Sparkify: Churn Prediction with PySpark

Author : Moussa Abdi

June 14th, 2020

Table of content

List of Figures

1 Introduction

1.1 Project description

In this project, we study the data of a music streaming application in order to identify the users that are likely to churn.

Multiple users interact in parallel with the application, and for each action an event is created (like e.g. playing a new song or listening to an ad) and added to a database with the corresponding caracteristics (time stamp, type of event, etc...).

The goal of the project is to use the generated data in order to perform churn prediction. The dataframe is analyzed using PySpark.

The dataset is provided by Udacity in the context of the capstone project of the Data Scientist Nanodegree program.

1.2 Project steps overview

The first step of the project will consist in exploratory data analysis.

First, we will do some analysis on the initial dataframe to have a first insight into the data.

Then, we will explain why this is more useful to construct a new user-indexed dataframe, and we will perform a deeper understanding of the data using this new dataframe.

After exploratory data analysis, we will build some features and inject them in a model that will be ran using PySpark. In terms of modeling, we will also perform hyper-parameter tuning and the final performance of the best model will be given.

Some results analysis will be done, and some corrective actions to reduce the number of churners will be suggested.

In the conclusions, some ideas to improve the model performance will be given.

2 Data description

2.1 Initial dataframe overview

The analyzed dataframe has in total 286500 lines that were collected between the following start and end dates:
- First observation in the dataframe : Monday, October 01, 2018 12:01:57
- Last observation in the dataframe : Monday, December 03, 2018 01:11:16

We provide below the corresponding schema:

root
|– artist: string (nullable = true)
|– auth: string (nullable = true)
|– firstName: string (nullable = true)
|– gender: string (nullable = true)
|– itemInSession: long (nullable = true)
|– lastName: string (nullable = true)
|– length: double (nullable = true)
|– level: string (nullable = true)
|– location: string (nullable = true)
|– method: string (nullable = true)
|– page: string (nullable = true)
|– registration: long (nullable = true)
|– sessionId: long (nullable = true)
|– song: string (nullable = true)
|– status: long (nullable = true)
|– ts: long (nullable = true)
|– userAgent: string (nullable = true)
|– userId: string (nullable = true)

We also provide the first row to have a better understanding of these different fields:

Row(artist='Martha Tilston', auth='Logged In', firstName='Colin', gender='M', itemInSession=50, lastName='Freeman', length=277.89016, level='paid', location='Bakersfield, CA', method='PUT', page='NextSong', registration=1538173362000, sessionId=29, song='Rockpools', status=200, ts=1538352117000, userAgent='Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) Gecko/20100101 Firefox/31.0', userId='30')

There are different kinds of fields of interest in the dataframe:

- User specific fields :

  - firstName, lastName, gender, location,
  - userId : id of the user in the dataframe
  - userAgent : type of device used to interact with the application
  - registration: user registration timestamp
- Artists specific fields : artist, song, length (song length in seconds)
- level: subscription level (free / paid)
- auth: authentication level (Logged In / Logged Out / Cancelled / Guest)
- log-specific fields:
  - ts: timestamp of a given log
  - page: type of interaction with 22 categories (Home, Login, NextSong, Thumbs up, etc.)
  - sessionId: log session
  - itemInSession: log number in a given session
  - method: http request method (GET / PUT)
  - status: http status code (200 / 307 / 404)

2.2 Initial dataframe cleaning

No unexpected missing values were found in the dataframe. The only item to investigate was the user with empty userId field.

When we list the pages that this user visited, we see that there are only these sub-categories:

- 'About'
- 'Error'
- 'About'
- 'Error'
- 'Help'
- 'Home'
- 'Login'

- 'Register'
- 'Submit Registration'

No real usage have been made by this user (no songs). This seems to correspond to unregistered users. We can then drop this userId safely. After dropping this userId, we are left with 278154 lines.

2.3 Defining the churn event

If we look deeper in the page content, we see the following fields with the number of times they were used:

Cancel 52
Cancellation Confirmation 52
Submit Downgrade 63
Submit Upgrade 159
Error 252
Save Settings 310
About 495
Upgrade 499
Help 1454
Settings 1514
Downgrade 2055
Thumbs Down 2546
Logout 3226
Roll Advert 3933
Add Friend 4277
Add to Playlist 6526
Home 10082
Thumbs Up 12551
NextSong 228108 |

We will define the churn event by the 'Cancellation Confirmation' event. Looking at the data above, there are 52 users that churned in the dataframe.

We also found 173 loyal users, for a total of 225 users.

3 Consideration on the features to construct

3.1 Defining a statistical unit for our study

The goal of the study is to predict whether a user will churn. Since the decision is user-based, we should consider that the elementary unit of our study is a user. We should therefore construct a dataframe indexed by the users where each column is a feature describing how the user interacts with the application.

These features should represent e.g. a pattern of how the user listen to the music, add friends, add songs to a playlist, but also some external event out of control like error or advertising.

3.2 Considerations on the features

Notice that in order to avoid data leakage, we should make sure we do not use feature that takes advantage of an indirect knowledge of users who churned. Such features could be e.g. the total number of times an action (e.g. 'NextSong') is realized for a given user.

Actually, the users who churned stopped using the application earlier than loyal users, and thus mechanically make less interactions with it. However before they churn, i.e. at the time we want to make our prediction, we do not have this information.

In order to cope with such data leakage, we will make no usage of absolute quantities per user in the dataframe. Instead, we will consider quantities relative to some other quantity. We will consider two possibilities:

- A quantity per unit of time
- A quantity per visit to the application

We will thus need to define an observation window per user.

3.3 Defining an observation window per user

In order to define quantity per unit of time, we need to compute the duration over which a user is able to interact with the application. We will compute a start date and an end date using the 'registration' and 'Cancellation Confirmation' fields of the dataframe.

We have to take care of users having registered before the data collection start, and also of users who churned. We will thus compute the start and end of the user window as follows:

- start date :
  - for users who registered before the first date of the data frame, we consider the start of the dataframe
  - for users who registered after the first date of the data frame, we consider the registration time
- end date :
  - for users who did not churn, we consider the end of the dataframe
  - for users who churned, we consider the churn timestamps

4 Feature construction

4.1 Feature description

We use the previous considerations to construct features as quantities per unit of time and quantities per visit. We built the following features:

- Events that happened per hour : Thumbs Down per hour, Home per hour, Roll Advert per hour, Save Settings per hour, About per hour, NextSong per hour, Settings per hour, Add Friend per hour, Add to playlist per hour, Thumbs Up per hour, Help per hour, Error per hour, actions per hour
- Events that happened per visit : songs per visit, items per visit, hours per visit
- gender

Some of the features above are straightforward to understand and others deserve an explanation:
- actions per hour is the average number of interaction of any kind per hour a user has with the application
- items per visit is the average number of interaction of any kind per visit of the application
- hours per visit is the average duration per visit for a given user

4.2 Feature visualizations and selection

We provide below some visualization of the features to gain more insight on their relations and their interaction with the churn event.

4.2.1 Two-dimensional analysis

First, we provide the correlation matrix of the features, including the churn variable :

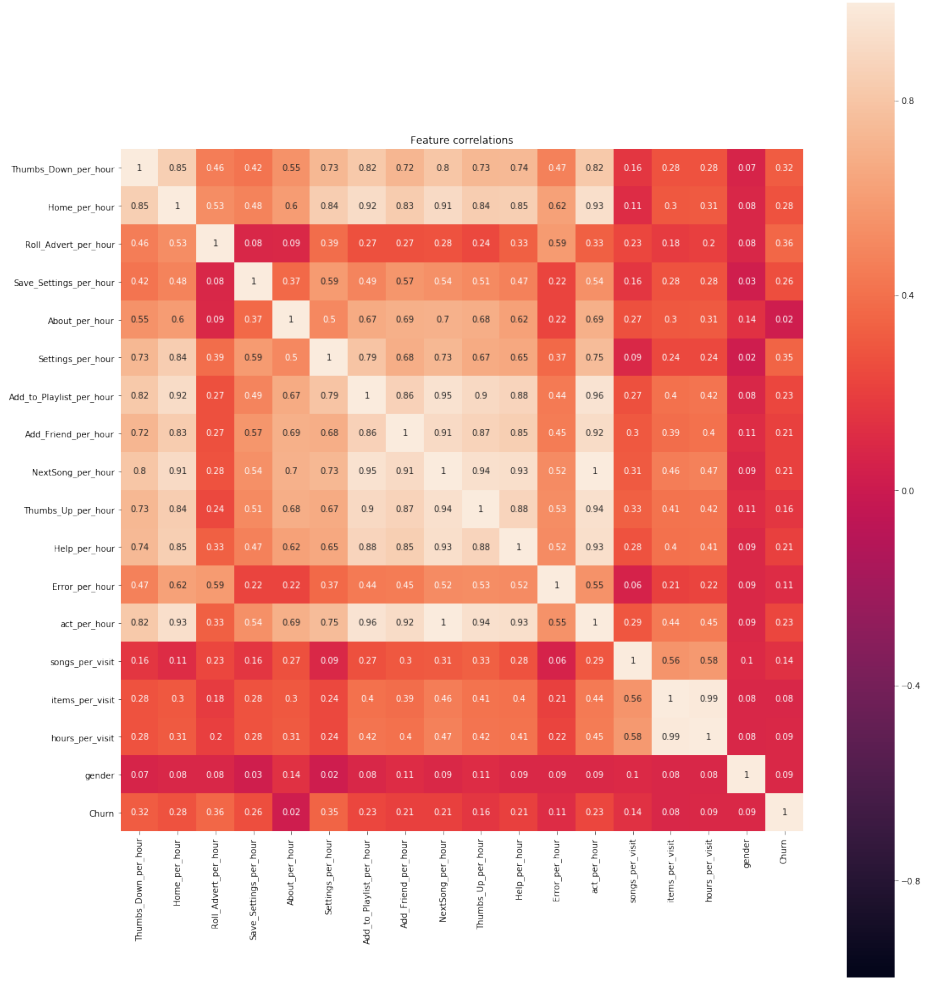

Figure 4.1: Features correlations

First, we notice that for many features we have a significant correlation with the variable we want to predict, which is a good indicator of good-quality features.

We notice some highly correlated features that deserves further multi-collinearity analysis. Based on this figure 4.1 and also on manual tuning, we finally decided to remove the following features: items per visit, actions per hour and Add to playlist per hour.

Notice that there is probably some room for improvement in this selection.

4.2.2 One-dimensional analysis

We provide below some plots of the selected features with a differentiation between churners and loyal users.

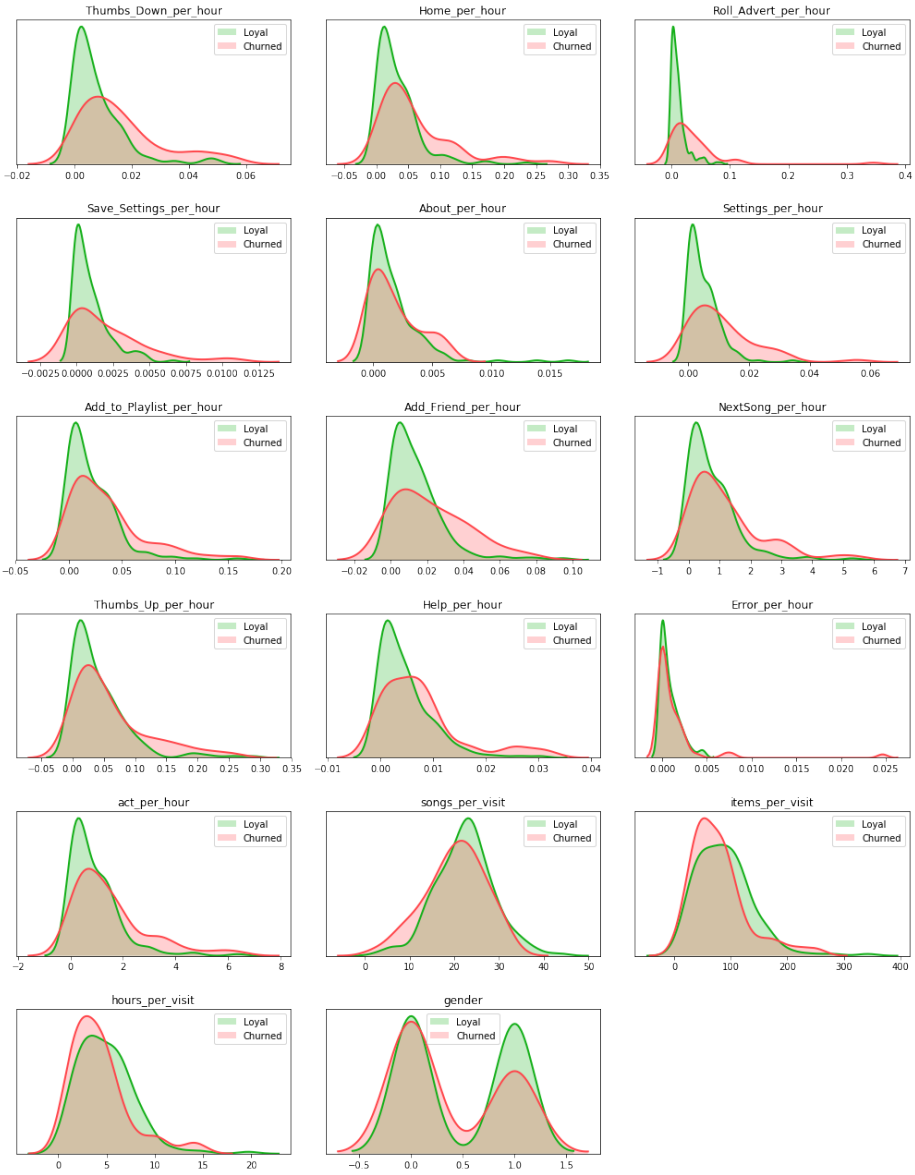

Figure 4.2: Probability density function of features for churners and loyal users

We notice that for almost all features, we see a clear difference of behavior between churner and loyal users, which is a good sign showing that these feature capture information about the variable we want to predict.

5 Models construction

5.1 Chosen models

We decided to compare three different models:

- Logistic regression classifier
- Random Forest classifier
- Gradient-boosted tree classifier

For each model, we first evaluate the performance using the model out of the box, and then we use grid-search to perform hyper-parameters tuning and obtain a better model.

5.2 Chosen metrics

In order to evaluate our models, we considered the following metrics:

- F1-score
- AUC
- Accuracy

The most important metric above is the F1-score, the others are given just for information. In particular, we used the F1-score as the metric to tune our models.

6 Results analysis

6.1 Model results comparison

We provide in the table below the results for the best model of each kind after grid-search.

Classifiers comparison			
Classifier	F1-Score	AUC	Accuracy
Logistic Regression	0.47	0.77	0.74
Random Forest	0.61	0.68	0.71
Gradient-Boosted Tree	0.64	0.6	0.65

Since we are interested in optimizing the F1-score, the best model is the Gradient-Boosted tree. Notice that the Random Forest also provides good results, so both models are interesting to analyze.

6.2 Results analysis and discussion

In the figure below, we provide the feature importances as given by both the Gradient-Booster tree and the random forest classifier:

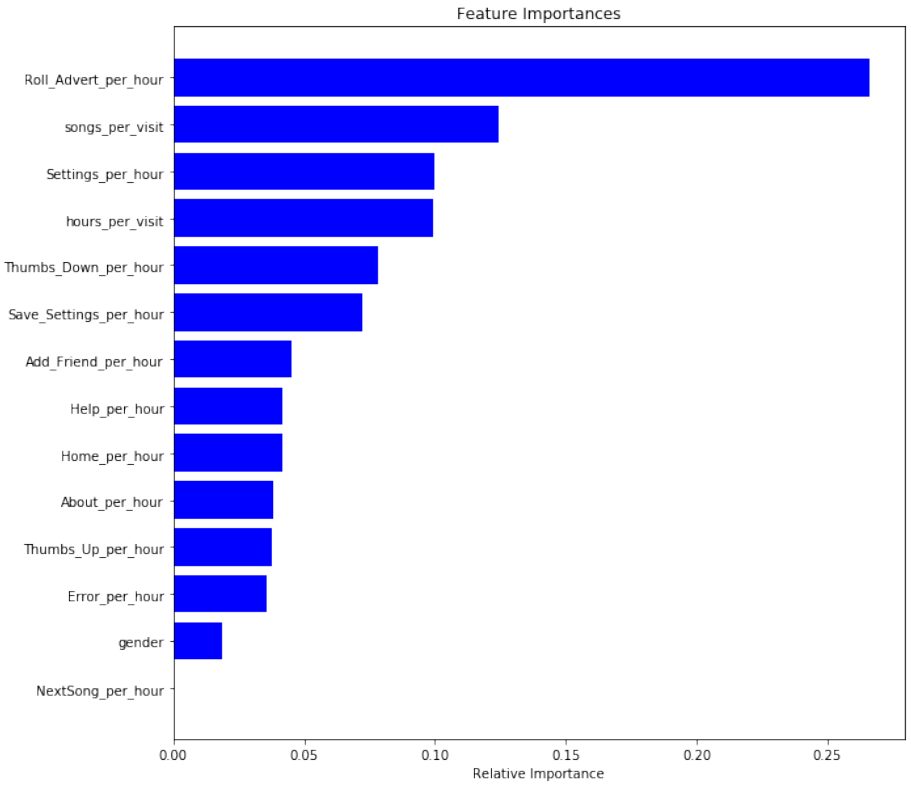

Figure 6.1: Gradient-Booster tree feature importance

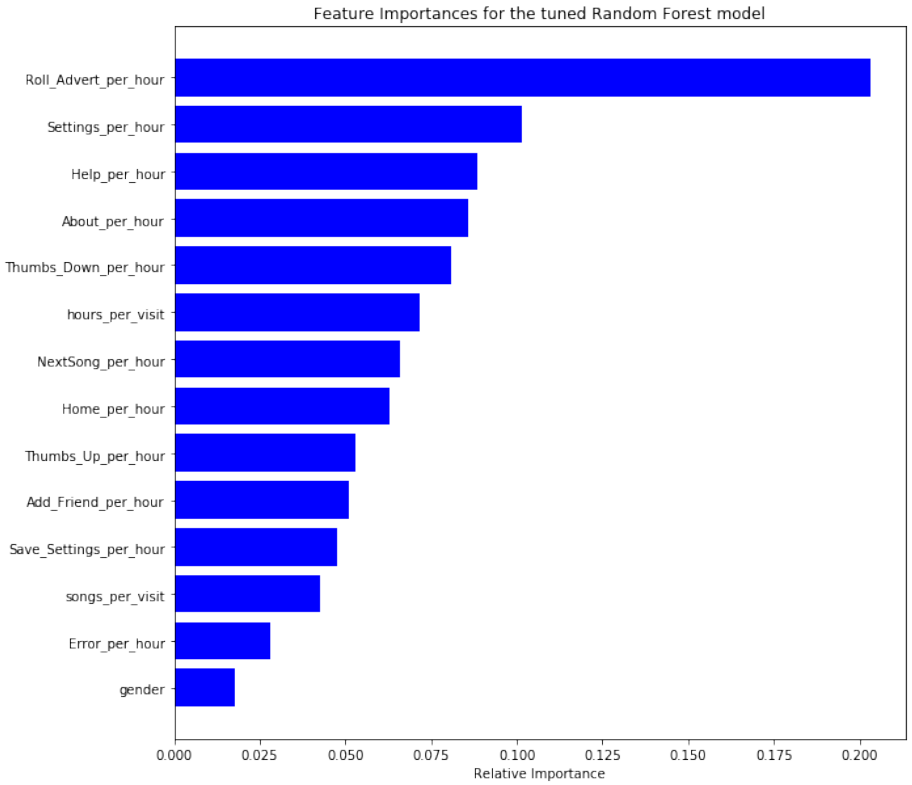

Figure 6.2: Random forest feature importance

These plots are very informative and provide some insight on why users are churning. If we analyze the features that are seen as important by both models and for which we can provide corrective actions, we can make the following comments:

- The most important feature for both models by far is the Roll advert per hour. This is easily interpretable and provide a simple way to reduce the churn through a decrease of the amount of advertising for users who are likely to churn. This reduction should be tuned as a tradeoff between churning decrease and revenue loss.

- Another important feature is the Settings per hour. This may show that the platform is not simple to use, and the user experience could be improved here. Some A/B testing could help to find some interesting evolutions. Maybe a more intuitive interface can be developed.

- A third feature of importance for both models is the Thumbs down per hour. This may indicate that churners are more often exposed to music they do not like. Some recommendation system can help here to provide better music. We could change the recommendations to users that are likely to churn.

- The other features are easily interpretable but do not provide simple corrective actions.

7 Conclusion and possible improvements

In this project we provided a model to predict the churner of a music application based on the logs of users usage. We constructed some features and perform modeling for various models. The best model reached an F1-score of 0.61, but there is some room for improvement.

We actually provide below ideas to have better results:

- Retrain models on the entire training set : in the modelling, after grid-search, we did not retrain the models on the full training set. Doing so could slighly improve the performance.
- Using the full data set of 12GB : The dataset used to train our models is only a sub-set (1%) of the full available dataset. Training on the full dataset could improve at two levels:
  - First, provide a more important number of users. Here we had only 225 users, which very small. Having more users could help models to be more robust.
  - Second, having more data could help to have more accurate features.
- Another improvement could result from the usage of other statistics on the measures instead of just the mean. We could use the standard deviation to distinguish between users having always the same behavior and users having high variations in their behavior
- We could also consider more features, here are some ideas:
  - Add other per-visit features just like the per-hour features : advertising, help, etc...
  - Group the type of devices into main families and use this. The device can have an impact on the interface and on the user experience.
  - Consider the inter-duration of the usage (like the inter-time duration between successive visits) and compute mean, variance, and other statistics on it.

All these improvements are quite interesting and implementable, this can be the basis of new models with better results.