Churn Prediction on a Music Streaming Platform

Ithier d'Aramon
Gaspard Baumelou

December 2025

Contents

1 Introduction

User retention is a central challenge for music streaming platforms. The goal of this project is to build a *churn prediction* model that can anticipate which users are likely to leave the service in the near future, using their past usage logs.
We work with a dataset from a music streaming service, for a Kaggle competition. It contains about 17 million user events (rows) generated by 19,140 users between October 1st and November 20th, 2018.

2 Data and Initial Understanding of the Task

Each row of the dataset corresponds to one user event. It contains at least:

- a user identifier;
- a timestamp and a session identifier;
- metadata such as device and subscription level (`free` vs `paid`);
- a page or action name (e.g. `NextSong`, `ThumbsUp`, `AddToPlaylist`, `Home`, `Error`, `RollAdvert`, `Cancellation Confirmation`).

The global objective is to predict which users will churn *during the 10 days following the last log*. A churner is defined by the fact he visited or not the page `Cancellation Confirmation`. Nevertheless, at this stage, our understanding of the exact label and temporal constraints was still approximate. Therefore, we started by exploring the raw events, without committing to a final target definition.

3 Exploratory Data Analysis

3.1 Which pages do churners visit?

We first looked at the distribution of page visits for the users eventually labeled as churners vs non-churners in the training data.

We observed that churners spend more time on pages associated with friction or account changes: `RollAdvert`, `Downgrade`, `ThumbsDown`, `Cancel`, `Settings`.

Non-churners, on the other hand, visit more pages associated with positive engagement: `AddToPlaylist`, `AddFriend`, `ThumbsUp`, and especially `NextSong`.

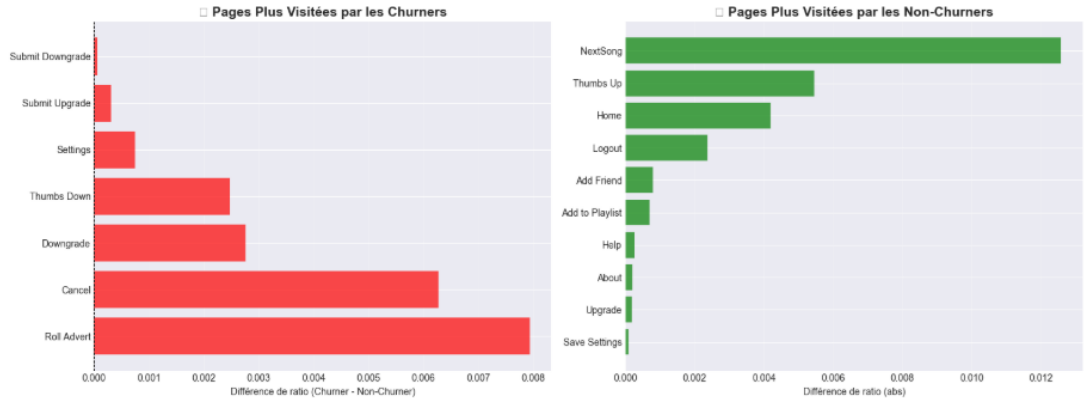

Figure 1: Pages that are disproportionately visited by churners (left) and non-churners (right).

This suggested that ratios of page types could be informative features.

3.2 How active are churners over time?

Then, we inspected the overall activity level: total number of events per user, number of active days, and the date of last activity. Two patterns clearly emerged:

- churners have a shorter active period and fewer events on average;
- their last event typically happens much earlier than for non-churners.

More precisely, the median last activity is around October 23rd for churners, and around November 15th for non-churners. Only about 1.7% of churners are still active on November 19th–20th, versus about 24% of non-churners.

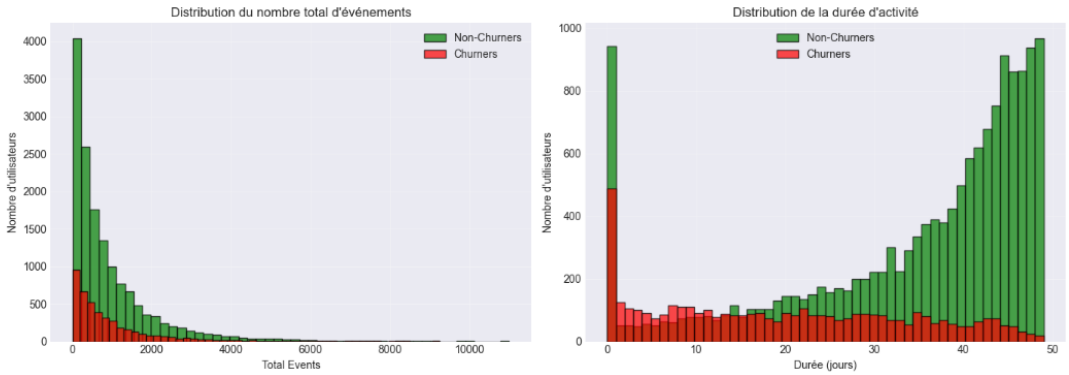

Figure 2: Distribution of total events (left) and active duration (right) for churners vs non-churners.

4 First Modeling Attempt: Simple Churn Definition and Baseline Model

4.1 Step 1: a very simple churn label

For our first model, we needed a concrete binary label. As explained, a churner was a user who had visited the `Cancellation Confirmation` page.
Thus, our initial rule was:

- if a user visits `Cancellation Confirmation` at least once between Oct 1st and Nov 20th, we label them as `churner` (1);
- otherwise, we label them as `non-churner` (0).

This *operational* definition is easy to compute and closely matches the idea of "explicit account cancellation". At this stage, we did not yet worry about the precise timing of churn versus the evaluation period.

4.2 Step 2: basic feature set

For the baseline model, we engineered a small set of 29 features, including:

- total number of events, number of sessions, average session length;
- counts and ratios of positive and negative pages;
- several features computed over "the last X days before the user's last activity" (e.g. number of events in the last 7 days, last 3 days, etc.).

These last-activity-based features already exploited the strong signal seen in EDA: churners tend to stop much earlier than non-churners.

4.3 Step 3: baseline model and first surprise

We trained an XGBoost classifier with default paramters. Even if the final metric used (for Kaggle leaderboard) was the accuracy of our prediction, we decided to use ROC-AUC for the training as the proportion of churners was really small compared to non-churners.
Locally, the ROC–AUC on the validation set was very high. However, when we submitted predictions to Kaggle, the leaderboard score was only about 0.51 – barely better than random.

This discrepancy (excellent local AUC vs very poor Kaggle score) was our first clear sign that something was fundamentally wrong with our problem formulation.

4.4 Revisiting the Problem: What Are We Really Predicting?

To diagnose the issue, we went back to:

- the Kaggle task: it asks us to predict churn *after* the observation window (Nov 21–30);
- the EDA results on last activity dates;
- the way we were building features and splitting data.

We realized two critical points:

- Our label used any `Cancellation Confirmation` event inside the observation window (Oct 1–Nov 20), effectively training the model to detect churn events that had already happened.
- Many of our features were computed relative to each user's last activity date. Since churners tend to stop earlier, these features captured "how early the user stopped" more than their underlying behavior.

In other words, we had built a model that was very good at recognizing users who already cancelled, but not at forecasting who would churn *later*. This explains why the local validation was over-optimistic: we were predicting the wrong thing.

5 Improving the Model: Better Features and Better Validation

5.1 Removing temporal leakage: unbiased features

Then, our first idea was to redesign the features so that they do not rely on the date of last activity. Instead of "last 7 days before the user's last event", we computed aggregates over the fixed observation window (Oct 1–Nov 20).

This gave a more stable set of about 25 features:

- **usage intensity:** total events, number of active days, number of sessions, events per active day, average session length;
- **engagement:** ratios of `ThumbsUp`, `AddToPlaylist`, `AddFriend`, `NextSong`, combined into an engagement score;
- *paid_ratio*, downgrade indicator;
- **negative interactions:** error ratio, thumbs-down ratio, settings/downgrade page ratios.

We retrained XGBoost with class weighting (`scale_pos_weight` ≈ 3.5). The validation ROC–AUC dropped to a more realistic value (≈ 0.77), but the Kaggle score improved to about 0.58. This confirmed we were moving in the right direction: less leakage, better generalization.

5.2 Richer behavioral features

Once the basic leakage issue was addressed, we went back to EDA to design richer features capturing temporal dynamics and user value. We introduced:

- **recency and RFM:** days since last activity, last song, registration; recency/frequency/monetary percentiles and a combined *rfm_score*;
- **temporal trends:** activity per time quartile (Q1–Q4), Q4 ratio, Q1-to-Q4 decline ratio, early/late activity ratio;
- **rolling windows:** events in the last 7 days of the observation period and their share of total activity; short-term acceleration metrics;
- **engagement composition:** positive vs negative engagement, diversity of pages, volatility of hourly activity;
- **automatically generated page-level ratios:** for each page type, count divided by total events (e.g. nextsong_ratio, error_ratio, logout_ratio).

The idea was to use these quartile to help predicting future churns more than passed churns. With these features, XGBoost and LightGBM models performed much better. An ensemble of XGB+LGBM with 5-fold cross-validation reached a validation ROC–AUC close to 0.90 and a Kaggle score around 0.628.

6 Revising the target and final results

6.1 A new target definition

In the last stage, we focused on improving the feature engineering we made and stop the leakage of data. But we still had the problem of target. Indeed, we were not trying to predict a future churn in the target definition itself. Moreover, the span of 10 days after last log was something completely missing from our work yet.

Thus, we decided to set a time cutoff and use the logs before the cutoff to determine if a user had churned in the 10 days following it. Then, as shown in Figure 3, we repeated the method with different cutoffs in order to use as much data as we can.

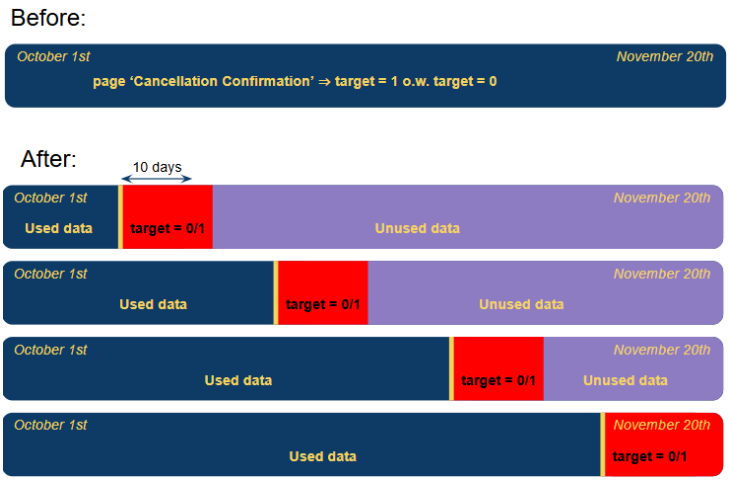

Figure 3: New definition of our target

6.2 New models with this target

We trained a LightGMB model with this new target and managed to reach an accuracy of 0.643 on Kaggle despite a validation ROC-AUC of 0.72 (which was less than our previous models). Therefore this new model was much more adapted to our problem as the metric of the validation set was closer to the test one: we had no over-optimistic validation metrics anymore.

Finally, we decided to fit an Autogluon model with this new target to try obtaining an even better result. And thanks to that we managed to reach 0.655 of accuracy on Kaggle!

6.3 Summary of the results

Stage	Main change	Val ROC–AUC	Kaggle
Baseline	Naïve label + leaky features	over-optimistic	≈ 0.51
Step 1	Unbiased aggregates	≈ 0.77	≈ 0.58
Step 2	Advanced temporal + RFM features	≈ 0.90	≈ 0.63
Step 3	Tuned LightGBM + same features	≈ 0.90	≈ 0.659

Table 1: How each modeling step improved the performance.

7 Conclusion

Our project illustrates how a churn prediction problem must be approached step by step:

- EDA first, to understand basic behavioral patterns and potential sources of leakage;
- a simple baseline model, revealing inconsistencies between local validation and the task;
- a careful redefinition of the label and the time horizon;
- progressive enrichment of the feature set, guided by domain intuition and by EDA;
- software engineering improvements to support fast, reliable experimentation.

Most of our gains came from the *reasoning process* (label, features, validation) rather than from changing the model family. Indeed, some really complex model using autogluon for instance were in the end way worst than a simple logistic regression with better sofwatre engineering and target definition. Thus, what really matters, improves the score and what we spent the most time on was to understand the request and analyse the data to adapt the features and the model to what we had to tackle.