Tackling User Churn with Magic (or DataScience…)

Image from: Capterra: The Top 3 Customer Success Factors Every SaaS Needs

After the initial issues with piracy, streaming seems to have evolved into a viable solution to the question of how to sell music in the 21. Century.[1] Especially during the pandemic it seems that streaming services of any kind, including those streaming music have gained in popularity.[2] This is why the industry has attracted a lot of new players in the industry – increasing the competition. Even the trillion dollar company Apple joined the game investing $3 billion subscription streaming music service Beats Music, and Beats Electronics.[3] At the same time it has become very easy to compare prices and features via the internet, making it is easy for users to switch a digital service. From the perspective of the consumers this is great as more competition will likely yield ore customer centric and cheaper services in the long run. For companies on the other hand, it is not that great. Companies have to keep a close eye on the number of their users in order to stay competitive. They have to be able to offer the right incentives to stay before a user switches to another service. This is commonly related to as 'churn'. 'Churn' is defined as 'the percentage of customers who stop buying the products or services of a particular company, calculated for a particular period of time'.[4]
We will be trying to help solve this issue by understanding better which users churn and predict those users early on to incentivize them to stay.

[1] https://www.recordingconnection.com/reference-library/recording-entrepreneurs/how-the-internet-changed-music/
[2] https://www.grandviewresearch.com/industry-analysis/music-streaming-market
[3] https://www.apple.com/newsroom/2014/05/28Apple-to-Acquire-Beats-Music-Beats-Electronics/
[4] https://dictionary.cambridge.org/dictionary/english/churn-rate

Project Overview

For this project, we are going to predict user churn using the historical data on customer interactions with the 'Sparkify App' – a fictional music streaming provider. For the project we will be using PySpark.

The Data

Udacity kindly provided the subset data (128MB) in the form of a JSON file. First, a Spark Session is initiated and the data is loaded. The first thing to do is always cleaning the data, so that is what we did. Empty strings for userIds were eliminated and 278,154 rows were left afterwards. The data contained users' interaction history with the Sparkify app (find the details in the GitHub project with all of the accompanying code).
After performing the cleaning of the data, we can start with EDA.

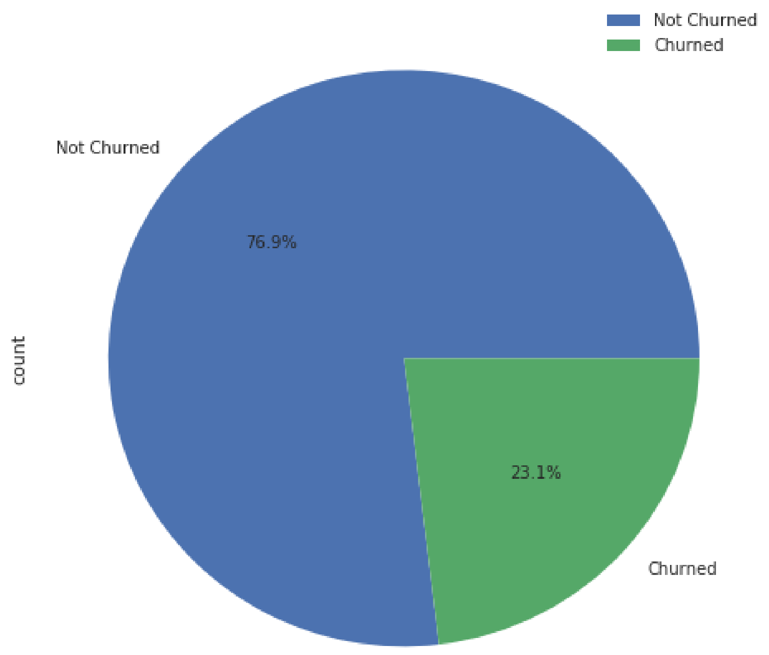
Exploratory Data Analysis

Initially we had to define churn with the data at hand. This meant using a 'Churn' column to use as a label for our model from the 'Cancellation Confirmation' page, appearing in the page features of the dataset. A 1 was assigned when users had churned and a 0 when they were loyal to the platform. Putting other characteristics into perspective with the churn will give us a better comprehension of which features can be used for churn prediction.

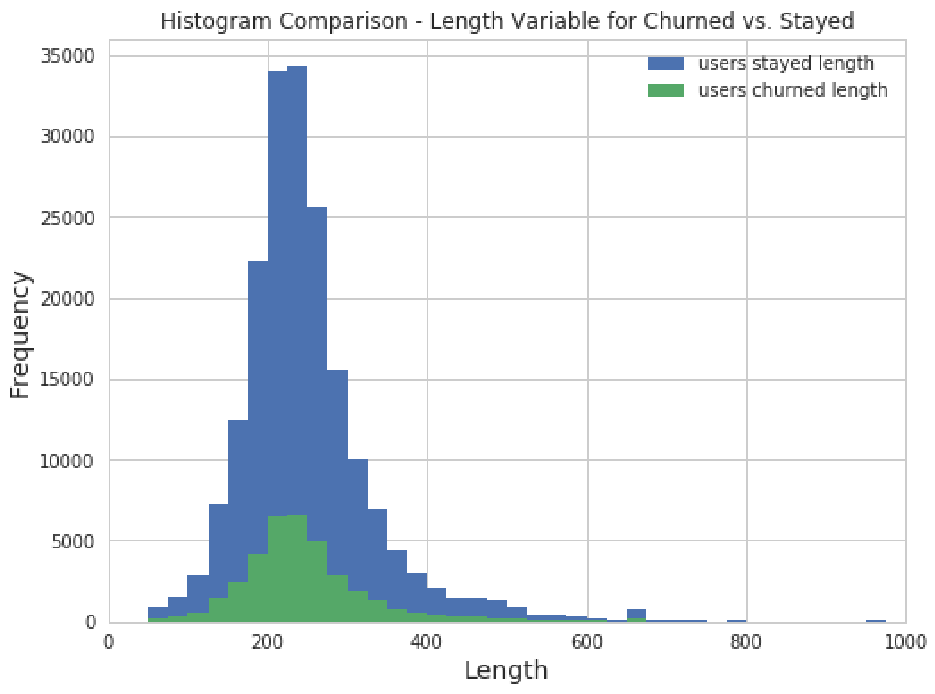
Number of Customers Churned

The below plot shows that 23% (52) of Sparkify's users from this dataset have churned. 173 users stayed with the platform (77%). As we can see from the graphs, there is quite an imbalance here.

Comparison of unique user churned and loyal users:

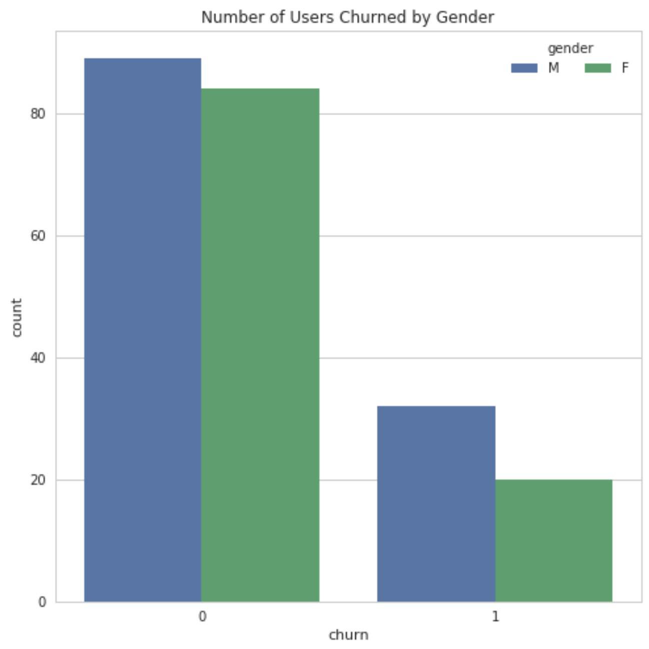


As we can see in the overlay of the histograms, the distributions are quite similar. This will most likely not be a good predictor for our churn rate. Let's take a look at another feature.



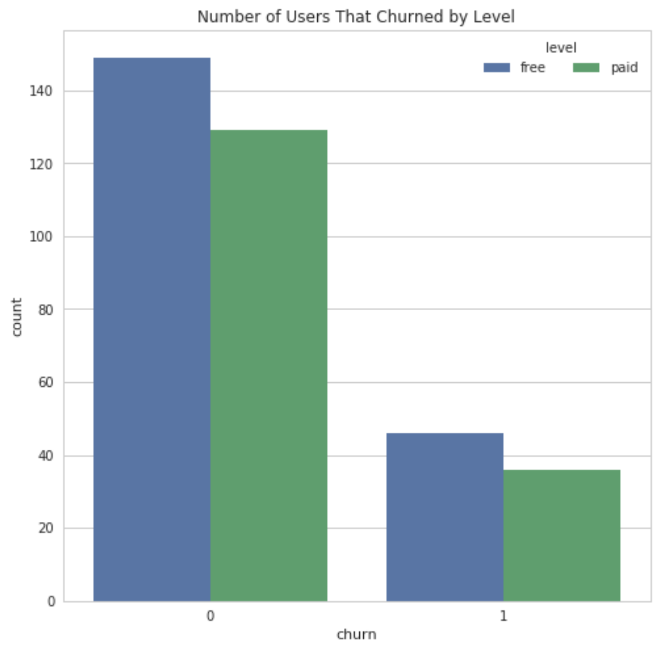
Gender

The first feature analysis indicates a higher tendency towards churning for male users.



Level - Free vs Premium

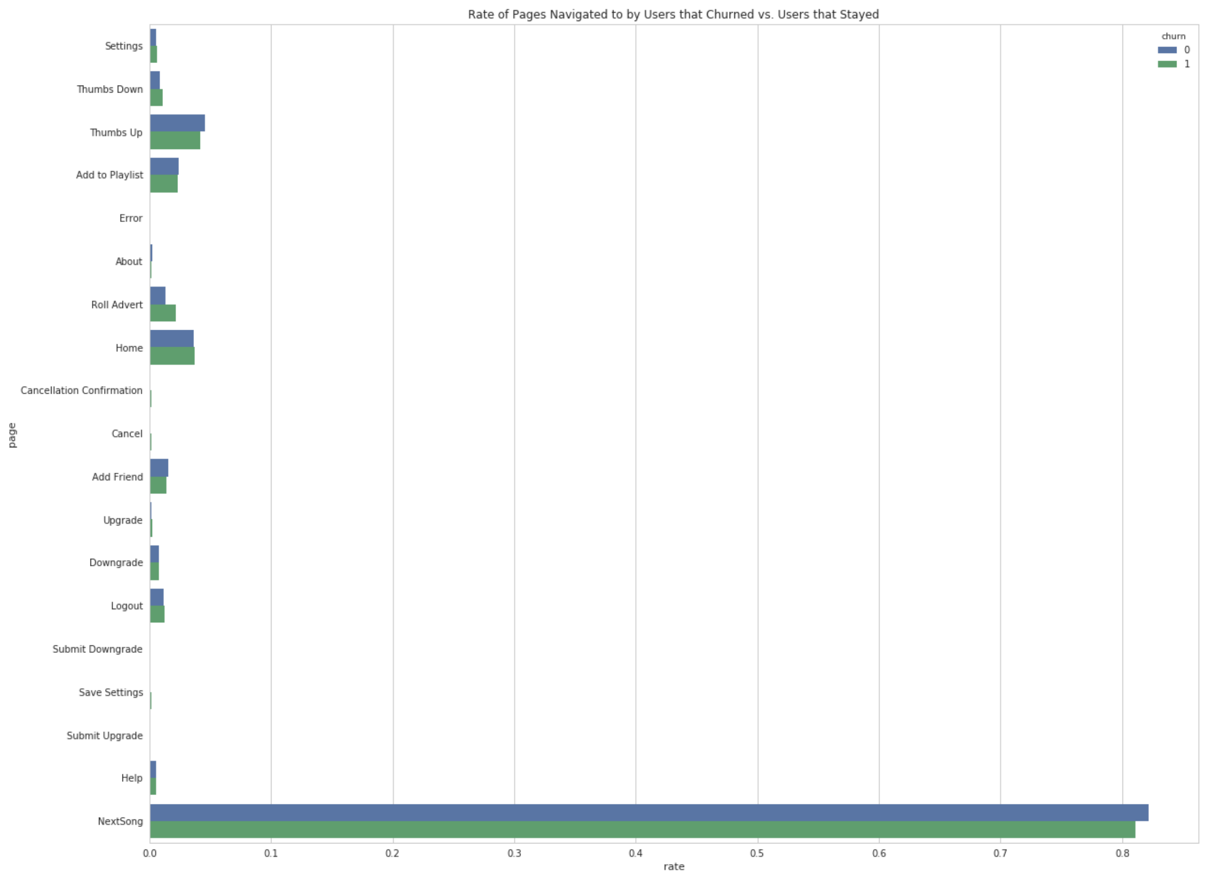
More users with a free subscription (~24%) level have churned when compared to the paid (~22%) users. The difference is rather marginal, so it is difficult to say if our previous assumption was really right.



Pages Visited

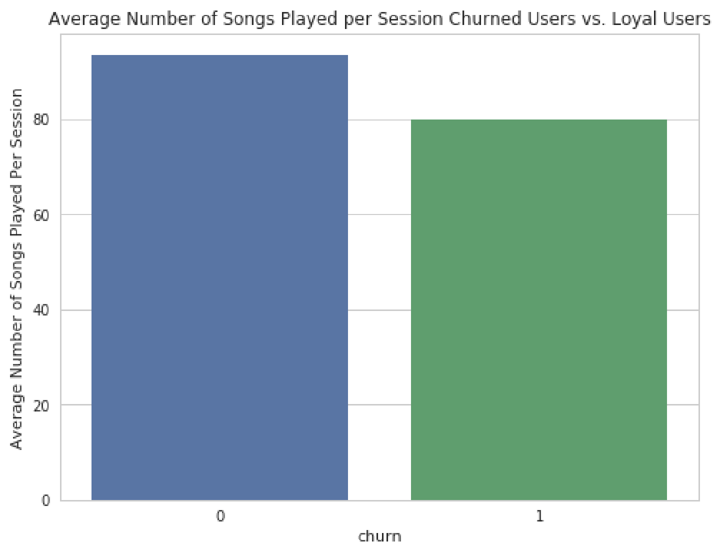
The most popular action for both user groups was the NextSong action.

Churned users used the thumbs down action more while the thumbs up action was clicked more often by the loyal users. Those who were more likely to stay also added friends and songs to playlists.



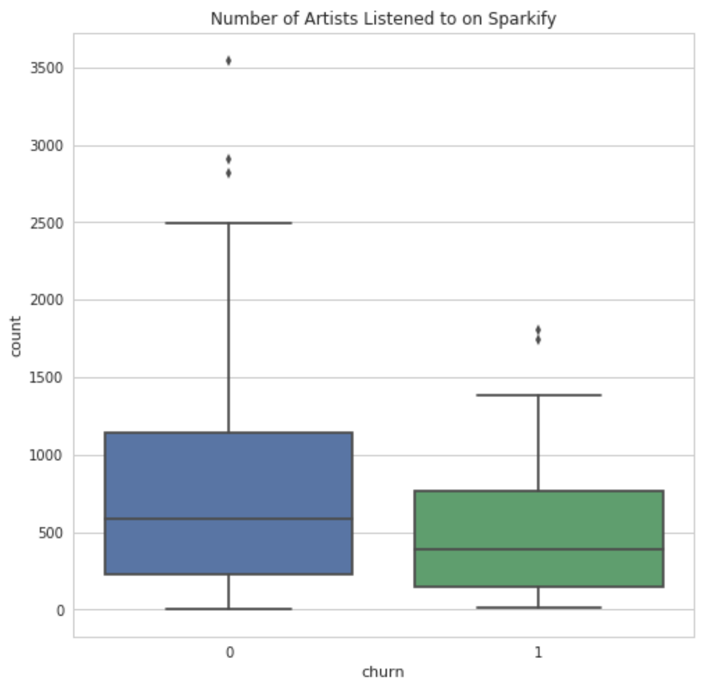
Average Songs Per Session

Per session the churned users were listening to fewer songs than those who stayed.



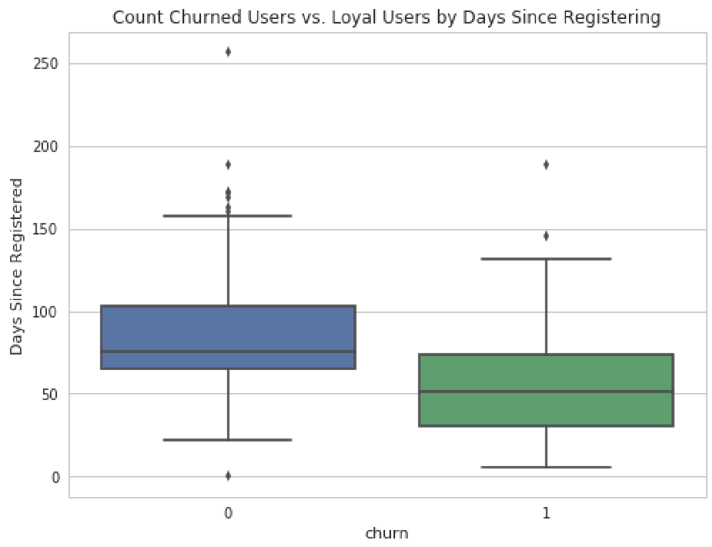
Number of Unique Artists

The loyal users listened to a greater variety of artists compared to the churned users.



Number of Days Since Registering

User who had been registered for longer are on average more loyal. More recently registered users are more likely to churn.



Feature Engineering

Now that we have gathered the first insights with EDA, we can start building out the most promising features to train our model on.

We will be focusing on:

Categorical:
`gender`
`level`

Numerical:

```
number of songs per session
number of rollads actions
number of thumb down actions
number of thumbs up actions
number of friends added
number of songs added to playlist
number of different artists listened to on Sparkify
number of days since registering
```

Furthermore, we will add a label for `churn` to be able to join these together in the end. A dataFrame where each row represents the information pertaining to each individual user will thus be created. When the userId is dropped, the df can be vectorized, standardized and delivered to various ML algorithms. learning algorithms.

We will start by converting our categorical variable `gender` and `level` to numeric variables.

Preprocessing

After changing our datatypes to floats we can vectorise and standardise our data.

Vector Assembler

First, we needed to transform our features into a vector which could be fed into the standard scaler. This is important to our machine learning model because it prevents those features with the highest values from dominating the results.

```
assembler = VectorAssembler(inputCols = ["gender", "level", "avg_song", "roll_ad", "thumbs_down", "thumbs_up", "add_friend", "playlist", "num_artists", "days"], outputCol = "vec_features")feature_df = assembler.transform(feature_df)
```

Standard Scaler

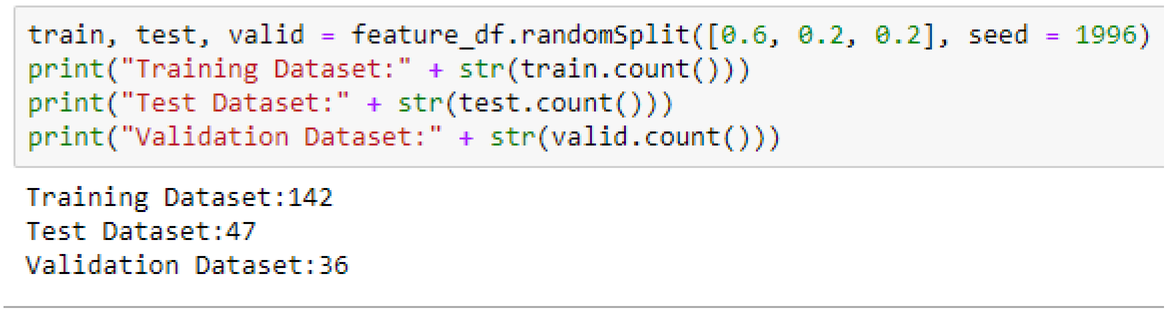
Now that we have performed vectorisation, we can standardise our values.

```
scaler = StandardScaler(inputCol="vec_features", outputCol="features", withStd=True)scaler_model = scaler.fit(feature_df)feature_df = scaler_model.transform(feature_df)
```

You should now see that the values are standardised. This is important because machine learning estimators may not play ball if individual features don’t look like standard normally distributed data.

Train/Test/Validation Split

Then we split our dataset into train, test and validation sets. We performed a 60:20:20 split respectively. We also included a seed for reproducibility.



```
train, test, valid = feature_df.randomSplit([0.6, 0.2, 0.2], seed = 1996)
print("Training Dataset:" + str(train.count()))
print("Test Dataset:" + str(test.count()))
print("Validation Dataset:" + str(valid.count()))
```

```
Training Dataset:142
Test Dataset:47
Validation Dataset:36
```

Modelling

Now we have created our features dataFrame with only numeric variables, we split the full dataset into train, test, and validation sets. We then tested out different machine learning classification algorithms, including:

- Logistic Regression
- Random Forest Classifier
- Gradient-Boosted Tree Classifier
- Linear Support Vector Machine
- Naive Bayes

We will use these classification algorithms since churn prediction is a binary classification problem, meaning that customers will either churn (1) or they will stay (0) in a certain period of time.

Metrics

Finally, we can move onto the modelling. We will be evaluating using accuracy and F1 score. We are using F1 score because this is an unbalanced dataset with only 52 customers in it having churned. First up was good old reliable logistic regression.

Logistic Regression

Logistic regression is the first machine learning algorithm we can try. Logistic regression is a reliable machine learning algorithm to try since this is a binary classification problem and logistic regression provides a model with good explainability. Logistic regression is also easy to implement, interpret and is efficient to train. It is also less inclined to overfitting.

Results:

- **F1 Score:** 0.65
- **Accuracy:** 0.72
- **Time Taken:** 110 seconds

Not terrible. But not good. We tried Random Forest Classifier.

Random Forest

Random Forest is a powerful supervised learning algorithm that can be used for classification. RF is an ensemble method that creates multiple decision trees to make predictions and takes a majority vote of decisions reached. This can help avoid overfitting. RF is also robust and has good performance on imbalanced datasets like we have here.

Results:

- **F1 Score:** 0.87
- **Accuracy:** 0.88
- **Time Taken:** 179 seconds

That is much more like it. Random Forest was also relatively quick to run. We liked Random Forest. We decided to find out if Gradient Boosted Trees could beat it.

Gradient Boosted Tree Classifier

GBT provides good predictive accuracy. This works by building one tree at a time where each new tree helps correct errors made by the previous tree compared to RF which builds trees independently. There is a risk of overfitting with GBT so this needs to be considered. However, GBT performs well with unbalanced data which we have here.

Results:

- **F1 Score:** 0.88
- **Accuracy:** 0.88
- **Time Taken**: 248 seconds

Nice! A *slightly* better F1 score than Random Forest, although it did take a minute longer to run for tiny gains. I was favouring Random Forest. But for the sake of fairness, we gave Linear Support Vector Machines a chance to overtake.

Linear SVC

SVC is another supervised learning binary classification algorithm. It works well with clear margins of separations between classes and is memory efficient.

Results:

- **F1 Score**: 0.68
- **Accuracy**: 0.78
- **Time Taken:** 3649seconds

That took a while. It was not worth the wait.

Naive Bayes

Finally, we will try Naive Bayes. This is another classifier algorithm that is easy to implement and is fast.

Results:

- **F1 Score:** 0.68
- **Accuracy:** 0.78
- **Time Taken:** 119 seconds

Relatively fast but didn’t have a great F1 Score.

Model Tuning

I chose to tune the Random Forest model, as this classifier and GBT achieved almost idenitical results but Random Forest took a minute less to run. We can get the parameters to optimise by running:

```
print(rf.explainParams())
```

We can then use ParamGridBuilder and CrossValidator.

I will select numTrees and maxDepth for our RF model tuning.

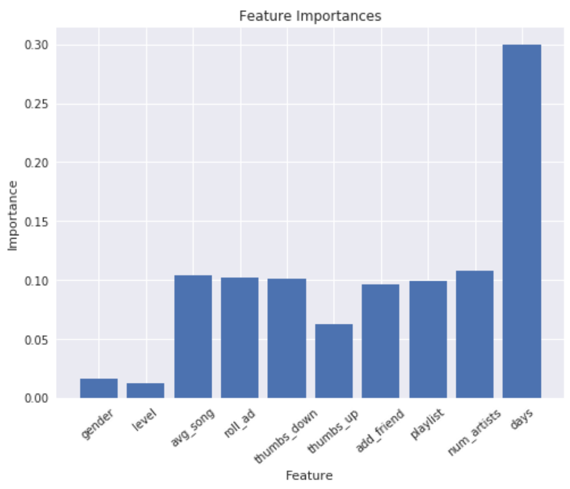
- **NumTrees**: I have chosen to go up to 100 trees to improve performance. Since these trees are individual randomised models in an ensemble there is not a great risk of overfitting with this numTrees parameter.
- **Maxdepth**: I have chosen a max of 15 to reduce the possibility of overfitting. Anything over 15 would increase the risk of overfitting greatly.
- **Numfolds**: I originally had numFolds = 5 but had to change to 3 to speed up the process.

```
paramGrid = ParamGridBuilder()\
   .addGrid(rf.numTrees, [20,50,100])\
   .addGrid(rf.maxDepth, [5,10,15])\
   .build()crossval = CrossValidator(estimator = rf,
                    estimatorParamMaps=paramGrid,
                    evaluator = MulticlassClassificationEvaluator(metricName="f1"),
                    numFolds = 3)
```

This took quite a long time to run. Our final performance scores are:

- **F1 Score:** 0.88
- **Accuracy:** 0.88

This is a good result. We can also look at feature importance of our best model:



From the above, we can see that days registered was the most important feature in our churn prediction.

Conclusion

We started the project with a small dataset of just 128MB and 225 unique customers. After loading and cleaning our data we explored the dataset for useful features to predict churn and were able to build out the most promising features. We then preprocessed these and used the features with different machine learning algorithms. Random Forest performed the best, achieving an accuracy and F1 score of 0.88.

Business Impact:

Now, Sparkify can use this information to target customers who are likely to churn and offer attractive incentives to stay, thereby saving Sparkify revenue and getting the customer a nice deal. Since newer customers are more likely to churn, we could target them with a nice free trial of the premium service without those pesky ads! It’s simple: predict churn, nip it in the bud, save your business — and customers — money. It could be the difference between saving your business from going under and losing it to the churning sea of our competitive economy.

Future Work:

This project could have been improved by:

- More feature engineering to select the best features to get a better score
- Investigating overfitting problems in more depth
- Analysing mispredicted users