Predicting churn, that is when a user cancels a service/subscription, is desirable for companies for a number of reasons, including having the ability to issue targeted promotions/incentives to retain users at risk of churning. Additionally, these predictive models can be analyzed to enable companies to identify what features lead to churn and make estimates about stability of their user population. In this post, I explore a large dataset of user account activity over time from a streaming music provider, engineer time bin features, and build/compare models from three different classifiers (Logistic Regression, Random Forest, and Gradient-Boosted Tree) predicting: (A) current time bin prediction - if a user will churn in a time bin given that time bin's feature values, (B) next time bin prediction - if a user will churn in the next time bin given the previous time bin's feature values, and (C) will churn soon prediction – if a user will churn in the current or following time bin (either A or B are true).

Exploration:

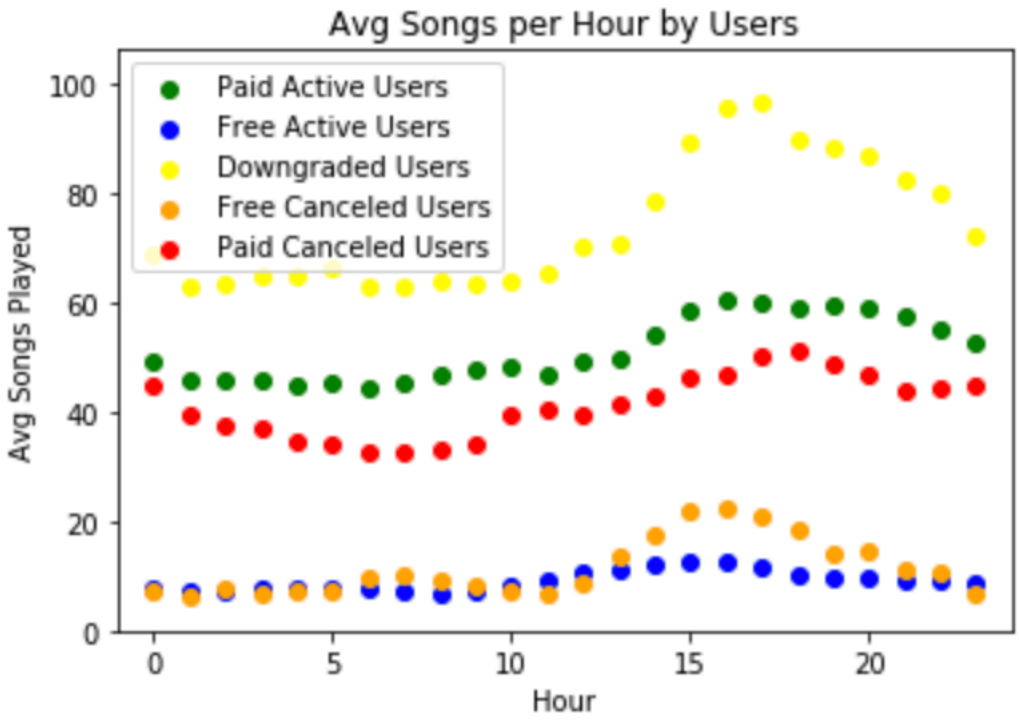
The code for this project leverages pyspark and is available here. The full dataset is 12GB, but for the purposes of this work I'll be working on a 128MB sample containing 286,500 rows, and is available via the Udacity Sparkify project. The schema for the data is:

```
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)
```

The "page" values provide insight into the actions that the user is taking over time:

page
Cancel
Submit Downgrade
Thumbs Down
Home
Downgrade
Roll Advert
Logout
Save Settings
Cancellation Conf...
About
Settings
Add to Playlist
Add Friend
NextSong
Thumbs Up
Help
Upgrade
Error
Submit Upgrade

To begin, I performed some exploratory analysis of the data based on free versus paid users that churn, downgrade, or remain unchanged based and their average daily listening habits:



Initial analysis just based on average listening habits per hour of the day for each user classification reveals:

- Paid users who churn (red) listen about 10-20% less on average than users who don't (green). This seems expected, as users who are not using the service as regularly are not seeing the value in continuing to pay and are canceling.
- Unexpectedly though, it appears that users who downgrade (yellow) are actually very active users who listen to about 30-50% more songs on average than the average paid user (green or red). A possible reason could be that highly active users could be drawn to another type of service, so they are downgrading.
- Looking at activity for the free user accounts, these users have much less activity than the other users. Upon this initial analysis though, it appears difficult to see any user activity trend corresponding to free user account churn.

In the next section, I'll engineer some additional features that will be useful in our models for further separating and predicting user churn.

Feature Engineering:

Because a user's activity can change over time, such as being a highly active user after sign-up and then being a dormant user around the time they churn, it is important to look at user activity within time bins. Since subscriptions are usually monthly based, I decided to look at activity in two-week windows and try to make predictions for the current window and for the next window. These two-week time windows will also help to normalize/smooth out any noise or irregularities from day to day usage of the service, such as weekend or other. The time bins

are built from the perspective of a user, so that time bin 0 is the first 14 days of activity for a user's account – this normalizes the time bin number meaning.

Within user time bins, I added engineered the following features:

- PaidInTimeBin and FreeInTimeBin – binary features about the user's subscription level, it is possible for a user to have both binary features set in a timebin if they change their subscription level
- ChurnedInTimeBin, DowngradedInTimeBin, UpgradedInTimeBin, PreviouslyDowngraded, WillChurnInNextBin, and WillChurnSoon – binary values about user changes to their account, these can be used as features or labels for our models
- OneHotEncoding of the page name with count of the times the user visited the page in the time bin. Note: I throw out some of the pages that we already have encoded (e.g., 'Cancellation Confirmation' == ChurnedInTimeBin)
- SessionsInTimeBin, ArtistsInTimeBin, and DistinctSongsInTimeBin – these are count distinct features of sessions, artists, songs for the user within the time bin

Null UserId records are thrown out, and all of the engineered features are converted to integers with nulls replaced as zeros.

This is the schema of the dataframe that I built and used in my models:

```
root
 |-- UserId: integer (nullable = true)
 |-- UserTimeBin: integer (nullable = true)
 |-- Gender: integer (nullable = true)
 |-- PaidInTimeBin: integer (nullable = true)
 |-- FreeInTimeBin: integer (nullable = true)
 |-- ChurnedInTimeBin: integer (nullable = true)
 |-- DowngradedInTimeBin: integer (nullable = true)
 |-- UpgradedInTimeBin: integer (nullable = true)
 |-- DaysRegisteredAtTimeBin: integer (nullable = true)
 |-- WillChurnInNextBin: integer (nullable = true)
 |-- PreviouslyDowngraded: integer (nullable = true)
 |-- About: integer (nullable = true)
 |-- Add Friend: integer (nullable = true)
 |-- Add to Playlist: integer (nullable = true)
 |-- Cancel: integer (nullable = true)
 |-- Cancellation Confirmation: integer (nullable = true)
 |-- Downgrade: integer (nullable = true)
 |-- Error: integer (nullable = true)
 |-- Help: integer (nullable = true)
 |-- Home: integer (nullable = true)
 |-- Logout: integer (nullable = true)
 |-- NextSong: integer (nullable = true)
 |-- Roll Advert: integer (nullable = true)
 |-- Save Settings: integer (nullable = true)
 |-- Settings: integer (nullable = true)
 |-- Submit Downgrade: integer (nullable = true)
 |-- Submit Upgrade: integer (nullable = true)
 |-- Thumbs Down: integer (nullable = true)
 |-- Thumbs Up: integer (nullable = true)
 |-- Upgrade: integer (nullable = true)
 |-- SessionsInTimeBin: integer (nullable = false)
 |-- ArtistsInTimeBin: integer (nullable = false)
 |-- DistinctSongsInTimeBin: integer (nullable = false)
 |-- WillChurnSoon: integer (nullable = true)
```

Class Imbalance Problem:
Since I'm focusing on predicting user churn – I'm interested in building models that will accurately predict the following labels:

- ChurnedInTimeBin – meaning the user cancelled their subscription within the time bin that we are looking at
- WillChurnInNextBin – meaning that the user will cancel their subscription in time bin following the one that we are looking at
- WillChurnSoon – meaning that either of the previous labels are true (that the user will churn in the current or next time bin)

However, when we look at the distribution of values, we see that user time bins with a positive churn label are observed far fewer times (~7% of the time) than those without churn labels:

	ChurnedInTimeBin	**WillChurnInNextBin**	**WillChurnSoon**
0	809 (94%)	822 (95%)	770 (89%)
1	52 (6%)	39 (5%)	91 (11%)

Table A: Distribution of Labels

To view the impact of the class imbalance on when building a model, I built a Notebook (rebalance_test.ipyb) where I trained random forest classifiers for each of the three labels with and without sampling. The sampling was achieved using the pyspark dataframe function sampleBy(), and the values in the fraction values for downsampling the non-churn time bins were tied to the amount of positive values for each label (Table A).

	ChurnedInTimeBin		**WillChurnInNextBin**		**WillChurnSoon**	
AUC	Unsampled	Sampled	Unsampled	Sampled	Unsampled	Sampled
PR	0.0916	0.6123	0.0721	0.7720	0.1360	0.6538
ROC	0.6203	0.5079	0.5564	0.7813	0.5254	0.6519

Table B: AUCs for Unsampled vs. Sampled RF Models

The above table shows that we can achieve significant improvements (~60x) in the area under the precision-recall curve (AUC PR) metric when rebalancing the dataset used in the random forest train/test split. The values here are not extraordinary, but keep in mind that we have not done any parameter tuning/cross-validation yet.

When evaluating data with class imbalance, areaUnderPR is the recommended metric of evaluation during cross validation (ref https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/) – and we can see that the Precision-Recall AUC is much more meaningful in Table B than ROC.

Churn Prediction Models:
Next, I built and examined 9 total models by using 3 different classifiers (Logistic Regression, Random Forest, and Gradient-Boosted Trees) to build predict 3 different binary labels: (A)

ChurnedInTimeBin – if a user churns in the time bin that is being analyzed, (B) WillChurnInNextBin – if a user churns in the bin following the one being analyzed, and (C) WillChurnSoon – if either of the previous are true.

The below table shows the AUC PR for each of the models:

	Logistic Regression	Random Forest	**Gradient-Boosted Trees**	Average
ChurnedInTimeBin	0.6035	0.4990	**0.6521**	0.5849
WillChurnInNextBin	**0.8588**	**0.8697**	**0.9442**	**0.8909**
WillChurnSoon	0.4332	0.4017	**0.7408**	0.5251
Average	0.6318	0.5901	**0.7790**	0.6670

Table C: AUC PR for each model (classifier + label)

Gradient-Boosted Trees (GBT) was the best classifier for making predictions on this dataset, and the WillChurnInNextBin was the best label for accurately predicting churn. The label with the best prediction was somewhat surprising, as I would have expected ChurnedInTimeBin or WillChurnSoon labels to have the highest predictive accuracy since the classifiers are weighing the features in the same time bin that the user cancels. However, it appears that there are tells in the features the time bin prior to the user's churn that allows for making this future prediction – this bodes well for forecasting near-future churn in time for companies to offer promotions/incentives for them to stay.

For further examining the features used in predicting WillChurnInNextBin I looked at the coefficient weights of the Logistic Regression model and noticed that there are a number of features that are zeroed out:

Feature	**Weight**
UserTimeBin	0.0
Gender	0.0
PaidInTimeBin	0.0
FreeInTimeBin	0.0
DowngradedInTimeBin	0.0
UpgradedInTimeBin	0.0
PreviouslyDowngraded	0.0
About	0.0
Add Friend	0.0
Add to Playlist	0.0
Downgrade	0.0
Error	0.0
Help	0.0
Home	0.00105
Logout	0.0

NextSong	0.0
Roll Advert	**0.00740**
Save Settings	**0.10228**
Settings	**0.01724**
Thumbs Down	**0.07118**
Thumbs Up	0.0
Upgrade	0.0
SessionsInTimeBin	0.0
ArtistsInTimeBin	0.000007
DistinctSongsInTimeBin	0.000003

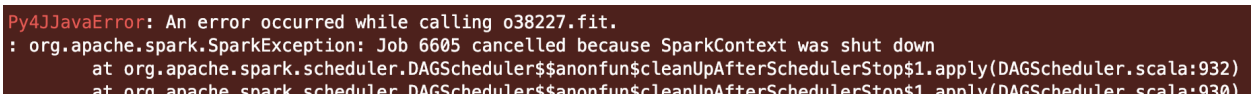
Table D: Logistic Regression weights for WillChurnInNextBin label prediction

From this table of weights, we can infer that recent changes to a user's settings is an important predictive feature followed by negatively reviewing songs and receiving advertisements.

Model Tuning & Technical Hurdles:

From running the models numerous times against the data while also doing hyperparameter tuning/cross-validation, it seemed to me that that the AUC PR values vary more drastically based on the data fed into the model's train/CV steps resulting from the sampling and split versus any hyperparameter tuning. Additionally, I did not see noticeable gains in accuracy when I increased the number of trees or iterations parameters or CV folds (just took longer to run).

Since I opted to explore 9 models at once, this took some time to run (~3 hours) with a local Spark context. A number of times I ran into out-of-memory errors or having the Spark context disappear (screenshot below) around the time that the code reached the 4th model in the loop. I was able to improve the speed of the run by moving specific dataframes into the Spark cache and improve memory usage by calling the unpersist() function when done with the dataframes. I also looked at my system performance when the model training/CV seemed to slow down dramatically, and I noticed that the java process was using 1.93GB of memory of the 2GB allocated to it (Xmxsize in Java 8) – in other words, my JVM was thrashing.



I recommend increasing your JVM memory size and/or running on a Spark cluster. (I ran on a local Spark context as part of a Udacity project to learn pyspark and how to build a model to predict user churn).

Next Steps:

Below is a list of next steps for experimentation and improving upon this work:

- Explore if it is better to combine into a multi-class predictive model versus multiple binary models. The individual models gave me more control, which is why I used that approach initially.
- Track and include a user's running average of activity over time to use for feature engineering – maybe a deviation from their average could be a good feature versus just comparing the user's activity to all users when building a model (i.e., track each user's pattern of life in a time bin and deviations from it within time intervals).
- Experiment with different time bin sizes to see changes in accuracy.
- Look at exploring other predictive classification models, such as building a neural net in pytorch.
- Explore dimensionality reduction techniques (e.g., PCA, SVD, UMAP), resulting latent features, and improve understanding of feature importance.
- Experiment with other sampling techniques (e.g., SMOTE) to deal with the class imbalance (small number of churn events in the log data) to improve the model.
- Make additional classification predictions, such as upgrades and downgrades.
- Create more visual representations of the data and model results for exploration.
- Move to Spark cluster, run against the full 12GB data, and compare results.

Conclusion:

In this post, I explored a streaming music provider's user activity log data to make predictions on when a user will cancel their subscription (i.e., "churn"). To do this I had to cope with time binning/engineering features and handling class imbalance (only about 7% of the time bins had a "churn" label). I sampled the data and used area under precision-recall curve (AUC PR) as an evaluation metric for building the classification models. I explored three classification techniques: logistic regression (LR), random forest (RF), and gradient-boosted trees (GBT) to predict three engineered labels: (A) churn in time bin, (B) churn in next time bin, and (C) will churn soon (i.e., either A or B).

When analyzing the model and label accuracies- it was apparent that GBT was the best classification technique and label B (churn in next time bin) was the label that had the highest predictive accuracy. When exploring feature weights, it appears that a user making changes to their settings, negatively reviewing songs, and receiving advertisements were the most important features in predicting this label (most all of the other features had a zero weight). This was just a small part of this project- there are numerous improvements and R&D listed in the 'Next Steps' section that can be done to further expand on this work. Being able to predict in advance of user churn is valuable, as it will enable companies to better understand why users churn/make improvements to the platform as well as for them to employee tactics (offer promotions/incentives) to keep these users from churning.