Sparkify Project Summary

by Bobby Mander

March 17, 2020

In this summary we will recap the work involved with our Sparkify music streaming project from start to finish.

Project Overview

Sparkify, the fictional music streaming service, needs our help. They want to survive and grow so in order to do that they need to reduce their customer's churn. Using our data science expertise, we can help them predict which customers will churn and which ones will not. Knowing this ahead of time will allow the company to present offers to the in jeopardy customers in an effort to retain them. Sparkify has given us a rich data set of events for its users so that we may learn how to perform this prediction.

Problem Statement

Churn reduction is a focal point for all subscription based businesses. The cost of losing a customer and the cost of acquiring a new customer to replace that one provides a powerful incentive to reduce churn as much as possible. Businesses like Netflix, Spotify, AT&T, and LA Fitness all would benefit from reduced churn. To help in this effort, predicting churn for those customers most likely to churn would be extremely helpful. If these companies had this information they would be able to present offers to retain those customers and eliminate their churn.

For predicting churn, Sparkify has provided a rich set of event data for their customers. The data is organized by events, one for each row, along with 18 pieces of information for each event. Most of this data we will not need for our prediction, but we will be focused on a few key areas based on our analysis. We will clean and prepare this data for use with various classification models.

Once the data is in place we will train and test various, powerful classification models available. We will also tune hyperparameters using the ultra-efficient grid search method. In the end we will have a trained model with tuned hyperparameters ready for use by Sparkify to identify which customers are likely to churn. Sparkify can target offers to these at risk customers and keep their music playing.

Metrics

The success of the project will be based entirely on prediction accuracy, specifically we will be focused on the f1 score for the churn classification. The higher the f1 score the more accurate the model. We will not focus on the f1 score overall as there are far more non-churn customers than churn customers in the data set.

Analysis

Let's take a closer look at the data we will be working with to train and test the classification models.

Data Exploration

The dataset we will be using is a json file with 286,500 records. Here is an example:

```
{"ts":1538352117000,"userId":"30","sessionId":29,"page":"NextSong","auth"
:"Logged
In","method":"PUT","status":200,"level":"paid","itemInSession":50,"locati
on":"Bakersfield, CA","userAgent":"Mozilla/5.0 (Windows NT 6.1; WOW64;
rv:31.0) Gecko/20100101
Firefox/31.0","lastName":"Freeman","firstName":"Colin","registration":153
8173362000,"gender":"M","artist":"Martha
Tilston","song":"Rockpools","length":277.89016}
```

The same record once read in using Sparkify looks like this:

```
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')
```

Just examining the raw data above, we have targeted the key fields that will be useful for our prediction problem:

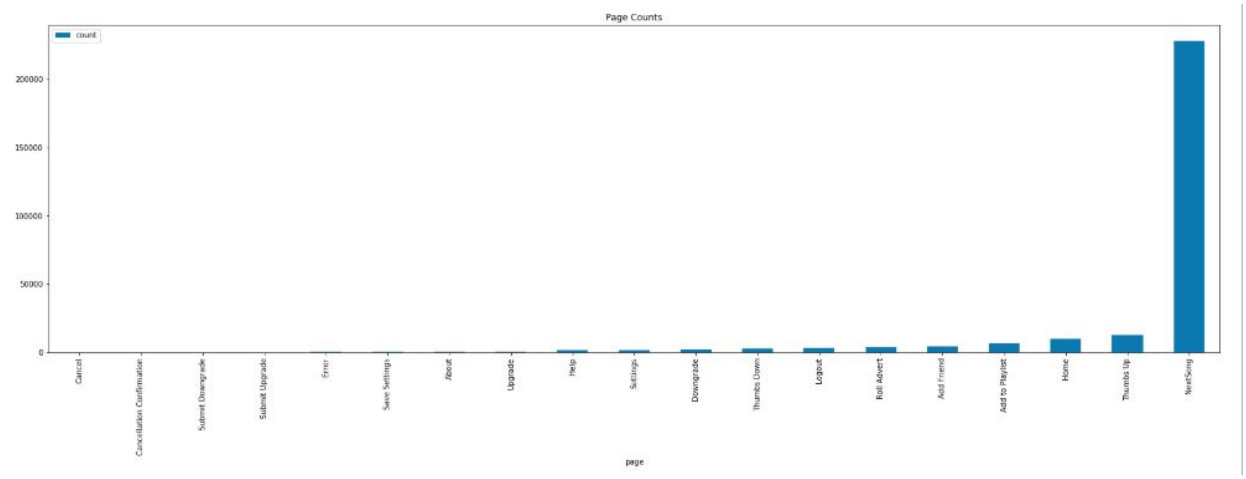
- gender: M, F, or blank
- location: geographic location string
- page type: enumeration of various pages used by the service
- timestamp: integer unix timestamp
- user id: numeric identifier

Using the fields above we will be able to group users and their activity together to find similarities with one another and hence attempt to predict their future behavior. The theory is that similar users will behave similarly in the future with respect to churn.

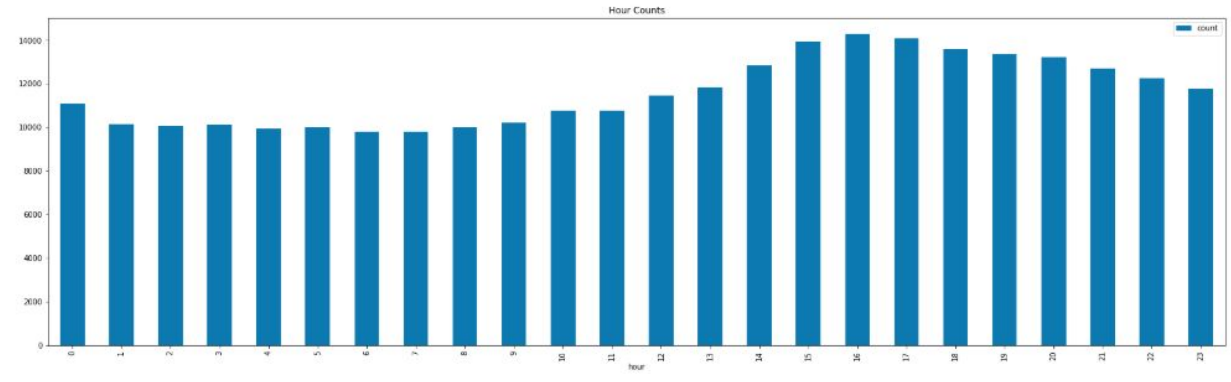
We also need to define churn. For Sparkify a user has churned if they have an event for the 'Cancellation Confirmation' event. If the user does not have such an event they have not churned. Therefore our prediction problem is a binary classification one in which we are most focused on the churn prediction.

Exploratory Visualization

Let's first take a look at the page type distribution across all users. This shows the majority of activity is when users are playing songs, but there are other important pockets of activity including 'Thumbs Up', 'Add to Playlist', and 'Add Friend' as well.



Next let's look at the distribution of song playing throughout the day. This shows a consistent demand for streaming music throughout the day with a gradual peak in the evening hours.



Algorithms and Techniques

Many algorithms and techniques will be used for this project. Here we outline the most important ones:

1. Pyspark. Spark is used for big data problems like this one involving a large event log with many thousands of records. Pyspark is the Python library API for Spark.
    a. Data loading. Loading large files like ours is a snap for Spark.

    b. Data preprocessing. Manipulating that data and preparing it for model use efficiently is Spark's bread and butter.
2. Pandas. After Spark has done the preprocessing we will use pandas to construct the final data model for feeding into our machine learning models.
3. Scikit-learn. This Python library is extensive and powerful with numerous tools for machine learning and data preparation as well.
    a. Data split. Scikit-learn will easily split our data into training and test sets.
    b. Classification model implementation. Scikit-learn offers so many classification models and we will try them all.
    c. Grid search. Scikit-learn provides a grid search library that makes running a grid search efficient and automated.

Benchmark

In terms of benchmarks, we will target achieving a churn f1 score of 0.50 with a non-churn f1 score of 0.8 and an overall f1 score of 0.75. Since there are far fewer customers who churn versus who do not churn, this will be a nice achievement and useful for Sparkify.

Methodology

Here we will share details of the processing we performed in order to achieve our final model that predicts churn.

Data Preprocessing

The event log data we will be using first had to be cleaned. After analysis we found that there were over 8,000 events with null data for various fields and a blank string for the user id. We eliminated these rows as without a user id the data was useless from a prediction perspective. Eliminating these rows left us with 278,000 events to work with.

Based on the key fields we described earlier, we decided to concentrate on the following in order to form the features for our problem:

1. **Gender.** As described earlier.
2. **Location**. As described earlier.
3. **Thumbs Up total**. We will count how many times the user applied a thumbs up to a song. The more the user applies these, the more sticky they should become to the service since the service knows more and more about them.
4. **Thumbs Down total**. We will count how many times the user applied a thumbs down to a song. Again, this should be indicative of the stickiness of the user but could also indicate that the user is not satisfied and/or the recommendation system used by the service is not working very well.
5. **Add to Playlist total**. We will count how many times a user added to a playlist. Again, a playlist is a service feature that helps make users more sticky and will help reduce churn.

6. **Add Friend total.** We will count how many times a user added a friend. Due to network effect, the more friends you have on a service, the more likely you are to stay with that service, thus reducing churn.
7. **Error total**. We will count how many errors the user encountered. Technical difficulties would turn off a user and make it more likely they would churn.
8. **Next Song count per day for last 7 days**. We will count how many songs the user has played per day for the last 7 days of their training data set. The theory is that if the user is active they should not churn, but if the user shows signs of slowing down their usage of the service recently, they would be more likely to churn.
9. **Next Song count per hour bucket on average over all time**. We will count how many songs the user has played per hour bucket over the training data set. An hour bucket is a grouping of hours so that songs can be summed over this time window. If you have 8 hour buckets, then each bucket represents 3 contiguous hours to count songs over. The theory here is that by looking at the song playing frequency over the course of a day we can categorize users and find users that are more alike to each other. This will lead to a better prediction.

Our data preprocessing involved constructing the above aggregated metrics as features from the raw event data.

Implementation

Implementation of the Sparkify solution involved the following technologies and techniques:

- AWS. Using Amazon's cloud, we were able to leverage SageMaker and Jupyter notebooks directly and cheaply using a base instance type and the conda_python3 kernel.
- Pyspark implementation in AWS. This was full featured but unfortunately we could not read from s3n locations due to a known limitation with the library.
- Scikit-learn. The many full featured libraries in scikit-learn were used for classification prediction. In particular, we used:
    - LogisticRegression. The most basic model for classification.
    - SVC. Support vector classification model.
    - K Neighbors. Cluster based classification model.
    - Decision Tree. Tree based classification.
    - Random Forest. Multiple tree based classification.
    - MLP. Multi-layer perceptron or neural network based classification.

Minor complications did occur during the implementation. These included:

- Grid search performance. Grid search can take quite a long while depending on the parameter grid you specify and how many options there are. The Udacity notebook performance was average so we switched to an AWS base instance type and sped up performance considerably. We also narrowed down the list of parameters to search for with trial and error.

- Songs per hour bucket and songs per day calculation. We experimented quite a bit with these algorithms before finding a better solution for our model. Trial and error and brainstorming was key to solve this and refine it as described below.

Refinement

As described, our set of hyperparameters was vast so many combinations had to be tried to find the best results. Here are the various hyperparameters we tuned for each model using grid search:

- KNeighbors:
  - 'weights': ['uniform', 'distance'],
  - 'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
  - 'p': [1, 2, 3]
- MLP:
  - 'max_iter': [3000, 5000],
  - 'activation': ['logistic', 'tanh', 'relu'],
  - 'warm_start': [False, True],
  - 'hidden_layer_sizes': [(200,), (100,), (300,), (50,10), (50,20), (100,20), (200,20)]
- DecisionTree:
  - 'criterion': ['gini', 'entropy'],
  - 'splitter': ['best', 'random'],
- RandomForest:
  - 'criterion': ['gini', 'entropy'],
  - 'n_estimators': [100, 300, 500],
  - 'bootstrap': [False, True]

We also performed refinement in terms of the feature set and the scaling of the feature set, specifically:

- Songs per hour bucket. We started with 24 features, one per hour window during the day. But this was far too many and would overwhelm our other features so we implemented hour window buckets of 3hrs each to reduce this to 8 buckets. We also saw that different users had quite different frequencies of song playing but we wanted this normalized so instead of using the total song count in the hour window, we scaled this by the total songs the user listed to overall so users would be more comparable.
- Songs per recent day. We started looking at songs in the past 14 days but this also would overwhelm the other features so we limited to the last 7. We also changed the algorithm to look at consecutive days and not the last 7 active days when the user played some songs. Finally we normalized the data as well so that the song count per day would be more comparable across users.

Results

Let us now present and evaluate the results we obtained.

Model Evaluation and Validation

Using grid search on the most promising classification models, we achieved the results below:

Classifier	**Churn f1**	**Non-churn f1**	**Overall f1**
K Neighbors	0.40	0.88	0.76
MLP	0.36	0.79	0.69
Decision Tree	0.60	0.89	0.82
Random Forest	0.53	0.91	0.82

The Decision Tree classifier was best overall using the grid search found hyperparameters 'criterion' set to 'entropy' and 'splitter' set to ''best'. Grid search was used with 3 folds so this appears to be a robust solution.

Justification

In the final model we exceeded our target benchmark performance target. Using the Decision Tree classifier we achieved a churn f1 score of 0.60, exceeding our target of 0.50 by 20%, a non-churn f1 score of 0.89, exceeding our target of 0.80 by 11%, and an overall f1 score of 0.82, exceeding our target of 0.75 by 9%.

Some of the more simplistic classifiers were ruled out early as this data set is complex and would be difficult to fit during training. Of the candidate classifiers, it seemed the MLP would be a good fit considering the complex nature of the data. It did not perform nearly as well as Decision Tree which was able to find the best ways to use the features to lead to our prediction. Often it's not clear which classifiers will work best beforehand so trial and error is necessary.

We also altered the feature set often to find the impact on prediction quality. We removed the songs per hour bucket feature, then we removed the songs per recent day feature, and then we changed the parameters to those features such as the number of buckets and the number of days. We tried all combinations of removed features as well. We found that including them all was best and using the 8 hour buckets and 7 days strategy gave the best results.

Reflection

The solution provided to Sparkify is to use a Decision Tree classification model with processed data as described by the features we outlined previously. Taking the large data set and distilling it into these features will provide the ability to predict which users will churn to a good degree of accuracy. The project provided a very interesting problem domain, gave us the ability to use

Spark, and let us try many, many classification models along with the powerful grid search. We enjoyed seeing the results of our efforts in making this user event log useful for prediction of churn.

Improvement

Some improvements could be made based on these results. One idea is to create a grid search on which features to include or not. For this project this was a manual exercise, but if the grid search concept could be applied to feature inclusion that could be made much more efficient and automated. Another idea is to try even more classifier models either from scikit-learn or elsewhere.

References

1. Amaresan, Swetha "What is Customer Churn?" https://blog.hubspot.com/service/what-is-customer-churn
2. Stec, Carly "How to Reduce Customer Churn" https://blog.hubspot.com/service/how-to-reduce-customer-churn
3. Orac, Roman "Churn prediction" https://towardsdatascience.com/churn-prediction-770d6cb582a5
4. MLPClassifier: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
5. Decision Tree Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
6. Random Forest Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
7. K Neighbors Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html