Music Service Data Analysis with Spark

UDACITY DSND PROJECT REPORT

BY ALEKSANDRA DEIS

Moscow, 2019

Contents

Definition

Project Overview

Predicting churn rates is a challenging and common problem that data scientists and analysts regularly encounter in any customer-facing business. It is crucial for businesses to identify customers who are about to churn and take action to retain them before it happens.

The goal of this project was to help Sparkify music service retain their customers. In this project, I analyzed Sparkify data, built a machine learning model to predict churn and developed a web application to demonstrate the results.

Problem Statement

As the goal of the project is to help to retain the customers, the main task of the project is to make a prediction, whether the customer is about to churn. Such a prediction can be made for each customer by a binary classifier model. The following tasks should be completed to create the model:

- Analyze and preprocess the data to extract features for each customer;
- Train classifier to predict customer churn;
- Evaluate the classifier concerning the chosen metric;
- Build a web application to demonstrate the results.

Metrics

The initial dataset analysis shows us that the dataset is imbalanced (see section Input Data

As input data I have several datasets, which contain the log of Sparkify music service events:

- medium_sparkify_event_data.json – medium sized dataset.
- mini_sparkify_event_data.json – a tiny subset of the full dataset, which is useful for preliminary data analysis.

Both files contain the following data:

#	Column Name	Type	Description
1	userId	string	Unique identifier of the user, the event is related to
2	artist	string	Name of the artist related to the song related to the event
3	auth	string	“Logged in” or “Cancelled”
4	firstName	string	First name of the user
5	gender	string	Gender of the user, “F” or “M”

#	Column Name	Type	Description
6	itemInSession	bigint	Item in session
7	lastName	string	Last name of the user
8	length	double	Length of the song related to the event
9	level	string	Level of the user's subscription, "free" or "paid". User can change the level, so events for the same user can have different levels
10	location	string	Location of the user at the time of the event
11	method	string	"GET" or "PUT"
12	page	string	Type of action: "NextSong", "Login", "Thumbs Up" etc.
13	registration	bigint	Registration
14	sessionId	bigint	Session id
15	song	string	Name of the song related to the event
16	status	bigint	Response status: 200, 404, 307
17	ts	bigint	Timestamp of the event
18	userAgent	string	Agent, which user used for the event, for example, "Mozilla/5.0"

Data Exploration): there are more than 3 times fewer users, who churned, than other users. That is why I can't use accuracy (which is the number of correct predictions divided by the total number of predictions) as a metric to evaluate the resulting model.

In our case, we should care about both types of errors: false negatives and false positives because in case of false negative we can miss the customer who is going to churn and lose the customer and in case of false positive we can have unnecessary costs on retaining the customer who was not going to churn. That is why as a metric to evaluate the model I chose F1 score because it equally considers both the precision and the recall.

Analysis

Input Data

As input data I have several datasets, which contain the log of Sparkify music service events:

- medium_sparkify_event_data.json – medium sized dataset.
- mini_sparkify_event_data.json – a tiny subset of the full dataset, which is useful for preliminary data analysis.

Both files contain the following data:

#	Column Name	Type	Description
1	userId	string	Unique identifier of the user, the event is related to
2	artist	string	Name of the artist related to the song related to the event
3	auth	string	“Logged in” or “Cancelled”
4	firstName	string	First name of the user
5	gender	string	Gender of the user, “F” or “M”
6	itemInSession	bigint	Item in session
7	lastName	string	Last name of the user
8	length	double	Length of the song related to the event
9	level	string	Level of the user’s subscription, “free” or “paid”. User can change the level, so events for the same user can have different levels
10	location	string	Location of the user at the time of the event
11	method	string	“GET” or “PUT”
12	page	string	Type of action: “NextSong”, “Login”, “Thumbs Up” etc.
13	registration	bigint	Registration
14	sessionId	bigint	Session id
15	song	string	Name of the song related to the event
16	status	bigint	Response status: 200, 404, 307
17	ts	bigint	Timestamp of the event
18	userAgent	string	Agent, which user used for the event, for example, “Mozilla/5.0”

Data Exploration

I used **mini_sparkify_event_data.json**, which is a tiny subset of the full dataset for data exploration.

Explore Missing Values

First of all, I analyzed the number of missing values in each column of the dataset:

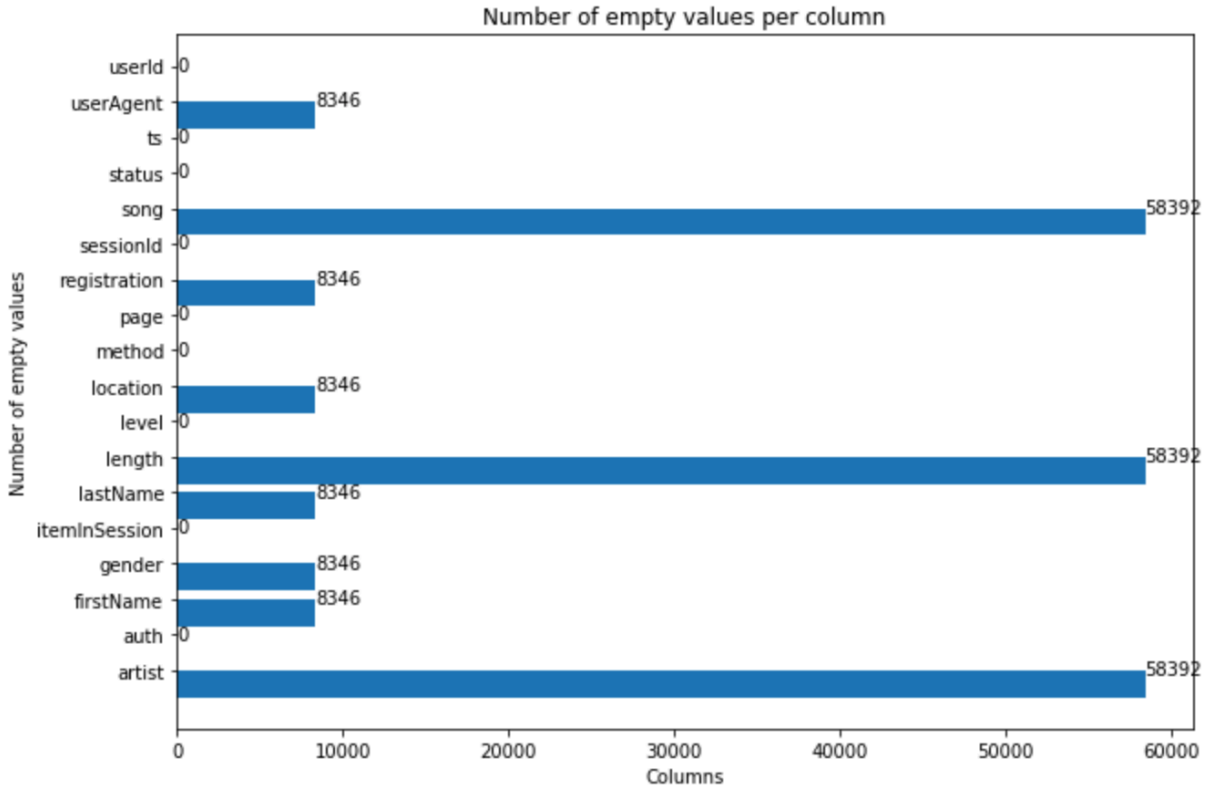


Figure 1 Number of missing values per column

Figure 1 above shows the number of empty values per column. We see that there are equal numbers of missing values for different sets of columns. Looking at the rows of the dataset we see that:

1. There are 8346 rows where userId is an empty string. In these rows other columns are missing:

```
+------+-----+---------+------------+--------+---------+--------+------+----+------+------+
|userId| page|userAgent|registration|location|firstName|lastName|gender|song|length|artist|
+------+-----+---------+------------+--------+---------+--------+------+----+------+------+
|      | Home|     null|        null|    null|     null|    null|  null|null|  null|  null|
|      | Help|     null|        null|    null|     null|    null|  null|null|  null|  null|
|      | Home|     null|        null|    null|     null|    null|  null|null|  null|  null|
|      |Login|     null|        null|    null|     null|    null|  null|null|  null|  null|
+------+-----+---------+------------+--------+---------+--------+------+----+------+------+
```

   These events correspond to the activity of the users, who are not logged in. These records can't help us to predict customer churn, because we can't identify the customer, that is why we should remove these rows from the analysis.

2. Columns song, artist and length aren't missing only for rows where page value is "NextSong".

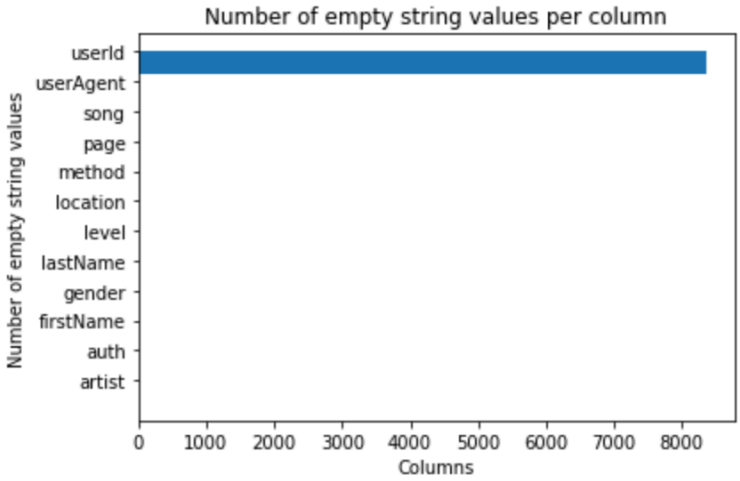


Figure 2 Number of empty string values per column

Figure 2 above shows that only column userId has empty string values. As mentioned above rows with empty userId should be removed during data cleaning stage.

In all further exploration I assume that rows with empty userId are already removed from the analysis.

Preliminary Analysis

After removing rows with empty userId from the dataset we have:

- **278154 lines**, each of them corresponds to an event for a user, who signed in;
- **225 unique users**.

Figure 3 below shows the values of categorical columns auth, gender, level, method, and page for events on the dataset:

1. The number of events for female users is slightly higher than for male users. However, this may be true only for our small dataset which is the subset of the full dataset.
2. The number of events for paid level accounts is approximately 4 times greater than the number of events for free level accounts.
3. The most frequent action is 'NextSong' which is not surprising for a music streaming service.

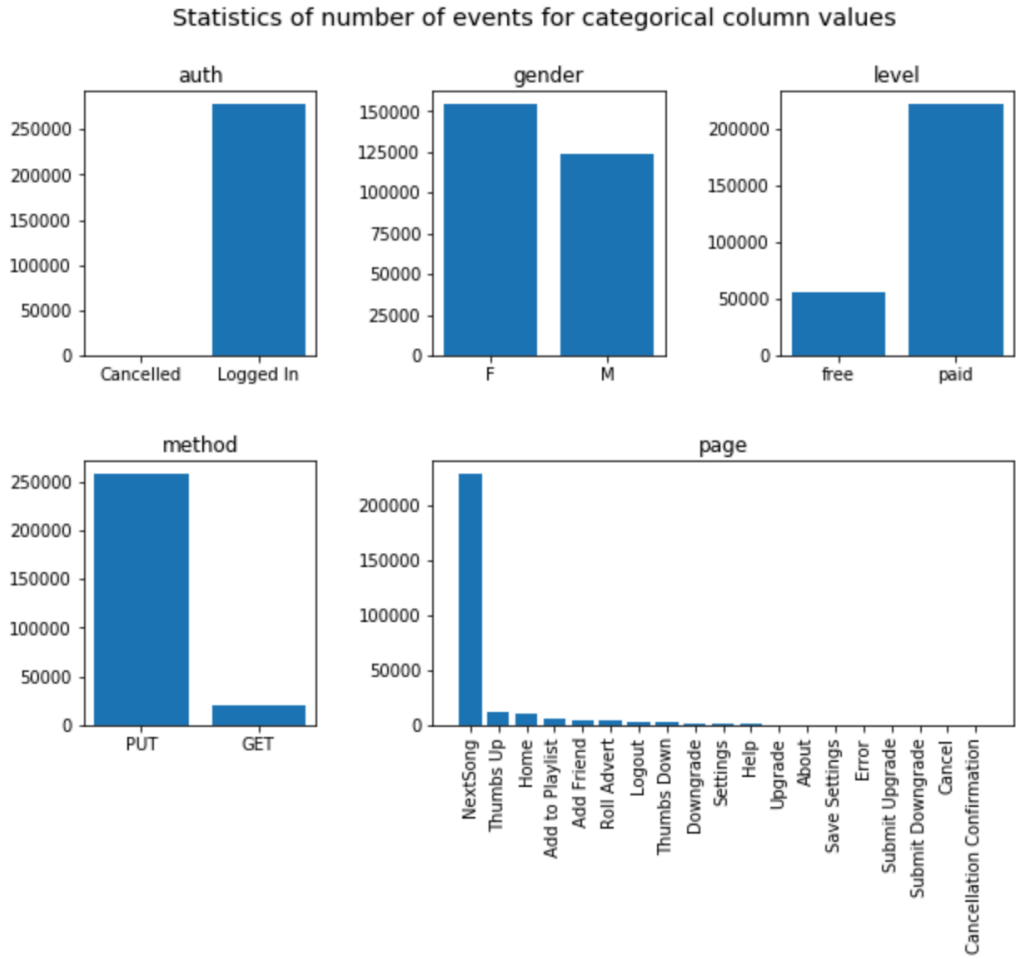


Figure 3 Categorical column values

Figure 4 below illustrates the number of female and male users, paid and free accounts:

There are slightly more male users in the mini dataset. However, comparing with the previous plot where we saw more events for female users, so female users more actively use the service than male users.

There are slightly more free accounts in the mini dataset, but there are more events for paid users. We can conclude that users with paid accounts are more active using the service.

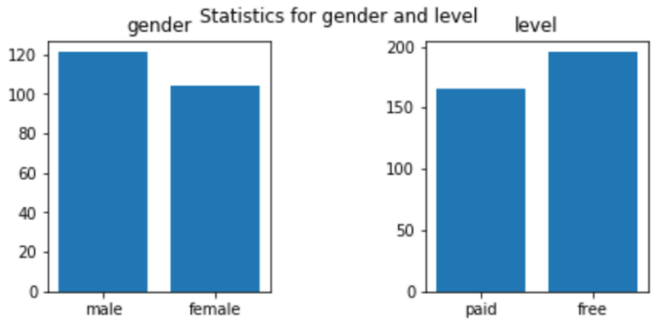


Figure 4 Statistics for users' level and gender

Definition of Churn

The next step is to define churn given the data from the input dataset. Among the values in the page column, there is “Cancellation Confirmation”, which means that the user has confirmed cancellation of his subscription. This event marks that the user churned.

To complete the data exploration and to analyze the behavior of different types of users the data is processed in the following way:

1. Created new column “churn”;
2. “churn” value is set to 1 for rows where “page” value is “Cancellation Confirmation,” otherwise set “churn” to 0;
3. “churn” value is set to 1 for all rows for a user, who has rows where “churn” is equal to 1.

As a result, “churn” value is equal to 1 for all events of all users who canceled their subscription.

Analysis of Behavior of the Users Who Churned

After churn is defined, it is possible to analyze the behavior of the users who churned versus the users who didn’t cancel their subscription.

In the mini dataset there are:

- **52** users who churned;
- **173** users who didn’t cancel their subscription.

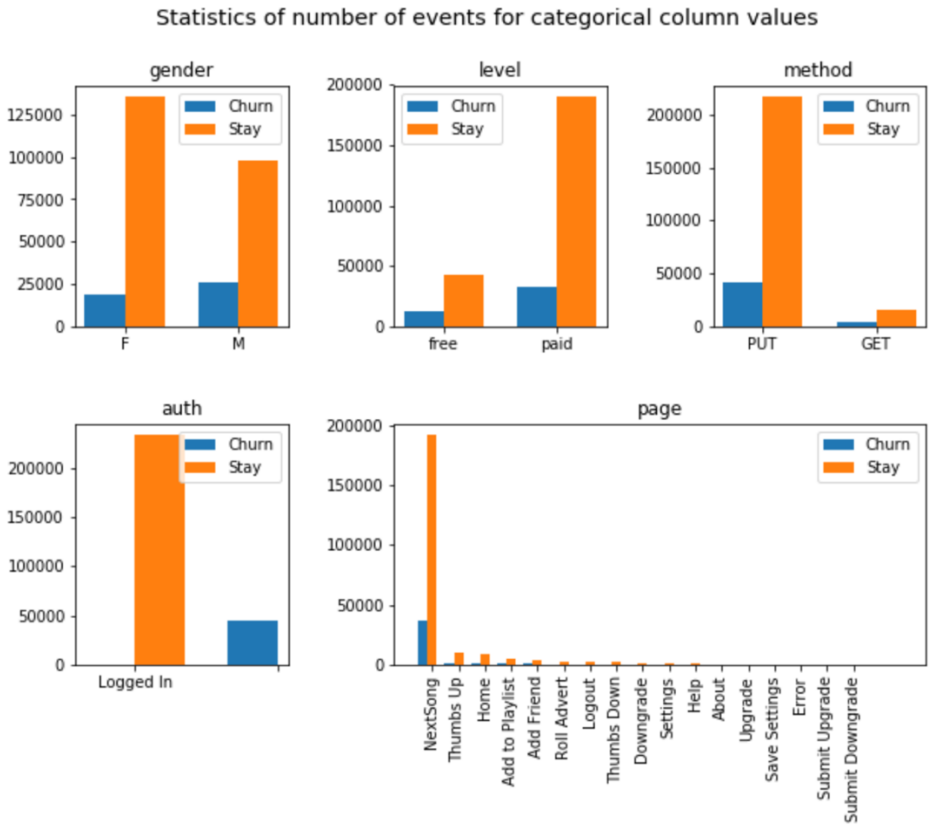


Figure 5 Number of events for users who churned vs other users

Figure 5 shows the number of events for users who churned versus other users concerning different categorical columns. We can make the following conclusions:

- Even though there are more events for female than male users, the number of events for churned male users is higher than for churned female users. That is why we can suppose that females who churned are less active service users than those who stay with their subscription.
- For all other categorical features, we see that there are more events for specific value there are in the dataset, the more events for this specific value correspond to churned users. So, we can't make any conclusion out of that regarding the behavior of churned users.

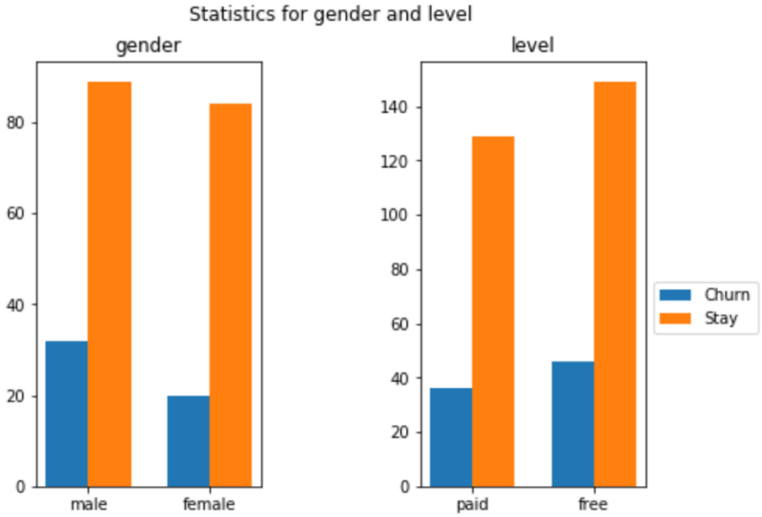


Figure 6 Number of male/female users, number of paid/free accounts

In Figure 6 there is an exploration of the number of male and female users, and the number of paid and free accounts:

- There are fewer female users than male users and also a fewer number of female users who churned.
- Percentage of male users who churned is higher than the percentage of female users, who churned.
- Percentage of paid accounts related to users who churned is almost the same as the percentage of free accounts related to users who churned.

In Figure 7 there is the distribution of the events among states.

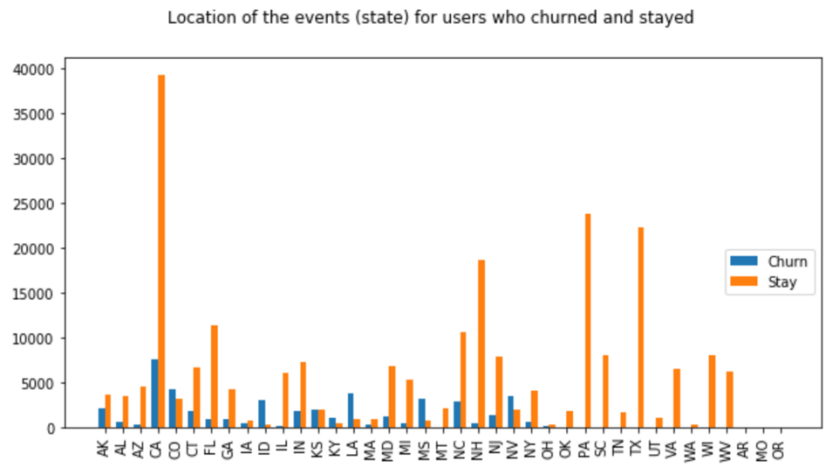


Figure 7 Location of the events

Data Exploration Conclusions

1. From the exploratory analysis above we learn that the behavior of female users is different compared to the behavior of male users:
    a. Female users are more active because there are fewer female users, but there are more events related to female users.
    b. Percentage of male users who churned is higher than the percentage of female users, who churned, so male users may be more likely to churn.
2. Female users who churned have different behavior rather than other female users: female users who churned are less active.

Methodology

Data Preprocessing

Before the implementation of machine learning methods, the initial dataset requires preprocessing:

1. Remove the rows with empty userId values (see section Explore Missing Values).
2. Define churn as in section Definition of Churn above.
3. Create a new dataset, which contains features extracted for each user labeled with churn values.

To predict whether the user is going to churn for each user the following features should be extracted for each user:

- **gender**, because data exploration showed that male and female users may have different behavior;
- **last level** (paid or free), because data exploration showed that paid level users are more active and also may have different behavior;
- **average events per day** to see how active the user is in general;
- **average songs per day** to see how active the user is in listening to the music;
- **number of thumbs up** to see how satisfied the user is with the content of the service;

- **a number of thumbs down** to see how satisfied the user is with the content of the service;
- **days since the date of the first event** to see how long the user has already been using the service;
- **last location** (postal abbreviation of the state): the popularity may vary depending on advertising in different regions;
- **number of add friend events**: perhaps the user is less likely to churn if his friends also subscribe to the service.

The preprocessing was done by PySpark. The final dataset prepared for machine learning looks as follows:

userId	gender	churn	last_level	days_active	last_state	avg_songs	avg_events	thumbs_up	thumbs_down	addfriend
89	M	0	paid	178	GA	111.17	133.5	30	6	14
148	M	0	free	181	FL	36.18	47.09	28	3	7
200015	M	1	free	184	CA	28.67	34.9	10	10	2
93	M	0	paid	186	NJ	58.18	74.09	35	4	13
300004	F	0	free	171	CA	68.0	87.33	17	2	2
114	M	0	paid	184	NV	76.0	85.94	74	12	13
200024	M	1	paid	183	KY	37.91	50.18	19	13	9
81	M	0	paid	186	MD	94.0	111.19	94	14	23
38	M	0	paid	186	PA	69.58	78.5	65	21	21
100024	M	1	free	183	PA	22.0	36.0	5	0	0
122	F	1	paid	183	AR	13.67	15.5	3	1	1
40	F	0	paid	186	TX	56.74	69.58	66	11	23
100008	F	0	free	178	CA	96.5	117.5	37	6	17
58	M	1	paid	186	AL	121.0	144.79	94	20	19
123	M	0	free	161	FL	21.43	28.0	5	1	5

The following Figure 8 illustrates the distribution of newly generated features:

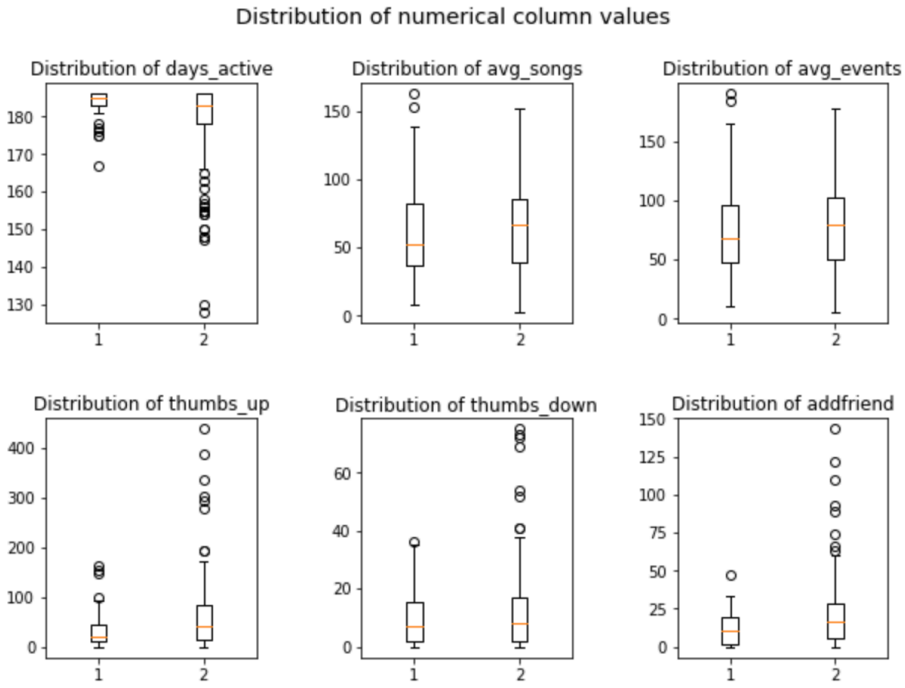


Figure 8 Distribution of numerical column values (1 – users who churned, 2 – users who stayed)

There is a difference in the distributions of those features between customers who churned and the others:

- The mean values of the average number of events and songs per day are lower for the customers who churned.
- The mean values of thumbs up, thumbs down and add friend events are lower for the customers who churned.

Stated above may mean that customers who churned are less active users of the service.

Implementation

The input datasets contain massive amounts of data, which can't be processed on a single machine. That is why I will use Spark clusters to analyze data and predict customer churn. I will use PySpark and SparkML libraries to implement the solution.

The implementation of the project consists of two parts:

- Application of machine learning methods to predict churn. This part involves creation of machine learning pipelines, evaluation and tuning of the approach.
- Development of a web application to demonstrate the resulting model.

Machine Learning Pipelines

Construction of a pipeline is required to apply machine learning methods for churn prediction. SparkML library was used to apply machine learning.

Figure 9 Machine learning pipeline steps

Machine learning pipeline for our task consists of the following steps (see Figure 9):

1. Split dataset into train, test, and validation.
2. Create dummy columns out of categorical columns 'gender', 'last_level', and 'last_state'. When using pyspark machine learning library sparkml, this step actually consists of two parts: indexing categorical column and encoding it.
3. Create a feature vector.
4. Train the classifier.

There are several classification algorithms from SparkML, which were used to build the final model for the customer churn prediction:

- Random Forest Classifier;
- Logistic Regression Classifier;
- Gradient-Boosted Tree Classifier;
- Naïve Bayes Classifier.

See the evaluation of metrics of models based on those algorithms in The web application allows the user to enter the information about the customer and then tells whether the customer is about to churn based on this information.

Refinement and

To increase those scores, we can take the following refinement measures:

- Add preprocessing to the pipeline: add scaling or normalization to preprocess numeric features.
- Refine last_state column values: introduce categorical value 'other' for rare states.

Scaling and Normalization

The F1 score of pipelines with **preprocessing** is as follows:

Classifier	Scaling (MinMaxScaler)		Normalization (Normalizer)	
	Train Score	Test Score	Train Score	Test Score
Random Forest	0.84	0.78	0.80	0.80
Logistic Regression	0.78	0.80	0.78	0.80
Gradient-boosted Tree	0.98	0.74	0.92	0.78
Naive Bayes	0.79	0.80	0.78	0.80

The table above shows that preprocessing increases slightly F1 scores of Random Forest and Gradient-boosted Tree algorithms and dramatically increases scores of Naïve Bayes algorithm.

Refine Location Feature

There are over 45 distinct values for the state in the initial dataset. Over 30 distinct values for last_state column remain after the preprocessing. That is why it is a common situation that the model has to predict for the unseen last_state value. In this case, the model won't perform well unless we introduce a new value 'OTHER' instead of rare states. After that, the model won't have to make predictions on unseen data anymore.

The table below shows F1 scores after refining the location feature:

Classifier	Train Score	Test Score
Random Forest	0.82	0.79
Logistic Regression	0.80	0.79
Gradient-boosted Tree	0.95	0.73
Naive Bayes	0.67	0.61

We can see that refining the location feature helps to improve scores for some algorithms.

Combine Refinement Approaches

The table below shows F1 scores for both refining location feature and normalization of numerical columns:

Classifier	Train Score	Test Score

Random Forest	0.83	0.81
Logistic Regression	0.80	0.79
Gradient-boosted Tree	0.93	0.75
Naive Bayes	0.80	0.79

As we see above the most effective measures for model refinement are normalization of numeric features and the additional preprocessing of location feature.

In conclusion I can say that the combination of the refinement measures taken above helped to **improve the final scores by 2-3%.**

Model Evaluation sections.

Web Application

The resulting web application is implemented with:

- Flask running the back-end,
- Bootstrap controls of front-end.

The web application consists of the following parts:

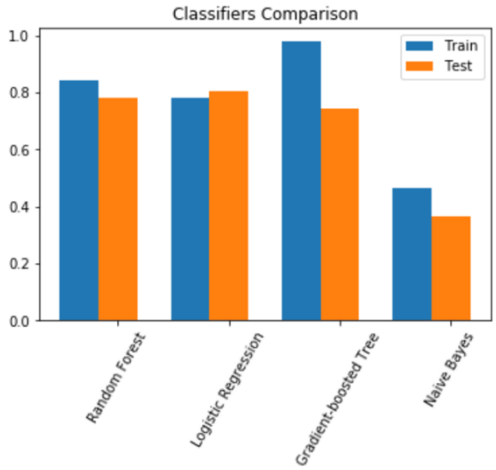
- Python script create_model.py which builds the machine learning model. This script accepts the path to the dataset and the path where the resulting model should be saved as parameters.
- The machine learning model, which is created by script create_model.py. The application loads the model and uses it to make predictions.
- Python script run.py, which runs the logic of the application and renders web pages. The script loads the model on start and applies it to make predictions out of the data provided by the user on the web page.
- Web page templates master.html and go.html of application web pages. Pages use bootstrap controls.

The web application allows the user to enter the information about the customer and then tells whether the customer is about to churn based on this information.

Refinement

The F1 score of pipelines for the initial approach is as follows:

Classifier	Train Score	Test Score
Random Forest	0.84	0.78
Logistic Regression	0.78	0.80
Gradient-boosted Tree	0.98	0.74



Naive Bayes	0.46	0.36

To increase those scores, we can take the following refinement measures:

- Add preprocessing to the pipeline: add scaling or normalization to preprocess numeric features.
- Refine last_state column values: introduce categorical value 'other' for rare states.

Scaling and Normalization

The F1 score of pipelines with **preprocessing** is as follows:

Classifier	**Scaling (MinMaxScaler)**		**Normalization (Normalizer)**	
	Train Score	**Test Score**	**Train Score**	**Test Score**
Random Forest	0.84	0.78	0.80	0.80
Logistic Regression	0.78	0.80	0.78	0.80
Gradient-boosted Tree	0.98	0.74	0.92	0.78
Naive Bayes	0.79	0.80	0.78	0.80

The table above shows that preprocessing increases slightly F1 scores of Random Forest and Gradient-boosted Tree algorithms and dramatically increases scores of Naïve Bayes algorithm.

Refine Location Feature

There are over 45 distinct values for the state in the initial dataset. Over 30 distinct values for last_state column remain after the preprocessing. That is why it is a common situation that the model has to predict for the unseen last_state value. In this case, the model won't perform well unless we introduce a new value 'OTHER' instead of rare states. After that, the model won't have to make predictions on unseen data anymore.

The table below shows F1 scores after refining the location feature:

Classifier	**Train Score**	**Test Score**
Random Forest	0.82	0.79
Logistic Regression	0.80	0.79
Gradient-boosted Tree	0.95	0.73
Naive Bayes	0.67	0.61

We can see that refining the location feature helps to improve scores for some algorithms.

Combine Refinement Approaches

The table below shows F1 scores for both refining location feature and normalization of numerical columns:

Classifier	Train Score	Test Score
Random Forest	0.83	0.81
Logistic Regression	0.80	0.79
Gradient-boosted Tree	0.93	0.75
Naive Bayes	0.80	0.79

As we see above the most effective measures for model refinement are normalization of numeric features and the additional preprocessing of location feature.

In conclusion I can say that the combination of the refinement measures taken above helped to **improve the final scores by 2-3%.**

Model Evaluation

To predict customer churn there was built a machine learning pipeline which consists of:

- the indexers and an encoder for categorical features,
- the feature assembler,
- the normalizer for numeric features,
- the Random Forest Classifier.

The Random Forest Classifier was chosen from the set of other models because it demonstrated the best performance in terms of the F1 score (81%).

To tune parameters for the Random Forest algorithm I applied cross validation. F1 scores for different parameter sets are demonstrated in the table below:

f1	featureSubsetStrategy	impurity	maxDepth	numTrees
0.692592	auto	gini	5	100
0.692592	sqrt	gini	5	100
0.692592	log2	gini	5	100
0.690154	auto	gini	5	50
0.690154	sqrt	gini	5	50
0.690154	log2	gini	5	50
0.687989	auto	entropy	5	10
0.687989	sqrt	entropy	5	10

f1	featureSubsetStrategy	impurity	maxDepth	numTrees
0.687989	log2	entropy	5	10
0.684809	auto	gini	5	10
0.684809	sqrt	gini	5	10
0.684809	log2	gini	5	10
0.678648	auto	entropy	3	50
0.678648	sqrt	entropy	3	50
0.678648	log2	entropy	3	50
0.676230	auto	entropy	5	50
0.676230	sqrt	entropy	5	50
0.676230	log2	entropy	5	50
0.676230	auto	entropy	5	100
0.676230	sqrt	entropy	5	100
0.676230	log2	entropy	5	100
0.669582	auto	entropy	2	10
0.669582	sqrt	entropy	2	10
0.669582	log2	entropy	2	10
0.669582	auto	gini	2	10
0.669582	sqrt	gini	2	10
0.669582	log2	gini	2	10
0.669582	auto	entropy	3	10
0.669582	sqrt	entropy	3	10
0.669582	log2	entropy	3	10
0.669582	auto	gini	3	10
0.669582	sqrt	gini	3	10
0.669582	log2	gini	3	10

f1	featureSubsetStrategy	impurity	maxDepth	numTrees
0.669582	auto	entropy	2	50
0.669582	sqrt	entropy	2	50
0.669582	log2	entropy	2	50
0.669582	auto	gini	2	50
0.669582	sqrt	gini	2	50
0.669582	log2	gini	2	50
0.669582	auto	gini	3	50
0.669582	sqrt	gini	3	50
0.669582	log2	gini	3	50
0.669582	auto	entropy	2	100
0.669582	sqrt	entropy	2	100
0.669582	log2	entropy	2	100
0.669582	auto	gini	2	100
0.669582	sqrt	gini	2	100
0.669582	log2	gini	2	100
0.669582	auto	entropy	3	100
0.669582	sqrt	entropy	3	100
0.669582	log2	entropy	3	100
0.669582	auto	gini	3	100
0.669582	sqrt	gini	3	100
0.669582	log2	gini	3	100

For the final model, the best hyperparameters for the Random Forest Classifier were chosen using the cross-validation approach. The best model's scores are:

- 82% F1 score on train dataset,
- 78% F1 score on the test dataset,
- and 81% F1 score on the validation dataset.

I can say that the resulting solution is robust because the model generalizes well to the new validation dataset. The difference between train and validation scores is about 1%.

Justification

Summary of the final solution:

- Spark (PySpark and SparkML libraries) was used to analyze data and make predictions.
- The solution is based on dataset, which contains the following extracted features for each user: gender, last user level , the number of days since registration, last location, the average number of events and songs listened per day, the number of thumbs up events, the number of thumbs down events, the number of add friend action events.
- Categorical features (last level, last location, gender) are transformed into dummy variables.
- Numerical features are normalized. Normalization improved scores Gradient-boosted Tree Classifier because without normalization some weights update much faster than others in case of using gradient descent/ascent-based optimization. Normalization also improved scores of Naive Bayes, because this algorithm is sensitive to distances between points.
- In the final model Random Forest Classifier was used to predict churn. This classifier performs best in our case because this method is good at handling categorical variables and is resistant to overfitting. The final model has about 81% F1 score on the validation dataset.
- The flask web application was built to demonstrate the working model. Flask was chosen because it allows building a rapid solution in Python.

Conclusion

Results Summary

The goal of the project is to help the Sparkify service to retain the customers. The solution which I proposed to reach this goal is as follows:

- A large part of the solution is the preprocessing of the initial data. The initial data was in terms of music service events. I transformed it into records in terms of each Sparkify customer. Feature engineering and preprocessing were required to obtain the dataset which is ready for machine learning.
- The second large part of the solution is the machine learning pipeline, which predicts customer churn. I tried several classifiers and compared their F1 scores to choose the best performing solution. I also tuned the hyperparameters with the help of a grid search and cross-validation for the chosen classifier. The final F1 score of the solution is 81%.
- The last part of the solution is the web application which demonstrates the churn prediction. The web application allows the user to enter the information about the customer and then identifies whether this customer is about to churn.
- All parts of the solution are built in Python using Spark.

The most challenging parts of this journey were the feature engineering and the refinement of the model. In feature engineering it is challenging to propose features, which on one hand will help to predict churn and will not overfit the model on the other. Trying to refine the model I tried out several techniques, but not all of them worked (for example, bucketing of continuous numerical features).

Improvement

It was quite a journey, but still, a lot can be done to improve the proposed solution:

- Introduce the different approach for the customer churn prediction. We can try to predict the churn event depending on the sequence of preceding events.
- Use larger dataset for machine learning. Having more data could raise the robustness of the model.
- Try model stacking to raise the accuracy of the prediction. The ensemble of different classifiers often makes a more accurate resulting model.

Appendix

References

#	Description	Reference
1	GitHub repository with the code for the project	GitHub repo
2	Cover image reference	Photo by Carlos Muza on Unsplash