LOGISTIC REGRESSION

Before we dive into the code, let's take a moment to understand the intuition behind logistic regression. Imagine you are trying to predict whether someone will buy a product (yes or no) based on features like age and income. This is a classification problem, where the output is binary: 0 (no purchase) or 1 (purchase). You might think: "Let's just draw a line to separate the two groups." That's where logistic regression comes in.

Linear regression tries to draw a straight line that fits your data, but its predictions can be any number, not just 0 or 1.

Logistic regression solves this by using a special function: the sigmoid function, which maps any number to a value between 0 and 1.

This way, we get:

- A value close to 1 → more likely to belong to class 1 (e.g., "will buy")
- A value close to 0 → more likely to belong to class 0 (e.g., "won't buy")

We then choose a threshold (typically 0.5) to decide:

- If the output is > 0.5 → predict class 1
- Otherwise → predict class 0

Logistic regression finds the best line (or curve in higher dimensions) that separates the classes by:

- Estimating the weights (coefficients) of the model
- Minimizing a cost function (called log-loss) to improve the predictions

It's called "regression" because we estimate probabilities, but it's used for classification.

Why It's Useful

- Fast and simple to implement
- Interpretable: you can see the effect of each variable
- Good baseline for many classification tasks

This TP will walk you through:

- Loading and exploring data
- Training a logistic regression model
- Evaluating its performance with predictions, a confusion matrix, and ROC curves
- Visualizing the decision boundary to see how the model separates the classes

Objective:

Implementation of **logistic regression** and use of **Python functions** to visualize all results and interpret them. This lab is highly guided.

1. Load the file Social_Network_Ads.csv and visualize the different variables

- Use pandas to load the file.
- Display the first few rows and explore the columns.

2. Use the iloc function to extract the matrix X of individuals and the vector y of predictions.

Alternative: convert the dataframe into a NumPy array and extract from there.

3. Split the original dataset into a training set and a test set

Use the train_test_split function (or, if you prefer, write your own splitting function from scratch).

4. Using the LogisticRegression class from Python

from sklearn.linear_model import LogisticRegression

- Create a classifier and apply it to the training data.

5. Make predictions on the test set

6. Generate the confusion matrix between predicted test labels and actual test labels

You can use the Python function:

from sklearn.metrics import confusion_matrix

7. Visualize the confusion matrix using sns.heatmap

Import seaborn and use:

import seaborn as sns

8. Visualize the results (from matplotlib.colors import ListedColormap)

We want to draw the **decision boundary map** for all data points.

- First, generate a grid of points X1 (for x-axis) and X2 (for y-axis) between the minimum and maximum values (±1 margin) of the features.
- Use a step of 0.1 and the function np.meshgrid.
- Apply prediction on all grid points (X1, X2).
  Before doing that:
  - Flatten the grids using numpy.ravel(a, order='C')
  - Stack the two vectors to form a matrix with two columns (transpose if needed).
- Use the contourf function to display Z = f(x, y).

You should obtain a graph showing the **(0)** and **(1)** classification zones and the **test points** on the same plot.

To color the decision zones with contourf, you have two options:

- Either generate a color vector (each data point assigned a color)
- Or directly use cmap

9. Finally, we will plot the ROC curve.

Definition (from Wikimedia):
The ROC curve is a measure of the performance of a binary classifier — that is, a system whose goal is to categorize elements into two distinct groups based on one or more characteristics of each element.
Graphically, the ROC curve is often represented as a plot of the **true positive rate** (i.e., the fraction of actual positives correctly identified) against the **false positive rate** (i.e., the fraction of actual negatives incorrectly identified as positives).

Python provides all the necessary tools to directly plot the ROC curve.

from sklearn.metrics import roc_auc_score

from sklearn.metrics import roc_curve

- roc_auc_score(y_test, classifier.predict(X_test))
  → Computes the **Area Under the Curve (AUC)** from the predicted class scores.

- fpr, tpr, thresholds = roc_curve(y_test, classifier.predict_proba(X_test)[:, 1])
  → This function returns:
    - fpr: the vector of **false positive rates**
    - tpr: the vector of **true positive rates**
    - thresholds: the different classification thresholds

Using these values, you can then plot the ROC curve.

You can also refer to a very comprehensive example from the official documentation:
👉 Scikit-learn ROC Curve Example (Iris Dataset)

TUTORIAL ROC CURVE

Tutorial Overview

Predicting Probabilities

In a classification problem, we may decide to predict the class values directly.

Alternatively, it can be more flexible to predict the probabilities for each class instead. The reason for this is to provide the capability to choose and even calibrate the threshold for how to interpret the predicted probabilities.

For example, a default might be to use a threshold of 0.5, meaning that a probability in [0.0, 0.49] is a negative outcome (0) and a probability in [0.5, 1.0] is a positive outcome (1).

This threshold can be adjusted to tune the behavior of the model for a specific problem. An example would be to reduce more of one or another type of error.

When predicting a binary or two-class classification problem, there are two types of errors that we could make.

- • **False Positive**. Predict an event when there was no event.
- • **False Negative**. Predict no event when in fact there was an event.

By predicting probabilities and calibrating a threshold, the operator of the model can strike a balance between these two concerns.

For example, in a smog prediction system, we may be far more concerned with having low false negatives than low false positives. A false negative would mean not warning about a smog day when in fact it is a high smog day, leading to health issues in the public that are unable to take precautions. A false positive means the public would take precautionary measures when they didn't need to.

A common way to compare models that predict probabilities for two-class problems is to use a ROC curve.

What Are ROC Curves?

It is a plot of the false positive rate (x-axis) versus the true positive rate (y-axis) for various candidate threshold values between 0.0 and 1.0. Put another way, it plots the false alarm rate versus the hit rate.

The true positive rate is calculated as the number of true positives divided by the sum of the number of true positives and the number of false negatives. It describes how good the model is at predicting the positive class when the actual outcome is positive. **1**

True Positive Rate = True Positives / (True Positives + False Negatives)