Machine Learning Project 35 Report

Gabriel Deregnaucourt and Paul Ritzinger

January 18, 2026

Code Structure: `Part1.py` handles data preprocessing, `Part2.ipynb` identifies the best model through optimization, and `Part3.ipynb` generates the final predictions.

1 Data Cleaning and Pre-processing

We applied the same pre-processing steps before deciding which model is the best to predict the target variable on the test set.

1.1 Main Data Set

First, we load the main dataset (`learn_dataset.csv`). It contains 50,046 observations, including both numerical and categorical variables. The target variable is binary and slightly unbalanced, with 53.86% for class L and 46.14% for class T.

There are no missing values in the main dataset. We checked for inconsistencies, for example the age range. We noticed that some individuals with the 'retired' status are studying. However, we decided not to change this, as there are only a few cases and retirees can still attend classes.

1.2 Sport Dataset

Next, we loaded the `learn_dataset_sport.csv` dataset. It links the 'Unique_ID' to an associated sports club and contains 6,433 registered individuals. Since there are 97 different clubs and some have very few observations, we decided to group them.

We merged the data with `code_CLUB.csv`, which contains the club names and their corresponding sport categories. We created the `sport_categorie` variable using the following rule: for clubs with more than 50 athletes (calculated strictly on the learning set to avoid data leakage), we kept the specific club name; otherwise, we used the broader category. Finally, we merged this into the main dataset, keeping only our new `sport_categorie` variable.

1.3 Geographical Variables

We applied a string conversion to the 'Insee' variable. Given the high number of categories and rare modalities, we grouped cities by department and region using the `departments.csv` dataset. However, we chose to preserve the specific Insee code for cities with at least 50 observations in a new variable called 'Big_City'.

We also merged the main dataframe with municipality types and 'Residents' (city population) using their respective CSV files. We incorporated X and Y coordinates to provide the model with precise spatial positioning, allowing it to detect geographical clusters beyond administrative borders.

Finally, to refine our geographical data, we used three different external sources:

- `city_density.csv`: Population density per Insee code.
- `med_earnings_city.csv`: Median revenue per city.
- `dvf2018.csv`: Average price per square meter for each city.

To ensure consistency across these files, we mapped the specific arrondissements of Paris, Lyon, and Marseille (PLM) to their global city codes. Regarding missing values for density and revenue, we decided not to impute with the median, based on the assumption that the absence of data is informative in itself.

Geographical variables proved to be excellent predictors. We attempted to further enrich this aspect by integrating the 2022 Presidential Election results (sourced from the official government database). Unfortunately, including these variables led to a decrease in model accuracy. Consequently, we decided to exclude them from the final model to maintain optimal performance.

1.4 Job Tenure and Retired Former

We merged the main dataset with `learn_dataset_employement_tenure.csv`, which identifies the working population. We also merged it with `learn_dataset_retired_former.csv` to capture data on the retired population.

We observed that approximately half of the sample is currently working, while a quarter is retired. We noticed a discrepancy: some individuals were identified as retired (or pre-retired) in the main data but did not appear in the `retired_former` dataframe. This motivated us to impute the retirement age under specific conditions. If an individual was older than 62 (the legal retirement age in France in 2018) but missing from the former file, we imputed their `retirement_age` using the median value. However, we chose not to impute `retirement_benefits`, as we considered that doing so would lead to a loss of information or introduce noise.

1.5 Job Dataset

We loaded the job dataset, which contains information about employees' current roles. The `work_desc` variable had too many categories (406). We chose to map it to the 'N2' nomenclature, which aligns well with the `job_42` variable in the main dataset. It offers a good compromise between the very large N1 level and the too detailed N3 level.

We identified missing values in `working_hours`, `Job_dep`, `employee_count`, and `employer_type`. To handle missing `working_hours`, we implemented a hierarchical imputation strategy:

1. We first tried to impute using the median calculated by CSP, contract type, and job condition.
2. If no median was found, we used a back-up mechanism, progressively removing variables from the group to find a match.

We noticed that 4,881 individuals appeared in the job tenure dataset but were not classified as employees. We broadly defined this population as 'independents'. We filled the categorical job-related columns with the label 'Independent' to distinguish them from the non-working population.

1.6 Retired

We merged the main dataset with `learn_dataset_retired_pension.csv` to retrieve retirement benefits. Then, we loaded the `retired_jobs` dataset. We replaced `work_desc` with 'N2'.

As imputation can be difficult for categorical variables, we chose to fill missing values with 'Unknown' to distinguish retirees with known past employment from the others. We added the prefix *former* to each variable to avoid confusion with the active job dataset.

Note: We define the rules for the data preparation with only the learning set and not the test set to avoid data leakage. So all thresholds (median, big_ club) are calculated in the learning set.

2 Model Selection and Optimization Strategy

To ensure a robust evaluation, we split the learning dataset into a training set (80%) and a validation set (20%) using stratified sampling.

2.1 Baseline and Exploratory Benchmark

We started by establishing a baseline using a simple Decision Tree, which achieved an accuracy of 72.78% but failed to capture the minority class effectively. Following this, we conducted an extensive testing phase to challenge various algorithms (Random Forest, XGBoost, LightGBM, CatBoost) and metrics (Accuracy, ROC-AUC, Recall, F1-Score).

2.2 The Choice of XGBoost and F1-Score

After analyzing the results, XGBoost proved to be the most robust candidate, particularly in its ability to handle missing values and heterogeneous features without extensive imputation. Regarding the metric, we moved away from Accuracy (which can be misleading on unbalanced datasets) and ROC-AUC (which is sometimes too optimistic). We selected the **F1-Score** as our primary objective. This metric forces the model to find a tangible compromise between Precision and Recall. We found an interesting trade-off between less accuracy and more recall on 'T' by using F1-Score.

2.3 Final Comparison: Random Forest vs. XGBoost

To keep this report concise, we focused the final analysis on the two strongest contenders: Random Forest (Bagging) and XGBoost (Boosting).

A. Quantitative Analysis (Metrics)

XGBoost outperformed Random Forest across all key indicators:

Table 1: Cross-Validation Comparison (Optimization Target: F1-Score on Class T)

Model	Mean F1	Std Dev	Best Parameters
Random Forest	0.712	± 0.006	*class_weight='balanced', max_depth=None, min_samples_leaf=4*
XGBoost	**0.731**	**±0.002**	*scale_pos_weight=1.2, max_depth=6, learning_rate=0.1*

Note: XGBoost shows a higher mean F1-Score and a significantly lower standard deviation compared to Random Forest.

- **Validation Accuracy:** XGBoost reached 75.94%, gaining nearly +2 points over Random Forest (74.05%).
- **Class 'T' Detection (Minority):**
  - *Recall:* XGBoost identified 72% of the 'T' profiles, compared to only 69% for RF.
  - *Precision:* XGBoost was also more precise (0.75 vs 0.73).
  - *F1-Score:* XGBoost achieved 0.74, significantly beating the RF score of 0.71.

B. Qualitative Analysis (Feature Importance)

Both models are different when making decisions.

- **Random Forest** relied heavily on geographic and demographic noise (X coordinate, Residents, Densité, Y).
- **XGBoost** successfully captured the socio-economic logic. Its top features are Act (Activity status) and `employement_tenure`. It understood that professional situation is a better predictor of the target than simply "where the person lives."

2.4 Optimization Process

We used `GridSearchCV` with Stratified K-Fold ($k = 5$) to tune both models. The search space included depth control (`max_depth`), ensemble size (`n_estimators`), and specifically class balancing parameters (`scale_pos_weight` for XGBoost and `class_weight` for RF) to maximize the F1-Score on the 'T' class.

We tried to build a different model by type of population (splitting workforce, non-working, retired). However, this attempt was unsuccessful and led to a decrease in the model score.

3 Experimental Results and Assessment

3.1 Estimation of Expected Quality

We evaluated our final model (XGBoost) on a validation set that the model had never seen during training. Based on these results, we can confidently estimate its future performance on the test data:

"The expected performances of our final model on new data are a accuracy of approximately 75.9% and a weighted F1-Score of 0.76."

This means we expect to correctly classify about 3 out of 4 individuals, which is a significant improvement over our initial baseline.

3.2 Full Assessment of Predictions

To understand where our model makes mistakes, we used two main visualization tools.

Confusion Matrix Analysis

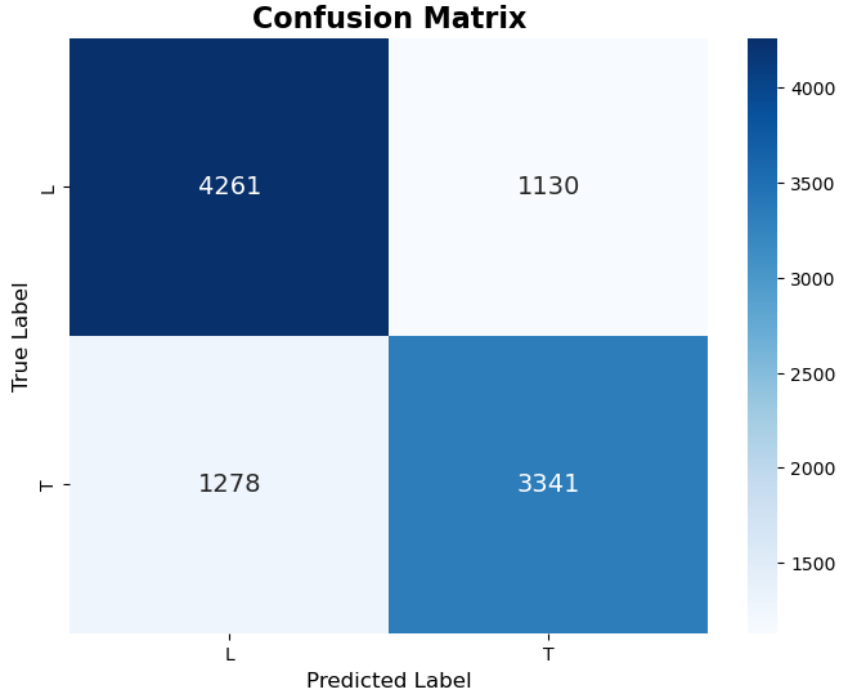

Figure 1: Confusion Matrix

Looking at the matrix (Figure 1), we can see that the model is quite balanced. It correctly identifies 3341 people from the 'T' class (True Positives) but misses 1278 (False Negatives). On the other side, it wrongly classifies 1130 people as 'T' when they are actually 'L' (False Positives).

What is interesting here is that the number of False Positives and False Negatives is very similar. This shows that our parameter tuning (specifically `scale_pos_weight`) worked well: the model doesn't favor one class too much over the other. It accepts a trade-off to capture as many 'T' profiles as possible.

ROC Curve Analysis

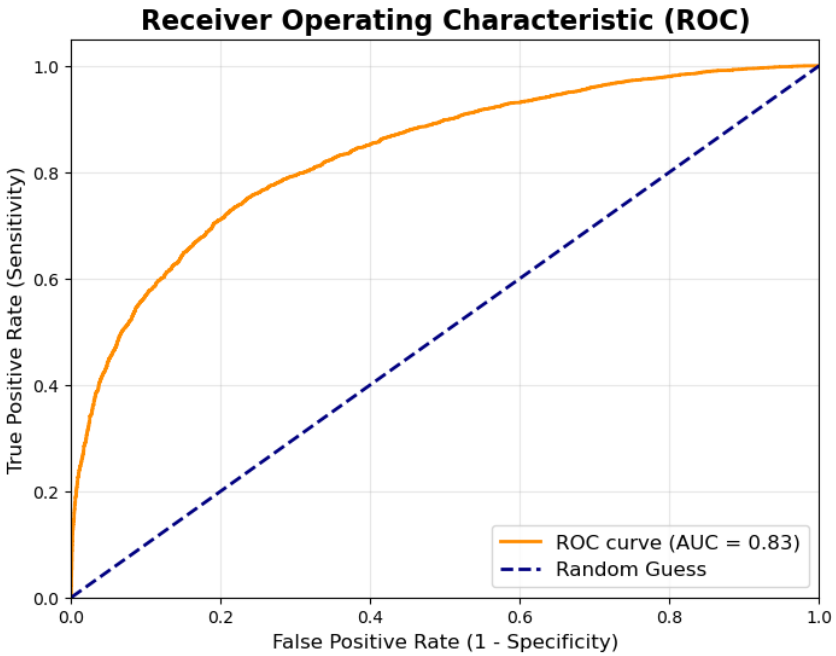

Figure 2: Receiver Operating Characteristic (ROC)

The ROC curve (Figure 2) is well above the diagonal line (which represents random guessing). We obtained an **AUC (Area Under Curve) of 0.83**. Since an AUC of 0.5 is random and 1.0 is perfect, a score of 0.83 confirms that our classifier has a good ability to distinguish between the two classes, regardless of the threshold used.

3.3 Additional Assessment: Variable Importance

Finally, as requested, we analyzed the variables that drive our final predictions to ensure they make sense.

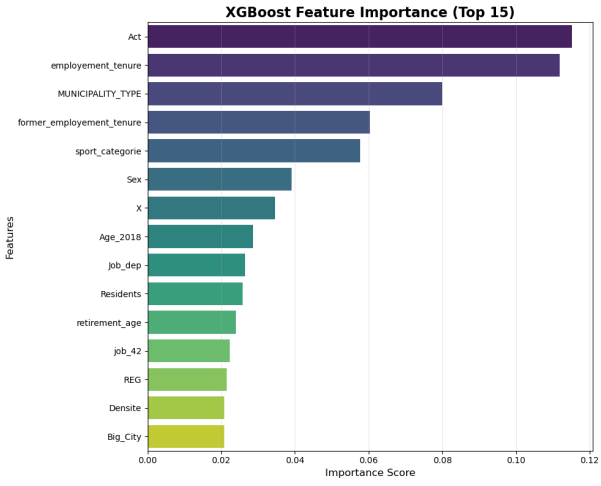

Figure 3

The graph confirms that our model is learning from logical socio-economic patterns rather than just memorizing data:

- **Importance of professional situation:** The two most powerful predictors are Activity Status (`Act`) and Employment Type (`employement_tenure`). This is very reassuring: the model identifies that the type of contract is the best way to predict an individual's target. It shows the model is using "real-world" logic rather than just memorizing data.
- **Some nuance** The model also uses `MUNICIPALITY_TYPE` and `Sport_categorie`. This shows it combines professional data with lifestyle and environment context to refine its decision.
- **Robustness:** Since the model relies on these solid concepts rather than random noise, we can assume the predictions will remain reliable on the new test data.

3.4 Final Distribution Check

An analysis of the generated `predictions.csv` file reveals a class distribution of approximately 44.61% for class 'T' and 55.39% for class 'L'. This ratio is really close to the balance observed in the original learning dataset.