Sparkify Churn prediction

Table of Contents

Project definition

Project overview

Sparkify is a digital music service, in which users are streaming their favorite songs either using free tier or using the premium subscription model, where they pay a monthly flat rate. All interactions of the users with the App (events) are logged which provides data to gather insights, learn behaviour patterns and explore customer funnels. Per each user we collect history of events, such as song plays, likes, dislikes, log in, log out, upgrade, downgrade, etc. Analyzing this data we can help business to understand key drivers of “making customer happy” and answer such questions as how to increase customer satisfaction, how to keep users more engaged, motivate them upgrade the subscription.

Problem statement

Users can upgrade, downgrade or cancel the service. If we can predict the user intention to cancel the service, we can try to target such users with special offers which will potentially prevent the churn and save business millions of dollars. In this project we’ve built the model which identifies users with high propensity to churn based on the history of their events in the App.

Metrics

The main challenge of “churn prediction” problem is the target imbalance. In the present dataset churn cases make 22.5% of all users, which put constraints on the metric to use for model learning and tuning. We use F1-score to run grid search for optimal hyper parameters, because F1-score is robust for imbalanced classes. For business-level explanation it’s also helpful to calculate recall for the best model, because it shows what portion of churn cases we actually loose.

Analysis

Data Exploration & Visualization

We run data exploration for a sample of data which includes 278154 records for 225 registered users and 8346 records for unknown users. Full dataset includes data for 22278 registered users, which we use later to train the models.

Here is the short summary on the sample data:

2354	**63**	**17655**	**58480**	**22**
sessions	days	unique artists	unique songs	different page events

Main observations

1. There are 225 registered users in the dataset and 2354 sessions during 63 days. 97% of records cover the events for these users and only 3% include the data about the guests.
2. For guest users (auth='Guest') we don't have neiver songs data or user demographics data, nor *userId*. Their page visits are limited to: Home, Help, Register, About, Submit Registration, Error. We exclude guest users from model dataset.
3. There are 3% of records with auth='Logged Out', which include Home, Login, About, Help and Error events. There is no *userId* data for these events, so we exclude them from modelling dataset.
4. Only 20% of users belong to free tier, 80% are using paid subscription (Figure 1).

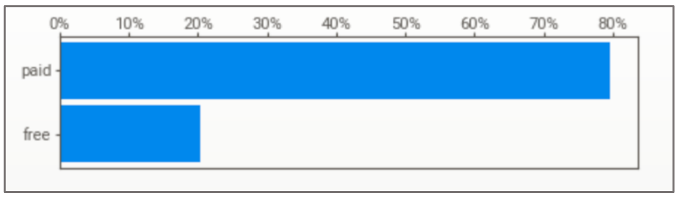

Figure 1 Distribution of paid vs free tier of users in Sparkify

5. 80% of records describe NextSong event and include artist and song data. 20% of events cover all over possible actions (Figure 2).

6. We have 52 cancellation events, which are described by auth='Cancelled' and page='Cancellation Confirmation'. There are 52 unique *userId*, who cancelled subscription. So this event is unique per user.

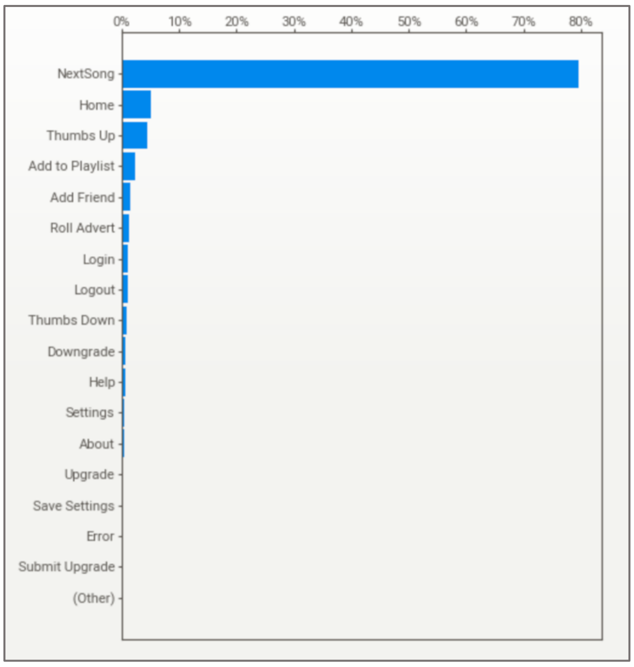

Figure 2 Distribution of different page events in Sparkify dataset sample

Define the target

We **define Churn** as a fact of cancellation of subscription from existing user. The fact of cancellation is translated through 2 columns: auth='Cancelled' and page='Cancellation Confirmation', which are uniquely defined, so we can use any of 2 to define the target. Let's use page='Cancellation Confirmation' as our target. We also noticed that there is event page='Cancel', which is coming prior to 'Cancellation Confirmation' , so we should exclude this event from the train dataset to avoid data leakage.

Compare churn users with active users

1. Among those who churn there are more males (57% M / 43% F), and vice versa, there are more women among those users, who stay subscrbed (42% M / 58% F).

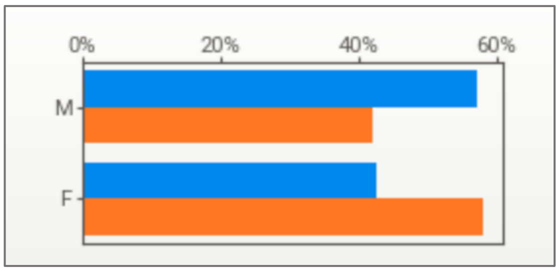

Figure 3 Gender distribution among churned (blue) and active (orange) users

2. Churned users usually have smaller number of items in session (churn median = 66 VS other median = 71).
3. Among churned users there are more free users (28% in churn VS 19% in other).

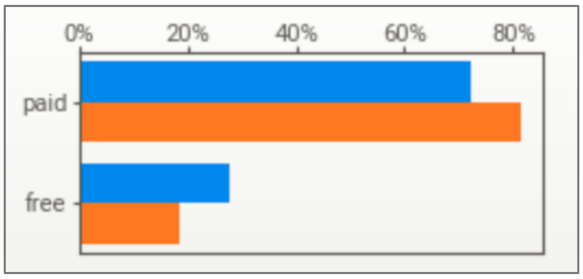

Figure 4 Tier distribution among churned (blue) and active (orange) users

4. From page events statistics we see that among churned there are less *Thumbs Up*, more *Thumbs Down*, almost two times higher frequency of *Roll Advert*.
5. Churned users have smaller lifetime.

Modelling approach

Feature engineering

Our original dataset consists of multiple records (events) per each user. Using this data we derive features at user-level, so that in train/test data we have only one row for each unique *userId*.

Step 1: aggregate features at user level

1. Exclude records with empty *userId*.
2. Add label: 1 = Churn, 0 = Not churn. Condition: page='Cancellation Confirmation'
3. Remove records of page='Cancellation Confirmation' and page='Cancel'.
4. Sort dataframe by userId and ts
5. Aggregate features at user level:
   - create list of songs
   - create list of artists
   - list of page events (Cancellation Confirmation and Cancel events preliminary filtered out to remove the leak)
   - session frequency: number of sessions/(last session time - first session time)
   - average number of songs per session: total number of songs/total number of sessions
   - binary feature: Male gender = 1/0
   - binary feature: paid acoount = 1/0
   - lifetime (days): time difference between last activity and registration date
   - derive OS/device features from agent values

Step 2: Apply TF-IDF

Apply TF-IDF transformation to lists: songs, artists, page events. For songs and artists we keep only top-100 values, page events are all included (since only 22 values in total).

Feature engineering is implemented with *pyspark.ml.feature* and incorporated as a stage into modelling pipeline.

Models

Churn prediction is a binary classification problem, so we considered the classifiers available in *pyspark.ml.classification*. Since most of the features are derived from categorical data (songs, artists, pages) we decided to use tree-based models. We focused on comparison of 2 algorithms:

- Random Forest Classifier
- Gradient-boosted trees Classifier

Both models were feed with the same data (70% train and 30% test records got by random split). We used grid search to find better hyperparameters.

Grid search set up:

- Random Forest Classifier:
  - Maximum tree depth: [5, 7]
  - Number of trees: [50, 100]

- Gradient-boosted trees Classifier
  - Maximum tree depth: [3, 5]
  - Number of iterations: [5, 10]

3-fold cross validation was used to measure the average F1-score.

Results

We compare 2 optimized with grid search models by F1-score and Recall.

Random forest	**Gradient-boosted trees**
Optimal hyper-parameters	
- Maximum tree depth: 7 - Number of trees: 50	- Maximum tree depth: 5 - Number of iterations: 10
F1-score	
0.78	0.86
Recall	
0.82	0.87

Both models show quite good scores (recall > 80% and f1-score > 75%). Since cross-validation with full dataset is a time-consuming procedure (> 1 hour to run) we've checked limited number of parameter combinations. We see that the best GBT model was got with highest options available for tree depth and number of iterations, thus we suppose that increasing these 2 parameters can give even higher model quality.

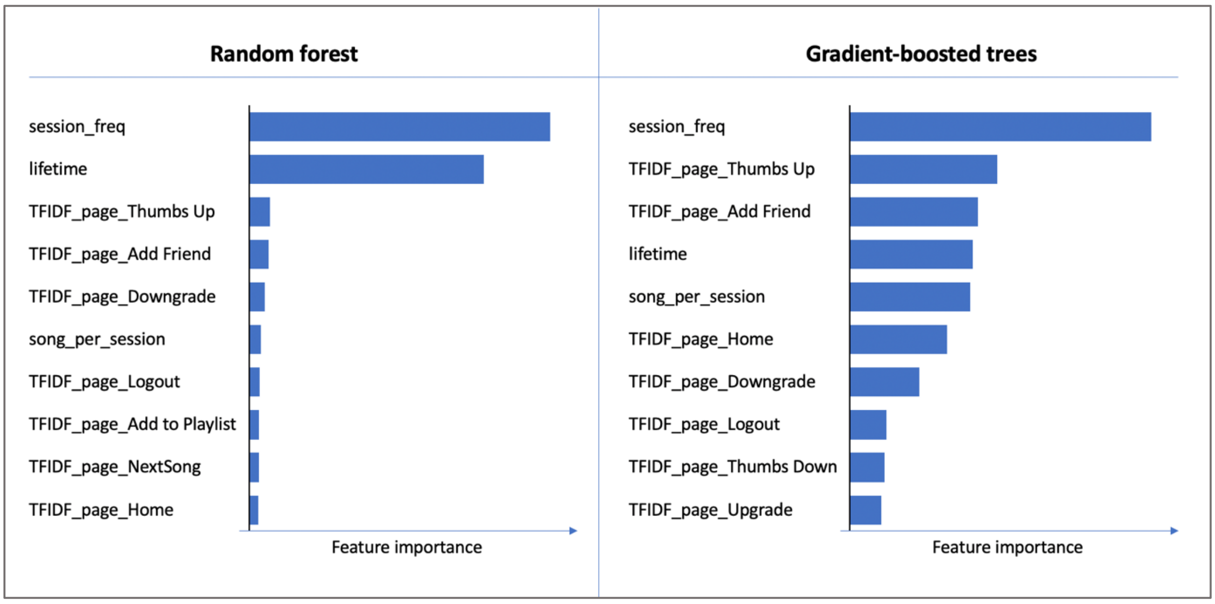

Figure 5 Top-10 most important features for Random forest and Gradient-boosted trees classifiers

We can clearly see that frequency of sessions is dominant factor in both models. The other important feature is lifetime. RF model is deprioritizing all over features in comparison with those 2, for GBT we see quite high significance for page events, which is good, because session frequency and lifetime are misleading factors for new users (due to low number of observations). We can also notice that most important are page events which are connected to user loyalty/satisfaction: e.g. Thumbs Up/Down, Add Friend, Upgrade/Downgrade.

Conclusion

Gradient-boosted trees demonstrated better quality even with low tree depth and number of iterations.

Here are some ideas of potential improvements for the model:

1. Run grid search for GBT with higher number of iterations and tree depth.
2. Include into grid search vocabulary size coming from feature pipeline.
3. Add geo-location features (were omitted for this dataset).
4. Balance dataset: if amount of data is enough, we can sample similar number of active (target = 0) and churn (target = 1) users.
5. Make cross-validation and train-test split time-dependent. Since we are going to learn model always with past data and predict churn with data observed next days-weeks,

we can change the approach to the dataset collection from user-level to user+date-level. For example, we can aggregate data by userId & week. While splitting the data by train/test, we should avoid shuffling and split it past/future and similar in cross-validation, we should apply time-series cross-validation.

Here are some ideas of business applications:

1. We can use 2 types of model outcomes: prediction 1/0 and predicted probability for outcome = 1. Using predicted probability we can segment potential churn as low-medium-high (choosing threshold from distribution) and propose to them different offers/discounts.
2. We should be mindful of new users. Usually, due to the low amount of events observed for new users, model prediction is less accurate. We would propose to start using the model for users who are with us at least N days (this N should be business-driven and depends on the funnel and retention). For new users retention strategy should be different.
3. It will be extremely useful to use churn prediction model together with LTV prediction model to select relevant offer.

References

1. GitHub repository: https://github.com/DariaSatco/SparkifyChurnPrediction