold ensures that our classification task is **legally grounded and environmentally meaningful**.

Definition of the two classes

Based on the EU regulation, we define the target variable as follows:

- **Class 0 – acceptable air quality:**
  NO_2 concentration below 200 µg/m³
- **Class 1 – exceedance event:**
  NO_2 concentration equal to or above 200 µg/m³

This creates two clear and interpretable classes.

Why is logistic regression suitable for this introductory lecture

With this reformulation, logistic regression is used to estimate the probability that an NO_2 exceedance occurs given the available sensor and meteorological data.

Logistic regression is well-suited for this task because:

- It provides **probability estimates**, not just class labels,
- Its classification boundaries are easy to interpret,
- It allows a clear demonstration of **feature engineering**,

- It makes overfitting and underfitting easy to observe,
- It supports intuitive visualization of model behavior.

Time-series dataset nature in the context of the problem reformulation

Although the data is collected over time, this does not cause a problem for the presented use case.

In this lecture:

- We do not model temporal dynamics explicitly,
- each observation is treated as an independent snapshot,
- A chronological train–test split is used to avoid information leakage.

Time-series modeling techniques are deliberately excluded from this unit to preserve the focus on fundamental machine-learning concepts.

Thus, a tailored use case allows us to explore a wide range of important machine-learning concepts within a realistic environmental context:

- transforming a regression dataset into a classification task using regular directives from the legislative bodies,
- engineering features based on environmental knowledge,
- understanding linear and nonlinear decision boundaries,
- separating training and testing data correctly,
- recognizing overfitting and underfitting,
- using visualization to interpret and analyze models.

As a result, the use case provides a clear and practical connection between **machine learning theory** and **real-world environmental applications [14]**.

Regarding the directive limits, we focus on the per-hour exceedance classification. First, we will load data using the ucimlrepo package. Other libraries and packages needed in this exposition are scikit-learn [15], pandas, numpy, and matplotlib.

[1] ✓ 8 s

```
# If running in Google Colab, install dependencies (safe to run even if already installed)
!pip -q install ucimlrepo scikit-learn pandas numpy matplotlib
```

[2] ✓ 5 s

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, ConfusionMatrixDisplay, roc_auc_score, RocCurveDisplay
)

# For reproducibility
RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)


from sklearn.dummy import DummyClassifier
from sklearn.metrics import (
    precision_recall_curve, average_precision_score, PrecisionRecallDisplay
)

```

The model is a LogisticRegression from scikit-learn, and the metrics are standard measures for evaluating model performance. A random state ensures reproducibility of results. DummycCassifier is a simple classifier that assigns all instances to the majority class (the most frequent class) – we need this to show problems with some standard metrics in the so-called imbalanced classification problems (this use case belongs to that type). Data loading is performed using ucimlrepo without requiring direct downloads.

[3] ✓ 1 s

```
from ucimlrepo import fetch_ucirepo

# UCI ML Repo dataset: Air Quality
# Note: In ucimlrepo, dataset IDs can change or multiple IDs can exist.
# We attempt fetch by name; if it fails, we'll guide a fallback.
try:
    air_quality = fetch_ucirepo(name="Air Quality")
    X_raw = air_quality.data.features
    y_raw = air_quality.data.targets  # may be empty for some UCI loaders
    meta = air_quality.metadata
    print("Loaded from UCI via ucimlrepo.")
    print("Features shape:", X_raw.shape)
    print("Targets shape:", None if y_raw is None else y_raw.shape)
    display(meta)
except Exception as e:
    print("Could not fetch via ucimlrepo:", repr(e))
    print("\nFallback option: download the dataset from UCI and upload to Colab, then read it with pandas.")
    raise

```

Cleaning missing values and parsing timestamps are part of the data preparation procedure. In that sense, we will:

- Combine **Date** and **Time** into a timestamp

- replace **-200** with NaN
- drop rows where the ground-truth NO_2 is missing (we need it for labels)
- (optionally) drop columns with too many missing values

[4] ✓ 0 s

```python
df = X_raw.copy()

# In many versions of the dataset, Date and Time columns exist like 'Date' and 'Time'
# We'll try to detect them robustly.
date_col_candidates = [c for c in df.columns if c.lower() in ("date", "data")]
time_col_candidates = [c for c in df.columns if c.lower() in ("time", "ora")]

print("Date candidates:", date_col_candidates)
print("Time candidates:", time_col_candidates)

if not date_col_candidates or not time_col_candidates:
    raise ValueError("Could not find Date/Time columns. Inspect df.columns and adjust parsing.")

date_col = date_col_candidates[0]
time_col = time_col_candidates[0]

# Create a timestamp (common format is dd/mm/yyyy and hh.mm.ss or hh.mm)
# We'll try flexible parsing.
df["timestamp"] = pd.to_datetime(df[date_col].astype(str) + " " + df[time_col].astype(str),
                                 errors="coerce", dayfirst=True)

# Replace -200 with NaN in numeric columns
for c in df.columns:
    if c == "timestamp":
        continue
    # Attempt numeric conversion if possible
    df[c] = pd.to_numeric(df[c], errors="ignore")

num_cols = df.select_dtypes(include=[np.number]).columns
df[num_cols] = df[num_cols].replace(-200, np.nan)

# Sort by time
df = df.sort_values("timestamp").reset_index(drop=True)

# Identify the NO2 ground-truth column (commonly 'NO2(GT)')
no2_candidates = [c for c in df.columns if "no2" in c.lower() and "gt" in c.lower()]
print("NO2(GT) candidates:", no2_candidates)
if not no2_candidates:
    raise ValueError("Could not find NO2(GT) column. Inspect df.columns to locate it.")
NO2_COL = no2_candidates[0]

# Drop rows without timestamp or without NO2 ground truth
df = df.dropna(subset=["timestamp", NO2_COL]).reset_index(drop=True)

print("After cleaning:")
print(df.shape)
df.head()

```

Data frame after cleaning:

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	AH	timestamp
0	4/1/2004	0:00:00	1.6	1143.0	106.0	6.3	825.0	96.0	986.0	86.0	1477.0	978.0	12.0	61.6	0.8593	2004-01-04 00:00:00
1	4/1/2004	1:00:00	1.2	1044.0	100.0	5.1	770.0	85.0	1031.0	70.0	1425.0	944.0	11.5	63.9	0.8652	2004-01-04 01:00:00
2	4/1/2004	2:00:00	1.1	1034.0	71.0	4.1	716.0	50.0	1085.0	55.0	1405.0	891.0	10.7	67.2	0.8630	2004-01-04 02:00:00
3	4/1/2004	4:00:00	0.7	909.0	44.0	2.4	615.0	57.0	1237.0	49.0	1322.0	790.0	10.2	66.6	0.8299	2004-01-04 04:00:00
4	4/1/2004	5:00:00	0.9	996.0	45.0	2.9	648.0	64.0	1176.0	50.0	1340.0	852.0	11.0	63.7	0.8325	2004-01-04 05:00:00

To reformulate the original regression problem into a classification problem, we will use EU exceedance levels from the mentioned directive. We create the labels:

- **y = 1** if $NO_2 \geq 200$ µg/m³
- **y = 0** otherwise

This corresponds to a **real policy threshold**, so the classes have real-world meaning.

[6] ✓ 0 s

```
df["y_exceed"] = (df[NO2_COL] >= EU_NO2_THRESHOLD).astype(int)

pos_rate = df["y_exceed"].mean()
print(f"Positive class rate (NO₂ exceedance): {pos_rate:.4f} ({pos_rate*100:.2f}%)")
df[["timestamp", NO2_COL, "y_exceed"]].head()

```

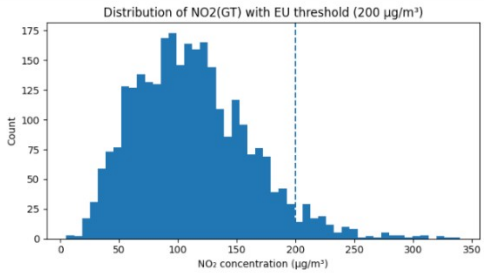
```
Positive class rate (NO₂ exceedance): 0.0478 (4.78%)
```

	timestamp	NO2(GT)	y_exceed
0	2004-01-04 00:00:00	86.0	0
1	2004-01-04 01:00:00	70.0	0
2	2004-01-04 02:00:00	55.0	0
3	2004-01-04 04:00:00	49.0	0
4	2004-01-04 05:00:00	50.0	0

From the above data, it is evident that only 4.78% of the total data points belong to class 1 (exceedance level), which is a typical example of an imbalanced classification. From the visualization of NO_2 distribution against the EU threshold, we can see the imbalance more clearly.

[22] ✓ 0 s

```
# NO2 distribution with EU threshold line
EU_NO2_THRESHOLD = 200.0  # µg/m³

plt.figure(figsize=(7,4))
plt.hist(df[NO2_COL].dropna(), bins=50)
plt.axvline(EU_NO2_THRESHOLD, linestyle="--")
plt.title(f"Distribution of {NO2_COL} with EU threshold (200 µg/m³)")
plt.xlabel("NO₂ concentration (µg/m³)")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# Class balance preview (will define label in next section)

```



We are proceeding with data preparation (median imputation) and feature engineering (feature selection) by dropping the non-feature columns Date, Time, and timestamps, keeping only numeric sensor and meteorological variables, and finally dropping sparse columns.

[8] ✓ 0s

```
# Drop non-features
drop_cols = [date_col, time_col, "timestamp", NO2_COL, "y_exceed"]
feature_df = df.drop(columns=[c for c in drop_cols if c in df.columns], errors="ignore")

# Keep numeric features only
X_all = feature_df.select_dtypes(include=[np.number]).copy()
y_all = df["y_exceed"].copy()

# Optionally drop columns with too much missing data (e.g., > 40% missing)
too_missing = X_all.columns[X_all.isna().mean() > 0.40]
print("Dropping columns with >40% missing:", list(too_missing))
X_all = X_all.drop(columns=too_missing)

# Median imputation
X_all = X_all.fillna(X_all.median(numeric_only=True))

print("Final feature matrix:", X_all.shape)
X_all.head()

```

```
Dropping columns with >40% missing: ['NMHC(GT)']
Final feature matrix: (2910, 11)
```

	CO(GT)	PT08.S1(CO)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	AH
0	1.6	1143.0	6.3	825.0	96.0	986.0	1477.0	978.0	12.0	61.6	0.8593
1	1.2	1044.0	5.1	770.0	85.0	1031.0	1425.0	944.0	11.5	63.9	0.8652
2	1.1	1034.0	4.1	716.0	50.0	1085.0	1405.0	891.0	10.7	67.2	0.8630
3	0.7	909.0	2.4	615.0	57.0	1237.0	1322.0	790.0	10.2	66.6	0.8299
4	0.9	996.0	2.9	648.0	64.0	1176.0	1340.0	852.0	11.0	63.7	0.8325

Now, we can show why absolute accuracy can be misleading in high-imbalanced datasets. With the EU threshold (NO_2 ≥ 200 µg/m³), the positive class is typically rare (an event detection problem). That means accuracy alone can be misleading: a model that always predicts “no exceedance” can achieve high accuracy while being useless. We compute the majority-class baseline (using DummyClassifier) and introduce precision/recall and PR-AUC (Average Precision) as more informative metrics for rare events. The output obtained from this illustrative example is *exactly* what we want for the “accuracy trap” demo — it’s a perfect illustration of imbalanced classification.

- Accuracy ≈ 0.952
  → About 95.2% of hours are “no exceedance” in our split.
  The dummy model predicts *only* the majority class (0), so it looks “good” in terms of accuracy.
- Precision = 0.0, Recall = 0.0, F1 = 0.0 → The model never predicts an exceedance (class 1).
  So:
  - It catches 0 exceedances → Recall = 0
  - It makes 0 positive predictions → Precision is defined as 0 (with zero_division = 0)

- F1 is therefore 0

- PR-AUC ≈ 0.0481
  This is the most informative number here. For such a dummy model (always predicting the same thing), PR-AUC tends to be close to the positive class rate. So our exceedance rate is likely around 4.8% (at least in that evaluation split). This is consistent with the 95% accuracy.

[12] ✓ 0s

```python
# DummyClassifier baseline (accuracy trap demonstration)
from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, average_precision_score

X_tmp = X_all.copy()
y_tmp = y_all.copy()

# Random split ONLY to demonstrate the accuracy trap
# (Proper chronologica Ask is used later for real evaluation)
Xtr, Xte, ytr, yte = t st_split(
    X_tmp, y_tmp,
    test_size=0.3,
    random_state=RANDOM_STATE,
    stratify=y_tmp
)

dummy = DummyClassifier(strategy="most_frequent")
dummy.fit(Xtr, ytr)

y_pred = dummy.predict(Xte)
y_proba = dummy.predict_proba(Xte)[:, 1]

print("DummyClassifier (majority class baseline)")
print("Accuracy :", accuracy_score(yte, y_pred))
print("Precision:", precision_score(yte, y_pred, zero_division=0))
print("Recall   :", recall_score(yte, y_pred, zero_division=0))
print("F1       :", f1_score(yte, y_pred, zero_division=0))
print("PR-AUC   :", average_precision_score(yte, y_proba))
```

```
DummyClassifier (majority class baseline)
Accuracy : 0.9518900343642611
Precision: 0.0
Recall   : 0.0
F1       : 0.0
PR-AUC   : 0.048109965635738834
```

Our Logistic regression model needs to learn from data, but we also demand that it generalizes well to unseen data from the same problem domain. For that reason, the dataset is regularly split into a training/test dataset. Since the original dataset is a time series dataset, to avoid data leakage, we will use the first 70% of the data as the training set and the last 30% as the test set (chronological split). This simulates a realistic situation: **train on the past, evaluate on the future**. Note on imbalance: After thresholding (EU NO_2 limit), exceedance events can be rare. We therefore rely on **precision/recall**, **PR-AUC**, and confusion matrices—not accuracy alone. Overall, we want the model to perform well on both the training and test datasets. To visualize decision boundaries, we select **two features** (any two informative features can be used).

We then train **logistic regression** with standardization and plot:

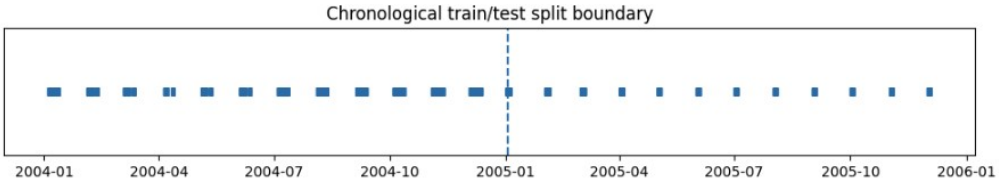
- the scatter of points (colored by class)
- the **linear** decision boundary

- probability contours (optional)

[13] ✓ 0s

```
# Chronological split index
split_idx = int(len(df) * 0.70)

X_train = X_all.iloc[:split_idx].copy()
y_train = y_all.iloc[:split_idx].copy()
X_test  = X_all.iloc[split_idx:].copy()
y_test  = y_all.iloc[split_idx:].copy()

print("Train size:", X_train.shape, "Test size:", X_test.shape)
print("Train positive rate:", y_train.mean(), "Test positive rate:", y_test.mean())

# Visualize split on timeline
Ask ure(figsize=(10,2))
plot(df["timestamp"], np.zeros(len(df)), marker="|", linestyle="None")
plt.axvline(df["timestamp"].iloc[split_idx], linestyle="--")
plt.title("Chronological train/test split boundary")
plt.yticks([])
plt.tight_layout()
plt.show()

```

```
Train size: (2036, 11) Test size: (874, 11)
Train positive rate: 0.007858546168958742 Test positive rate: 0.14073226544622425
```



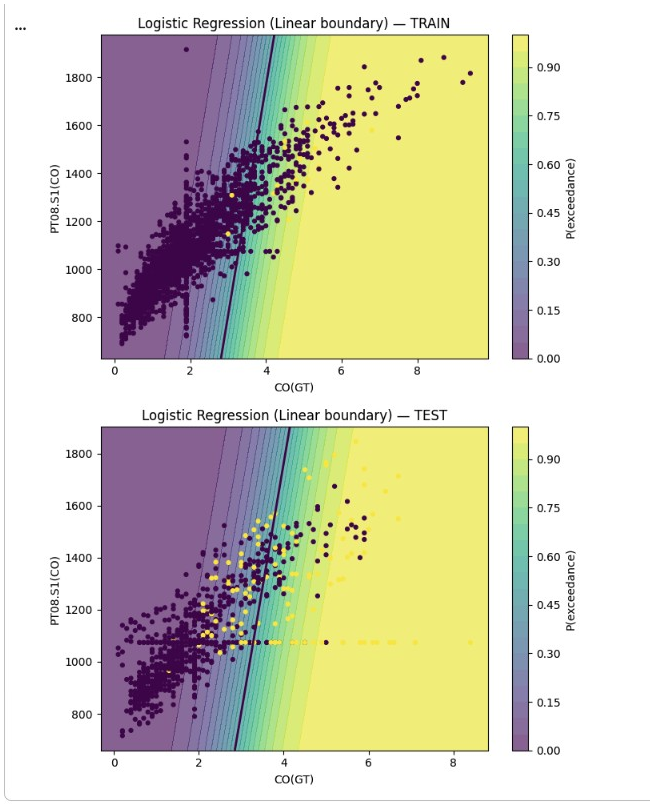
[14] ✓ 0s

```
# Pick two features to visualize
# We'll choose two columns with decent variance
candidate_cols = X_all.columns.tolist()
if len(candidate_cols) < 2:
    raise ValueError("Not enough numeric features to visualize.")

feat1, feat2 = candidate_cols[0], candidate_cols[1]
print("Using features:", feat1, "and", feat2)

X2_train = X_train[[feat1, feat2]].copy()
X2_test  = X_test[[feat1, feat2]].copy()

lin_model = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=2000, class_weight="balanced", random_state=RANDOM_STATE))
])
lin_model.fit(X2_train, y_train)

print("Train accuracy:", accuracy_score(y_train, lin_model.predict(X2_train)))
print("Test accuracy :", accuracy_score(y_test,  lin_model.predict(X2_test)))

# More informative metrics on TEST (imbalanced setting)
y_pred_lin = lin_model.predict(X2_test)
y_proba_lin = lin_model.predict_proba(X2_test)[:,1]

print("TEST Precision:", precision_score(y_test, y_pred_lin, zero_division=0))
print("TEST Recall   :", recall_score(y_test, y_pred_lin, zero_division=0))
print("TEST F1       :", f1_score(y_test, y_pred_lin, zero_division=0))
if len(np.unique(y_test)) == 2:
    print("TEST PR-AUC (AP):", average_precision_score(y_test, y_proba_lin))

```

```
Using features: CO(GT) and PT08.S1(CO)
Train accuracy: 0.8742632612966601
Test accuracy : 0.8649885583524027
TEST Precision: 0.5177304964539007
TEST Recall   : 0.5934959349593496
TEST F1       : 0.553030303030303
TEST PR-AUC (AP): 0.593996588340624
```

[15] ✓ 0 s

```python
def plot_decision_boundary_2d(model, X, y, title, grid_steps=200):
    # Create grid
    x_min, x_max = X.iloc[:,0].min(), X.iloc[:,0].max()
    y_min, y_max = X.iloc[:,1].min(), X.iloc[:,1].max()
    pad_x = 0.05 * (x_max - x_min + 1e-9)
    pad_y = 0.05 * (y_max - y_min + 1e-9)
    x_min, x_max = x_min - pad_x, x_max + pad_x
    y_min, y_max = y_min - pad_y, y_max + pad_y

    xx, yy = np.meshgrid(np.linspace(x_min, x_max, grid_steps),
                         np.linspace(y_min, y_max, grid_steps))
    grid = pd.DataFrame({X.columns[0]: xx.ravel(), X.columns[1]: yy.ravel()})

    # Predict probabilities for class 1
    probs = model.predict_proba(grid)[:,1].reshape(xx.shape)

    plt.figure(figsize=(7,5))
    # Probability contour
    cs = plt.contourf(xx, yy, probs, levels=20, alpha=0.6)
    plt.colorbar(cs, label="P(exceedance)")

    # Scatter points
    plt.scatter(X.iloc[:,0], X.iloc[:,1], c=y, s=12)
    # Decision boundary at p=0.5
    plt.contour(xx, yy, probs, levels=[0.5], linewidths=2)

    plt.xlabel(X.columns[0])
    plt.ylabel(X.columns[1])
    plt.title(title)
    plt.tight_layout()
    plt.show()

plot_decision_boundary_2d(lin_model, X2_train, y_train,
                          "Logistic Regression (Linear boundary) — TRAIN")
plot_decision_boundary_2d(lin_model, X2_test, y_test,
                          "Logistic Regression (Linear boundary) — TEST")
```



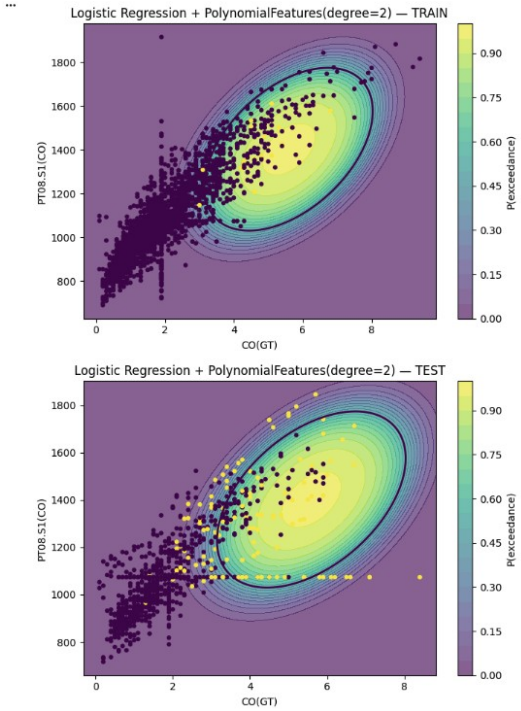
The linear decision boundary visualization shows how logistic regression separates the two classes with a straight line at the probability threshold (p = 0.5), revealing both the class separation and the model's confidence in different regions of the feature space.

Logistic regression is **linear in its parameters**, but by transforming the input features (e.g., adding squared and interaction terms), we can get a **nonlinear decision boundary in the original space**. We use PolynomialFeatures (degree=2) to perform additional feature engineering.

```
[23]
✓ 4 s
```

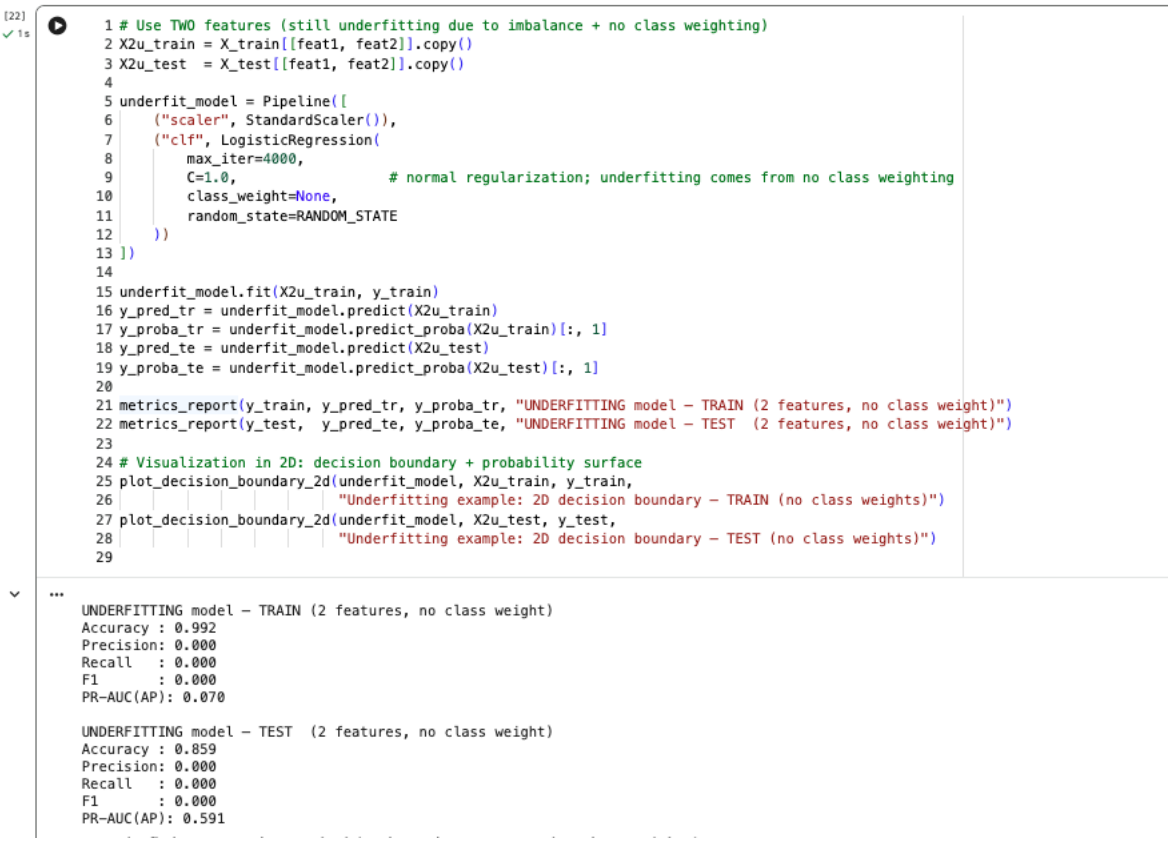
```python
poly2_model = Pipeline([
    ("scaler", StandardScaler()),
    ("poly", PolynomialFeatures(degree=2, include_bias=False)),
    ("clf", LogisticRegression(max_iter=5000, class_weight="balanced", random_state=RANDOM_STATE))
])
poly2_model.fit(X2_train, y_train)

print("Train accuracy (degree=2):", accuracy_score(y_train, poly2_model.predict(X2_train)))
print("Test accuracy  (degree=2):", accuracy_score(y_test,  poly2_model.predict(X2_test)))
# More informative metrics on TEST
y_pred_poly2 = poly2_model.predict(X2_test)
y_proba_poly2 = poly2_model.predict_proba(X2_test)[:,1]
print("TEST Precision (deg=2):", precision_score(y_test, y_pred_poly2, zero_division=0))
print("TEST Recall    (deg=2):", recall_score(y_test, y_pred_poly2, zero_division=0))
print("TEST F1        (deg=2):", f1_score(y_test, y_pred_poly2, zero_division=0))
if len(np.unique(y_test)) == 2:
    print("TEST PR-AUC (AP) (deg=2):", average_precision_score(y_test, y_proba_poly2))


plot_decision_boundary_2d(poly2_model, X2_train, y_train,
                          "Logistic Regression + PolynomialFeatures(degree=2) — TRAIN")
plot_decision_boundary_2d(poly2_model, X2_test, y_test,
                          "Logistic Regression + PolynomialFeatures(degree=2) — TEST")


```

```
Train accuracy (degree=2): 0.8487229862475442
Test accuracy  (degree=2): 0.8260869565217391
TEST Precision (deg=2): 0.4013605442176871
TEST Recall    (deg=2): 0.4796747967479675
TEST F1        (deg=2): 0.4370370703703706
TEST PR-AUC (AP) (deg=2): 0.44004674954757994
```



The nonlinear decision boundary visualization shows a curved separation produced by feature engineering, which increases model flexibility but also increases the risk of overfitting in noisy, imbalanced environmental data. Comparing model performance on both datasets, linear logistic regression outperforms the nonlinear (degree = 2) model, suggesting that a simple linear boundary is sufficient, whereas the nonlinear model becomes overly complex and begins to adapt to noise in the data.

A key concept in machine learning is model overfitting and underfitting. In the controlled experiment, we will explain these phenomena by controlling model complexity using polynomial degree (feature complexity) and regularization strength C in logistic regression. Specifically, to demonstrate underfitting, we will disable class weighting in logistic regression. By observing performance on the training and test datasets, it is possible to capture nuances in these problems.



```
[22]
✓ 1s
```

```python
# Use TWO features (still underfitting due to imbalance + no class weighting)
X2u_train = X_train[[feat1, feat2]].copy()
X2u_test  = X_test[[feat1, feat2]].copy()

underfit_model = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(
        max_iter=4000,
        C=1.0,                  # normal regularization; underfitting comes from no class weighting
        class_weight=None,
        random_state=RANDOM_STATE
    ))
])

underfit_model.fit(X2u_train, y_train)
y_pred_tr = underfit_model.predict(X2u_train)
y_proba_tr = underfit_model.predict_proba(X2u_train)[:, 1]
y_pred_te = underfit_model.predict(X2u_test)
y_proba_te = underfit_model.predict_proba(X2u_test)[:, 1]

metrics_report(y_train, y_pred_tr, y_proba_tr, "UNDERFITTING model — TRAIN (2 features, no class weight)")
metrics_report(y_test,  y_pred_te, y_proba_te, "UNDERFITTING model — TEST  (2 features, no class weight)")

# Visualization in 2D: decision boundary + probability surface
plot_decision_boundary_2d(underfit_model, X2u_train, y_train,
                          "Underfitting example: 2D decision boundary — TRAIN (no class weights)")
plot_decision_boundary_2d(underfit_model, X2u_test, y_test,
                          "Underfitting example: 2D decision boundary — TEST (no class weights)")

```

```
UNDERFITTING model — TRAIN (2 features, no class weight)
Accuracy : 0.992
Precision: 0.000
Recall   : 0.000
F1       : 0.000
PR-AUC(AP): 0.070

UNDERFITTING model — TEST  (2 features, no class weight)
Accuracy : 0.859
Precision: 0.000
Recall   : 0.000
F1       : 0.000
PR-AUC(AP): 0.591
```

This underfitting example was obtained by deliberately removing class weighting [16]. Without this correction, the logistic regression model primarily learns to predict the majority class and misses NO_2 exceedance events entirely. The high accuracy observed in the test and train datasets is due to class imbalance, whereas recall, F1 score, and PR-AUC are recorded low on both dataset indicating that the model fails to detect rare but important cases. This highlights how a model can appear successful when judged solely by accuracy, even if it does not learn a meaningful or practical decision rule.

```
[27]
✓ 1s
```

```python
overfit_degree = 7
overfit_C = 50.0

overfit_model = Pipeline([
    ("poly", PolynomialFeatures(degree=overfit_degree, include_bias=False)),
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(
        solver="lbfgs",
        max_iter=50000,
        C=overfit_C,
        class_weight="balanced",
        random_state=RANDOM_STATE
    ))
])
overfit_model.fit(X2_train, y_train)

y_pred_tr = overfit_model.predict(X2_train)
y_proba_tr = overfit_model.predict_proba(X2_train)[:,1]
y_pred_te = overfit_model.predict(X2_test)
y_proba_te = overfit_model.predict_proba(X2_test)[:,1]

metrics_report(y_train, y_pred_tr, y_proba_tr, f"OVERFITTING model — TRAIN (deg={overfit_degree}, C={overfit_C})")
metrics_report(y_test,  y_pred_te, y_proba_te, f"OVERFITTING model — TEST  (deg={overfit_degree}, C={overfit_C})")

plot_decision_boundary_2d(overfit_model, X2_train, y_train, f"Overfitting: nonlinear boundary (deg={overfit_degree}) — TRAIN")
plot_decision_boundary_2d(overfit_model, X2_test,  y_test,  f"Overfitting: nonlinear boundary (deg={overfit_degree}) — TEST")

```

```
...
OVERFITTING model — TRAIN (deg=7, C=50.0)
Accuracy : 0.854
Precision: 0.045
Recall   : 0.875
F1       : 0.086
PR-AUC(AP): 0.128

OVERFITTING model — TEST  (deg=7, C=50.0)
Accuracy : 0.838
Precision: 0.437
Recall   : 0.537
F1       : 0.482
PR-AUC(AP): 0.520
```

In the case of overfitting, the model becomes overly flexible and adapts too much to the training data. As a result, it detects most exceedance events but produces many false alarms, leading to low precision and an F1 score on the training set. When the same model is applied to new data, this behavior does not carry over: recall decreases, precision changes, and the overall performance becomes less reliable. The clear difference between training and test results indicates that the model has learned patterns that do not generalize well, a typical sign of overfitting.

Practical Exercise: Detecting Spruce/Fir Habitat using Logistic Regression

Dataset for this exercise: We use the **Forest Cover Type** dataset (often referred to as *Covertype*). It contains **cartographic + environmental variables** (elevation, slope, hydrology distances, hillshade, wilderness areas, soil types, …), and the label is the **forest cover type**.

In environmental terms, this is a typical task:

- **Predict habitat/cover classes from terrain and environmental descriptors**.

The original dataset comprises 7 cover types (a multi-class problem). To keep the exercise aligned with **binary Logistic Regression** and to make metrics/plots easier to interpret, we turn it into a binary task:

- **positive class (1):** Cover Type 1 = *Spruce/Fir*
- **negative class (0):** all other cover types

This reformulation reflects a realistic environmental scenario: detecting a specific habitat type of interest versus all other land-cover types. The dataset can be retrieved directly via scikit-learn's fetch_covtype().

Tools and Environment

- Programming language: Python (Google Colab)
- Libraries: numpy, pandas, scikit-learn, matplotlib
- Dataset access: sklearn.datasets.fetch_covtype
- Work mode: Individual student work

Feature Description Task

Students must consult the original dataset documentation and create a feature description table containing the following information:

- Feature name
- Feature type (continuous or categorical / one-hot encoded)
- Units (if applicable)
- Physical or environmental meaning

The dataset comprises 10 continuous environmental variables and 44 categorical variables (4 wilderness-area indicators and 40 soil-type indicators, encoded using one-hot encoding).

Student's assignments (Step – by – step)

1. Load data from scikit-learn using fetch_covtype()
2. The dataset is large, so subsample it (N=50000 records) using stratified sampling

3. Make imports of all libraries (same as in the lecture use case) and ensure experiment reproducibility by setting a fixed random seed
4. Convert a multiclass problem into a binary classification problem
    a. Class labeled as 1 = Spruce/Fir (positive class of interest)
    b. Class labeled as 0 = all other types (negative class)

    Threat spruce/Fir as the positive class since the main goal is to detect the habitat of interest!
5. Examine and report the class balance between the positive and negative classes
6. Explain why binary reformulation is realistic in environmental monitoring
7. Check for missing data
    a. If there are a few missing values → drop those
    b. If there are many → impute (mean or mode, depending on feature type)
    c. If none, explicitly state “no action needed.”
8. Check duplicates (optional), and check basic outliers (at least for continuous variables)
9. Create train and test datasets (70/30), keeping the same class distribution, using stratify = y, and explain the role of each split
10. Look at the first 10 feature scales and conclude what is needed for Logistic regression as the initial data preparation step → important: in what dataset that needs to be done?
    a. Use Pipeline with StandardScaler.
    b. Covertype has lots of 0/1 one-hot features; using StandardScaler(with_mean=False) is safer.
    c. Explain why scaling matters for LR (regularization depends on feature scale)
11. Train a baseline logistic regression model → consider that regularization controls complexity: smaller C = stronger regularization → simpler model (can underfit); larger C = weaker regularization → more flexible (can overfit, especially with engineered features). Logistic regression outputs probabilities: the default threshold of 0.5 is arbitrary.
12. Report evaluation metrics on the train and test datasets (what metrics did you choose and why)

13. Plot the confusion matrix on the test dataset, and explain:
    - Explain what TP, TN, FP, and FN mean in this environmental context.
    - FP → model predicts “Spruce/Fir”, but the location is some other cover type → environmentally risky → wrong habitat map → wrong management/conservation decisions
    - FN → model predicts “Other”, but the location is actually “Spruce/Fir” → environmentally costly, but less dangerous than FP → habitat under-protected/underestimated
    - This shows how ML supports environmental decision-making.
14. Compare the metrics from step 9 and interpret the trade-offs between the evaluation measures.
    a. What type of problem this is and why it is important to detect it; what about the detection of FP, and FN related to the cost of missing a habitat vs false alarm.
    b. This is a **habitat detection** problem with asymmetric costs, so changing the threshold affects FP/FN.
15. Explain precision, recall, and PR-AUC specifically for the task of Spruce/Fir habitat detection.
16. Increase model flexibility using polynomial feature expansion (degrees 1, 2, and 4) applied only to continuous features, and vary regularization strength (C = 0.1, 1.0, 50). Explain expected effects on model behavior.
    a. To avoid crashing Colab, apply polynomial features only to the **10 continuous variables**.
17. Visualize F1-score on training and test sets for all model configurations and interpret observed patterns.
18. Explain the concepts of underfitting and overfitting using results from the previous step.
19. Identify one underfitting and one overfitting configuration and justify your choice using numerical evidence.

20. Visualize decision boundaries in a 2D space using PCA for dimensionality reduction (visualization only) and describe how boundaries change with increasing model complexity.

What to submit – Exercise Report

- Format: Google Colab with markdown and code cells
- Must contain:
    a. Introduction (3–5 sentences)
        I. What was the problem? (original and reformulated)
        II. Why is this relevant for environmental sciences?
        III. Define a positive class and explain the asymmetric cost in this case.
    b. Data description
        I. Number of instances, number of features (grouped by type)
        II. Target variable and its distribution
        III. Missing data handling (write what you did)
        IV. Subsampling decision
    c. Methods
        I. Which models were trained and how (LogReg + parameters)
        II. Train/test split (give the ratio)
        III. Any scaling and on what dataset
    d. Results
        I. Present accuracy, precision, recall, and F1, ROC-AUC, and PR-AUC (in the table format)
        II. Show the confusion matrix with the explanation
        III. Say which model you would choose and why (not only because “it has higher accuracy,” but maybe because “it has higher recall on the positive class”)
    e. Discussion
        I. What are the risks of misclassification in environmental applications, for this example?
        II. What is the role of scaling?

III. What would you do next if this were used for environmental monitoring? (e.g., threshold tuning/precision-recall trade-off, feature importance/interpretability)

f. Conclusion

I. A few sentences summarizing what you learned from this exercise using real environmental data – mention generalization, metric choice, and trade-offs.

Assessment

- **Quiz** (10 questions): will be designed to assess conceptual understanding and applied reasoning gained through the lecture and practical exercise on classification techniques in environmental sciences. It will assess whether students understand supervised learning, decision boundaries (linear vs. non-linear), basic classification evaluation metrics (accuracy, precision, recall, F1-score), the role of the train/test split, the phenomena of underfitting and overfitting, and the importance of data visualization.

- **Practical Classification Task:** NO_2 Exceedance level prediction - This individual practical exercise gives students the opportunity to work hands-on with a classification problem drawn from an environmental monitoring scenario discussed in the lecture. Using an air-quality dataset, students train and evaluate a baseline linear model (Logistic Regression) to determine whether nitrogen dioxide (NO_2) levels exceed a defined safety threshold. The task is designed to assess students' ability to implement the full machine learning workflow, including data preparation, feature handling, model training, and evaluation. Special emphasis is placed on interpreting false positives and false negatives in the context of air-quality decision support, showing how classification models can assist environmental monitoring and risk assessment.

- **Practical Classification Task: Habitat Detection (Spruce/Fir) -** This individual practical exercise provides students with a hands-on opportunity to apply classification models to an environmental use case introduced in the lecture. Using a land-cover dataset with geographic and environmental variables, students build and evaluate a baseline linear classifier (Logistic Regression) to detect the presence of a specific habitat type (Spruce/Fir forest) versus all other land-cover types. The task is designed to assess students' ability to implement the full machine learning workflow, including data preparation, feature handling, model

training, and evaluation. Special emphasis is placed on interpreting false positives and false negatives in the context of environmental decision support, showing how classification models can aid habitat mapping, conservation planning, and land-use analysis.

- **Short Written Report**: The short-written report represents the final step of the practical habitat detection exercise. Its purpose is to allow students to summarize, interpret, and clearly communicate the results obtained from their classification models applied to the Spruce/Fir habitat detection problem. Through this report, students are expected to demonstrate their ability to:
  - clearly explain the steps of their classification workflow, from data preparation to model evaluation,
  - interpret classification metrics (namely, accuracy, precision, recall, F1-score, and the confusion matrix) in the context of habitat detection and environmental decision-making, and
  - reflect on the choice of model (linear versus non-linear), discussing how model complexity influences performance and what this means for real-world applications such as habitat mapping, conservation planning, and land-use analysis.

Evaluation Criteria

FOR QUIZ

- The quiz will assess factual, conceptual, and fundamental theoretical understandings of classification in ML, including:
  - The supervised learning paradigm,
  - The difference between linear and non-linear decision boundaries,
  - Role of train/validation/test split,
  - Classification metrics and when to use them
  - Examples of environmental classification problems (NO_2 exceedance and habitat detection)

Passing threshold: 50% correct answers.

FOR PRACTICAL CLASSIFICATION TASK

- Data Preparation and Preprocessing
  - Correct loading and inspection of the dataset (shape, target, missing values).
  - Appropriate handling of missing data (dropping or imputing) and, where needed, feature scaling/normalization.
  - Transparent and reproducible preprocessing workflow (code is clean, steps are clear).

- Model Implementation
  - Accurate setup and training of at least two classification models:
    - one baseline (e.g. Logistic Regression),
    - another nonlinear
  - Appropriate use of train/test split (and validation if used).
  - The code is organized, and parameters are explicitly stated.

- Model Evaluation and Diagnostics
  - Correct calculation and reporting of classification metrics:
    - accuracy
    - precision
    - recall
    - F1-score
    - confusion matrix
    - Logical comparison of models (e.g. “Logistic Regression improved recall on positive class compared to non-linear model obtained with feature engineering”).
    - Awareness of class imbalance and its effects on accuracy.

- Model Selection and Justification
  - Clear justification of which model is preferred (not only “higher accuracy” but, where relevant, “better recall on minority / environmentally critical class”).
  - Sensible explanation of the model’s generalization ability based on the results of the test dataset

- Visualization and Presentation of Results
  - The confusion matrix is plotted, labeled, and interpreted.
  - Best model selection based on the actual data from the table or graph

- Visuals are readable and support the argument.

- Connection to Environmental Context
  - The student demonstrates an understanding of misclassification in the context of habitat detection, recognizing that misclassifying non-habitat areas as Spruce/Fir habitat is generally more severe than missing a true habitat instance.
  - A student can explain how such a classifier could support environmental monitoring, conservation planning, and land-use management by helping identify and track the distribution of Spruce/Fir habitats, prioritize areas for protection or field inspection, and inform data-driven environmental decision-making.

FOR SHORT WRITTEN REPORT

- Clarity and Structure of Report
  - The report is logically organized (Introduction → Data → Methods → Results → Discussion → Conclusion).
  - Steps of the workflow are visible in the notebook
  - Language is precise and concise.

- Interpretation of Results and Discussion
  - Classification results (accuracy, precision, recall, F1, confusion matrix) are accurately reported and meaningfully interpreted.
  - The student shows understanding of errors per class from the confusion matrix.
  - The student reflects on why one model performed better than other
  - Discussion reflects awareness of overfitting and underfitting, as well as the role of validation.

- Connection to Ecological and Environmental Context
  - Findings are linked to the real environmental meaning (e.g., “higher recall on Spruce/Fir is more important than overall accuracy”).
  - The student demonstrates an understanding of how data-driven classification can contribute to environmental monitoring and informed decision-making.

- Quality of Visual Presentation
  - Figures (confusion matrix, evaluation metrics, models comparison) are clearly labeled.
  - Visuals are referenced in the text ("as shown in Figure 1…").
  - Visuals help explain model behavior, not just decorate the report.

- Communication and Style
  - Writing is clear, objective, and professional.
  - ML and environmental terminology are used correctly (class, label, feature, precision, recall, positive class).
  - The student acknowledges the limitations (small dataset, imbalance, and lack of a spatial component), the challenges posed by high-dimensional data, and the need for large-dataset subsampling when working in a free Google Colab environment.

QUIZ (10 questions)

1. Supervised learning

Which of the following best describes supervised learning in the context of environmental data analysis?

A. The model discovers patterns in data without predefined labels
B. The model learns a mapping from input features to known target labels
C. The model groups similar environmental observations automatically
D. The model optimizes decision rules without using training data

2. Binary reformulation

Why was the original multi-class forest cover problem reformulated into a binary classification task?

A. Logistic Regression cannot be used for multi-class problems
B. To simplify the interpretation of metrics and decision boundaries
C. The dataset contains only two meaningful cover types
D. To avoid feature scaling

3. Positive class definition (habitat detection)

In the Spruce/Fir habitat detection task, what does the positive class (1) represent?

A. Any forested area
B. Areas with high elevation
C. Locations covered by Spruce/Fir forest
D. Locations with minimal human influence

4. False positives in NO_2 detection

In the NO_2 exceedance classification task, what does a false positive represent?

A. Correctly identifying unsafe air quality
B. Predicting safe air quality when NO_2 exceeds the threshold
C. Predicting NO_2 exceedance when air quality is actually safe
D. A failure to train the model

5. Misclassification severity (habitat detection)

In the habitat detection task, which misclassification is typically considered more environmentally risky?

A. False Negative
B. False Positive
C. True Negative
D. True Positive

6. Metric selection

Which metric is most appropriate if the primary goal is to detect as many true Spruce/Fir habitats as possible?

A. Accuracy
B. Precision
C. Recall
D. Specificity

7. Precision vs. recall

In which situation would precision be more important than recall?

A. When a habitat is very costly

B. When false alarms lead to expensive field inspections
C. When the dataset is perfectly balanced
D. When visualizing decision boundaries

8. Train/test split

What is the main purpose of splitting data into training and test sets?

A. To reduce the dataset size
B. To tune hyperparameters
C. To estimate generalization to unseen data
D. To improve visualization

9. Underfitting vs. overfitting

A model performs very well on training data but poorly on test data. This is an example of:

A. Underfitting
B. Overfitting
C. Class imbalance
D. Random error

10. Decision boundary visualization

Why were decision boundaries visualized in 2D using PCA in the practical exercises?

A. To improve model accuracy
B. To reduce overfitting
C. To make high-dimensional model behavior interpretable
D. To replace evaluation metrics

Suggested Weighting

- Quiz: 20%
- Practical Classification Task (notebook/code): 50%
- Short Written Report: 30%

References / Further Reading

[1] C. O. De Burgh-Day and T. Leeuwenburg, “Machine learning for numerical weather and climate modelling: A review,” Nov. 14, 2023, *Copernicus Publications*. doi: 10.5194/gmd-16-6433-2023.

[2] M. Karmoude *et al.*, “Machine learning for air quality prediction and data analysis: Review on recent advancements, challenges, and outlooks,” Nov. 01, 2025, *Elsevier B.V.* doi: 10.1016/j.scitotenv.2025.180593.

[3] G. O. Ojwang *et al.*, “An integrated hierarchical classification and machine learning approach for mapping land use and land cover in complex social-ecological systems,” *Frontiers in Remote Sensing*, vol. 4, 2023, doi: 10.3389/frsen.2023.1188635.

[4] K. S. Bobba, K. K, V. K. Sai, D. Bugga, and V. M. S. Bolla, “AI-Enhanced Acoustic Analysis for Comprehensive Biodiversity Monitoring and Assessment,” Oct. 2024, [Online]. Available: http://arxiv.org/abs/2410.12897

[5] T. M. . Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[6] A. E. Maxwell, T. A. Warner, and F. Fang, “Implementation of machine-learning classification in remote sensing: An applied review,” May 03, 2018, *Taylor and Francis Ltd.* doi: 10.1080/01431161.2018.1433343.

[7] M. J. Kochenderfer *et al.*, *Decision Making Under Uncertainty*. The MIT Press, 2015. doi: 10.7551/mitpress/10187.001.0001.

[8] F. K. Ewald, L. Bothmann, M. N. Wright, B. Bischl, G. Casalicchio, and G. König, “A Guide to Feature Importance Methods for Scientific Inference,” Aug. 2024, doi: 10.1007/978-3-031-63797-1_22.

[9] Partha Protim Roy, Md. Shahriar Abdullah, and Iqtiar Md. Siddique, “Machine learning empowered geographic information systems: Advancing Spatial analysis and decision making,” *World Journal of Advanced Research and Reviews*, vol. 22, no. 1, pp. 1387–1397, Apr. 2024, doi: 10.30574/wjarr.2024.22.1.1200.

[10] S. Zhong *et al.*, “Machine Learning: New Ideas and Tools in Environmental Science and Engineering,” *Environ. Sci. Technol.*, p. acs.est.1c01339, Aug. 2021, doi: 10.1021/acs.est.1c01339.

[11] T. F. Deressu, A. K. Bojer, T. G. Debelee, W. G. Negera, S. Nadarajah, and K. W. Gebissa, “Enhancing land use and land cover classification with deep learning-based satellite imagery

segmentation,” *International Journal of Applied Earth Observation and Geoinformation*, vol. 144, Nov. 2025, doi: 10.1016/j.jag.2025.104839.

[12] A. Elsayed, J. Levison, A. Binns, M. Larocque, and P. Goel, “Regression-based machine learning models for nitrate and chloride prediction in surface water in a small agricultural sand plain sub-watershed in southwestern Ontario, Canada,” *Front. Environ. Sci.*, vol. 13, 2025, doi: 10.3389/fenvs.2025.1543852.

[13] T. Vo-Van, A. Nguyen-Hai, M. V. Tat-Hong, T. Nguyen-Trang, and F. Gomariz, “A New Clustering Algorithm and Its Application in Assessing the Quality of Underground Water,” *Sci. Program.*, vol. 2020, 2020, doi: 10.1155/2020/6458576.

[14] *Machine Learning in Earth, Environmental and Planetary Sciences*. Elsevier, 2023. doi: 10.1016/C2022-0-00549-3.

[15] F. Pedregosa FABIANPEDREGOSA *et al.*, “Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot,” 2011. [Online]. Available: http://scikit-learn.sourceforge.net.

[16] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O’Reilly Media, Inc., 2023.