MACHINE LEARNING

Applied to French Football

Predicting Ligue 1 Match Outcomes using AI

Written by

Sami BENNANE

Data & Artificial Intelligence Project

October 23, 2025

1 Introduction

1.1 Project Overview

This project explores the use of machine learning to predict the outcome of French Ligue 1 football matches. The problem is formulated as a **multi-class classification task** where the target variable y can take three possible values:

$$y \in \{\text{Home Win}, \text{Draw}, \text{Away Win}\}.$$

Given a set of features $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ describing each match (teams, goals, etc.), the goal is to approximate a function $f_\theta(\mathbf{x})$ parameterized by θ:

$$\hat{y} = f_\theta(\mathbf{x}) \approx y.$$

1.1.1 Dataset Overview

The dataset $\mathcal{D}$ contains historical results from the French Ligue 1, covering multiple seasons and totaling 7,378 matches. Each observation represents one match and includes both categorical and numerical data:

$$\text{Home Team}, \ \text{Away Team}, \ \text{Home Team Goals}, \ \text{Away Team Goals}, \ \text{Winner}.$$

Formally, the dataset can be written as:

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}, \quad m = 7378.$$

Here:

- $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})$ is the **feature vector** describing match i (for example, the home team, away team, and goals scored),
- $y^{(i)}$ is the **true outcome** of match i (Home Win, Draw, or Away Win),
- m is the **number of samples**, i.e. the total number of matches in the dataset.

In other words, the dataset $\mathcal{D}$ contains $m = 7378$ examples, each composed of a vector of n input features and a corresponding label $y^{(i)}$. Mathematically, this structure is represented as:

$$X \in \mathbb{R}^{m \times n}, \quad y \in \{0, 1, 2\}^m,$$

where each row of X corresponds to a specific Ligue 1 match and each entry in y encodes its outcome.

1.1.2 Feature Selection Rationale

In the original dataset, additional attributes were available such as:

- match date and stadium,
- referee name,
- competition round or season,

- audience or weather conditions.

However, many of these fields either:

1. lack a clear predictive relationship with match outcomes (e.g., referee name), or
2. introduce unnecessary noise or high-dimensional sparsity (hundreds of unique stadiums).

To keep the model simple and interpretable, we selected only the most informative and universally available features:

$$\mathbf{X} = [\textbf{Home Team, Away Team, Home Team Goals, Away Team Goals}].$$

This ensures the model focuses on fundamental competitive factors rather than context-dependent randomness.

1.1.3 Matrix Representation and Python Integration

After preprocessing, the data is represented in matrix form:

$$X \in \mathbb{R}^{m \times n}, \quad y \in \{0, 1, 2\}^m,$$

where:

- m is the number of samples (i.e., the total number of matches, here $m = 7378$),
- n is the number of features describing each match (here $n = 82$ after encoding),
- X contains both numerical and one-hot encoded categorical variables,
- y contains the encoded outcomes (`Home Win = 0`, `Draw = 1`, `Away Win = 2`).

Each row of X corresponds to one Ligue 1 match, and each column corresponds to a measurable or categorical characteristic of that match. For example:

$$X^{(i)} = [\text{Home\_Lyon}, \text{Away\_PSG}, \text{Goals\_Home} = 1, \text{Goals\_Away} = 2, \dots]$$

represents all the information associated with match i between Lyon and Paris SG.

In Python, this transformation is achieved using `pandas.get_dummies()` for categorical variables and `LabelEncoder()` for the target variable:

$$X_{\text{final}} = [\text{One-Hot(Home Team)}, \text{One-Hot(Away Team)}, \text{Goals\_Home}, \text{Goals\_Away}]$$

This results in a rectangular matrix X of shape (m, n) ready to be used in vectorized linear algebra operations.

This matrix representation is essential since most machine learning algorithms—including logistic regression—are defined as vectorized models:

$$\hat{\mathbf{y}} = \sigma(X\boldsymbol{\theta}),$$

where:

- X is the **design matrix** (the input data),
- $\boldsymbol{\theta}$ is the **parameter vector** learned during training,
- σ is the **activation function** (softmax in the multiclass case).

This explicit matrix formulation allows efficient computation with `NumPy` and `scikit-learn`, bridging the theoretical foundation of machine learning with its practical implementation in Python.

1.1.4 Football Interpretation

From a football perspective:

- The team identifiers in X encode each club's identity and past performance trends.
- The goal statistics provide a quantitative link between offensive/defensive strength and the match outcome.
- The model captures realistic patterns, such as the **home advantage effect** or the dominance of top teams like PSG.

By combining mathematical modeling and domain intuition, the project aims to connect statistical learning with real-world football dynamics.

2 Mathematical Model

2.1 Logistic Regression: Theory

For binary classification, logistic regression models the probability:

$$P(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{x}^\top \boldsymbol{\theta}) = \frac{1}{1 + e^{-\mathbf{x}^\top \boldsymbol{\theta}}}.$$

For K outcomes (Home, Draw, Away), the model generalizes via the **softmax function**:

$$P(y = k \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\top \boldsymbol{\theta}_k}}{\sum_{j=1}^{K} e^{\mathbf{x}^\top \boldsymbol{\theta}_j}}, \quad k = 1, \ldots, K,$$

where each $\boldsymbol{\theta}_k$ corresponds to a class (e.g., Home Win).

3 Matrix Formulation and Python Implementation

3.1 Matrix Representation

Let:

$$X = \begin{bmatrix} | & | & & | \\ \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \cdots & \mathbf{x}^{(m)} \\ | & | & & | \end{bmatrix}^\top \in \mathbb{R}^{m \times n}, \quad \boldsymbol{\Theta} = \begin{bmatrix} \boldsymbol{\theta}_1 & \boldsymbol{\theta}_2 & \cdots & \boldsymbol{\theta}_K \end{bmatrix} \in \mathbb{R}^{n \times K}.$$

The model outputs:

$$\hat{Y} = \mathrm{softmax}(X\boldsymbol{\Theta}) \in \mathbb{R}^{m \times K},$$

where each row $\hat{y}^{(i)}$ is a probability vector over all K classes.

The vectorized **cost function** is:

$$J(\boldsymbol{\Theta}) = -\frac{1}{m} \mathrm{Tr}\left(Y^\top \log(\hat{Y})\right),$$

and its gradient:

$$\nabla_{\boldsymbol{\Theta}} J = \frac{1}{m} X^\top (\hat{Y} - Y).$$

These compact equations show how all m examples are processed in parallel through efficient matrix operations.

3.2 Python Implementation

Listing 1: Vectorized implementation of multinomial logistic regression.

```
# X: input matrix (m, n)
# Theta: parameters (n, K)
# y_onehot: true labels in one-hot encoding (m, K)

z = X @ Theta                                          # Linear step
y_hat = np.exp(z) / np.sum(np.exp(z), axis=1, keepdims=True)  # Softmax
cost = -(1/m) * np.sum(y_onehot * np.log(y_hat))                # Loss

grad = (1/m) * X.T @ (y_hat - y_onehot)                         # Gradient
Theta -= learning_rate * grad                                   # Update rule
```

This implementation corresponds exactly to the analytical formula:

$$\boldsymbol{\Theta} := \boldsymbol{\Theta} - \eta \nabla_{\boldsymbol{\Theta}} J(\boldsymbol{\Theta}),$$

where η is the learning rate.

4 Data Processing

Categorical variables such as *Home Team* and *Away Team* were encoded via **one-hot encoding**:

$$X_{\text{encoded}} = [\text{Home\_Team\_Lyon}, \text{Away\_Team\_PSG}, \ldots, \text{Goals\_Home}, \text{Goals\_Away}].$$

After preprocessing:

$$X \in \mathbb{R}^{7378\times 82}, \quad y \in \{0, 1, 2, \ldots, 40\}.$$

The data was then split into 80% training and 20% testing subsets.

5 Results and Analysis

The multinomial logistic regression achieved a test accuracy of:

$$\text{Accuracy} = 54.47\%.$$

This performance, although modest, reflects the inherent unpredictability of football results. Nevertheless, the model successfully captured strong tendencies such as *home advantage* and the dominance of major teams.

5.1 Interpretation

The model parameters $\boldsymbol{\theta}_{\text{home}}$ and $\boldsymbol{\theta}_{\text{away}}$ act as learned weights reflecting each team's influence. The predicted class is given by:

$$\hat{y} = \arg\max_k \mathbf{x}^\top \boldsymbol{\theta}_k.$$

Teams with larger positive weights (e.g., PSG, Lyon) tend to yield higher predicted probabilities for winning at home.

6 Evaluation Metrics and Football Interpretation

Evaluating a machine learning model is crucial to understanding both its predictive power and its limitations. In this project, we focus on the most relevant metrics for football match prediction: **Accuracy**, **Precision**, **Recall**, **F1-score**, and their aggregated versions (**macro** and **weighted averages**).

6.1 Accuracy

The simplest and most intuitive metric, accuracy measures the overall proportion of correct predictions:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}.$$

For instance, if the model correctly predicts 800 matches out of 1476:

$$\text{Accuracy} = \frac{800}{1476} \approx 54.47\%.$$

In football terms, this means the model "guesses" the right outcome slightly more than once every two matches.

6.2 Precision and Recall

To evaluate the quality of predictions for each possible outcome (Home Win, Draw, Away Win), we use:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}.$$

- **Precision** indicates how often the model is correct when it predicts a given result. For example, when the model predicts a *home win*, how often is it right?
- **Recall** measures how well the model identifies all true occurrences. For example, among all matches that actually ended with a home win, how many did the model correctly detect?

The harmonic mean of both gives the **F1-score**:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

6.3 Macro and Weighted Averages

Because some classes (teams or results) appear more often than others, we use two types of averages:

$$\text{Macro Average} = \frac{1}{K} \sum_{k=1}^{K} \text{Metric}_k$$

$$\text{Weighted Average} = \frac{\sum_{k=1}^{K} n_k \times \text{Metric}_k}{\sum_{k=1}^{K} n_k}$$

where n_k is the number of samples (matches) in class k.

- **Macro average** gives equal importance to all classes — useful to see if the model treats small clubs (e.g., Metz, Brest) fairly.
- **Weighted average** gives more weight to frequent classes — which in football means the model performance is driven by the big clubs that dominate the dataset (e.g., PSG, Lyon, Marseille).

6.4 Football Interpretation

In our Ligue 1 dataset:

- The **macro average** is slightly lower than the accuracy, showing that smaller or less frequent outcomes (like draws or away wins) are harder to predict.
- The **weighted average** is close to the accuracy, indicating that the model is consistent on teams with many games (the major clubs).

This means the model captures strong patterns such as:

- Home advantage (more likely to predict a home win correctly).
- The dominance of top teams (PSG, Lyon, Marseille are predicted more accurately).
- Greater uncertainty for balanced or rare outcomes (draws, small teams winning away).

6.5 Visual Representation

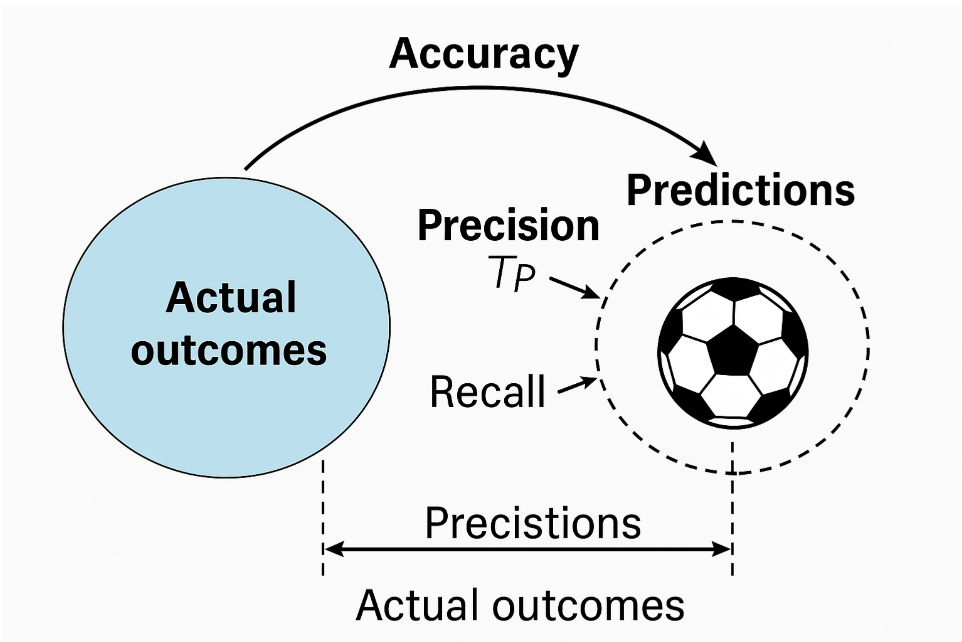

Figure: Conceptual diagram showing the relationship between predictions and actual outcomes. A perfect model would have precision and recall equal to 1. In football prediction, some imprecision is inevitable due to random events (injuries, referee decisions, etc.).

7 Conclusion

This project demonstrates the full pipeline of a machine learning approach — from data preprocessing to mathematical modeling and numerical optimization.

Key takeaways:

- The mathematical model can be entirely expressed in **matrix form**.
- NumPy seamlessly implements these vectorized operations, achieving high computational efficiency.
- Extending this model to neural networks would only add more layers of similar matrix multiplications.

Future improvements could include:

- Adding regularization:
$$J_{\text{reg}}(\mathbf{\Theta}) = J(\mathbf{\Theta}) + \lambda \|\mathbf{\Theta}\|_2^2$$
- Using ensemble models or neural networks.
- Integrating temporal dynamics of teams with LSTM-based models.

Keywords: Machine Learning, Logistic Regression, Matrix Formulation, NumPy, Sports Analytics, French Ligue 1

Appendix — Complete Gradient Derivation via Differentials

We now derive the gradient of the multiclass logistic regression cost function using the **differential approach**, providing a step-by-step derivation with all details.

1. Problem Setup and Definitions

Let:

$$J(W) = \frac{1}{m}\sum_{i=1}^{m} L_i, \quad L_i = -\sum_{k=1}^{K} Y_{ik}\log(\hat{Y}_{ik}), \quad \hat{Y} = \text{softmax}(Z), \quad Z = XW$$

where:

- $X \in \mathbb{R}^{m\times n}$ is the data matrix (m examples, n features)
- $W \in \mathbb{R}^{n\times K}$ are the model parameters
- $Y, \hat{Y} \in \mathbb{R}^{m\times K}$ are the true and predicted label matrices
- $Z \in \mathbb{R}^{m\times K}$ are the logits

2. Step 1: Gradient of Loss with Respect to Logits

First, we compute $\frac{\partial J}{\partial Z}$.

2.1 For one example L_i:

$$\frac{\partial L_i}{\partial z_j} = -\sum_{k=1}^{K} Y_{ik}\frac{1}{\hat{Y}_{ik}}\frac{\partial \hat{Y}_{ik}}{\partial z_j}$$

2.2 Using softmax derivatives:

$$\frac{\partial \hat{Y}_{ik}}{\partial z_j} = \hat{Y}_{ik}(\delta_{kj} - \hat{Y}_{ij})$$

where δ_{kj} is the Kronecker delta.

2.3 Substitution and simplification:

$$\begin{aligned}
\frac{\partial L_i}{\partial z_j} &= -\sum_{k=1}^{K} Y_{ik} \frac{1}{\hat{Y}_{ik}} \hat{Y}_{ik}(\delta_{kj} - \hat{Y}_{ij}) \\
&= -\sum_{k=1}^{K} Y_{ik}(\delta_{kj} - \hat{Y}_{ij}) \\
&= -Y_{ij} + \hat{Y}_{ij} \sum_{k=1}^{K} Y_{ik} \\
&= \hat{Y}_{ij} - Y_{ij} \quad \left(\text{since } \sum_{k=1}^{K} Y_{ik} = 1 \text{ for one-hot vectors}\right)
\end{aligned}$$

2.4 For all examples (matrix form):

$$\frac{\partial J}{\partial Z} = \frac{1}{m}(\hat{Y} - Y)$$

3. Step 2: Differential of the Cost Function

Using the definition of differential via inner product:

$$dJ = \left\langle \frac{\partial J}{\partial Z}, dZ \right\rangle = \mathrm{Tr}\left(\left(\frac{\partial J}{\partial Z} \right)^{\top} dZ \right)$$

Substitute $\frac{\partial J}{\partial Z} = \frac{1}{m}(\hat{Y} - Y)$:

$$dJ = \left\langle \frac{1}{m}(\hat{Y} - Y), dZ \right\rangle$$

4. Step 3: Relating dZ and dW

Since $Z = XW$ and X is constant:

$$dZ = X \cdot dW$$

Substitute into the differential:

$$dJ = \left\langle \frac{1}{m}(\hat{Y} - Y), X \cdot dW \right\rangle$$

5. Step 4: Matrix Inner Product Manipulation

We use the property of matrix inner products:

$$\langle A, BC \rangle = \langle B^{\top} A, C \rangle$$

Apply this with $A = \frac{1}{m}(\hat{Y} - Y)$, $B = X$, $C = dW$:

$$dJ = \left\langle X^{\top} \cdot \frac{1}{m}(\hat{Y} - Y), dW \right\rangle$$

6. Step 5: Identification of the Gradient

By definition of the gradient in matrix calculus:

$$dJ = \langle \nabla_W J, dW \rangle$$

Comparing with our expression:

$$\langle \nabla_W J, dW \rangle = \left\langle \frac{1}{m} X^\top (\hat{Y} - Y), dW \right\rangle$$

Since this holds for all dW, we can identify:

$$\boxed{\nabla_W J = \frac{1}{m} X^\top (\hat{Y} - Y)}$$

7. Dimension Verification

- $X^\top \in \mathbb{R}^{n \times m}$
- $(\hat{Y} - Y) \in \mathbb{R}^{m \times K}$
- $X^\top (\hat{Y} - Y) \in \mathbb{R}^{n \times K}$ (matches dimensions of W)

8. Interpretation and Usage

The differential approach shows that:

$$dJ = \langle \nabla_W J, dW \rangle$$

meaning $\nabla_W J$ gives the direction of steepest ascent of J.

In gradient descent:

$$W \leftarrow W - \eta \nabla_W J$$

parameters are updated opposite to the gradient, reducing the loss.

Key Insights:

- The Kronecker delta property and one-hot encoding simplification are crucial
- Matrix inner products provide elegant notation for chain rule applications
- The final result $\nabla_W J = \frac{1}{m} X^\top (\hat{Y} - Y)$ is remarkably simple
- This approach generalizes naturally to neural networks and backpropagation