Predicting Customer Churn for a Music Streaming App

1. Business Problem

The problem to be solved in this project is how to accurately predict user churn for the KKBox subscription service. Churn refers to when a customer has discontinued a subscription service. For a subscription based business to be successful its churn rate has to lower that its rate of new customer subscriptions. Specifically, they want to forecast if a user makes a new service subscription transaction within 30 days after the current membership expiration date. It is important to note the temporal component of defining and evaluating churn. Since churn rates are specific to certain time windows, the model features must be tuned to the time window in question, and it is necessary that the time window for the training data be different than the time window one wants predictions for. In this case, the training data is for churn in the month of March 2017, and the test data is for churn in the month of April 2017. The evaluation metric for this challenge is the log loss of the predicted probability of churn for all the users in the test set.

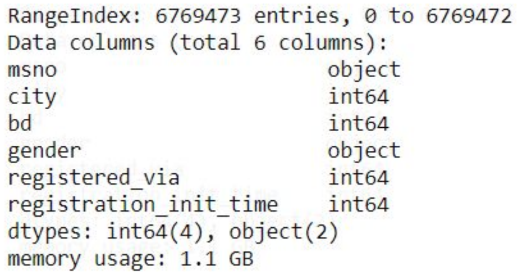
2. Client

KKBox is a music streaming app based in southeast Asia and mainly targets the music market for Mandarin speakers in Taiwan, Hong Kong, Malaysia, Singapore, and surrounding areas. They are a subsidiary of a Japanese telecom corporation called KDDI. KKBox presented this challenge as a Kaggle competition and provided anonymized user data to do so. This is valuable to the marketing team for applying targeted customer retention efforts and to analyze what factors are causing customers to churn which could be used for product improvement. Predicting the future is never easy but with such a large sample of user data it should be possible to use a machine learning algorithm to classify churners and non-churners using insights from their user data. An accurate model will help the business identify customers who are likely to churn.

3. Description Of Data

The data provided by KKBox consists of 5 csv files labeled: train, members, transactions, users logs and sample submission. The train file has two columns, the anonymized user id, and a binary ‘is churn’ value. There are about 993 thousand rows/users in the train file and they were all selected as having memberships that expired in February 2017. All other information about the users is jumbled somewhere in the members, transactions and user logs files. The sample submission file (evaluation set) is 970,000 rows/user who have been selected for having subscriptions that expire in March 2017. So correctly stated, the train set is for user churn in March 2017, and the test set if for user churn in April 2017.

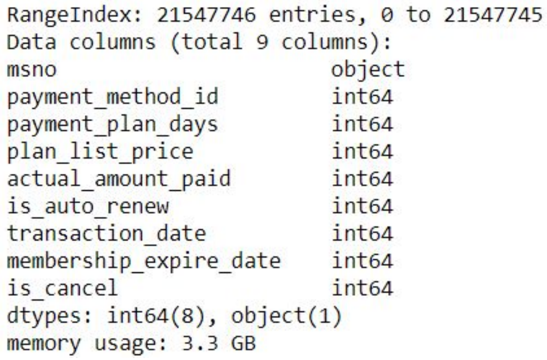
The Members File:



```
RangeIndex: 6769473 entries, 0 to 6769472
Data columns (total 6 columns):
msno                        object
city                        int64
bd                          int64
gender                      object
registered_via              int64
registration_init_time      int64
dtypes: int64(4), object(2)
memory usage: 1.1 GB
```

The members file has almost 6.8 million rows/users and 5 columns of information about each anonymized user id. These columns are city, age, gender, registration method and date of initial registration. About 4.5 million of these users have NaN values in the gender column. There are also about 4.5 million users with a value of zero in the age column and well as about 6,000 outliers that make for impossible ages such as -5 or 873.

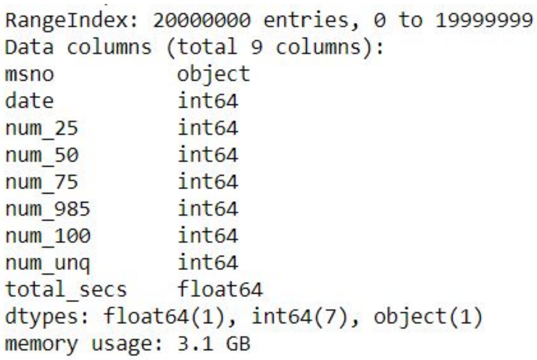
The Transactions File:



```
RangeIndex: 21547746 entries, 0 to 21547745
Data columns (total 9 columns):
msno                        object
payment_method_id           int64
payment_plan_days           int64
plan_list_price             int64
actual_amount_paid          int64
is_auto_renew               int64
transaction_date            int64
membership_expire_date      int64
is_cancel                   int64
dtypes: int64(8), object(1)
memory usage: 3.3 GB
```

The transactions file does not have one row per unique user. It is a 21.5 million row by 9 column csv file that has about 2.4 unique users. Each row is a specific transaction that one user made and since KKbox subscription typically renew monthly there is a transaction row for each month the user has been a member. Columns of the transaction file include: payment method id, payment plan days, plan list price, actual amount paid, 'is auto renew' binary values, transaction date, membership expire date, and binary 'is cancel' values.

The User Logs File
(first 20 million rows):



```
RangeIndex: 20000000 entries, 0 to 19999999
Data columns (total 9 columns):
msno          object
date          int64
num_25        int64
num_50        int64
num_75        int64
num_985       int64
num_100       int64
num_unq       int64
total_secs    float64
dtypes: float64(1), int64(7), object(1)
memory usage: 3.1 GB
```

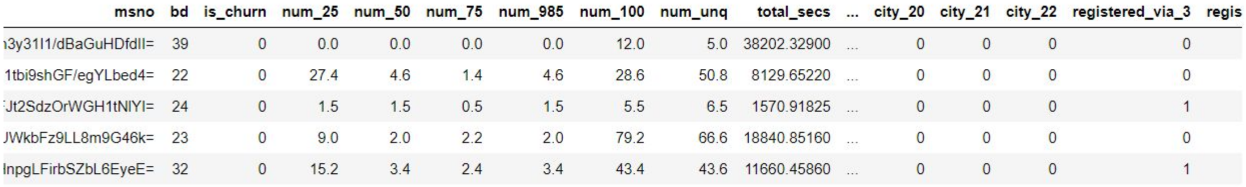
Each row of the user logs file contains data about one day of one user's usage of the KKBox app, so it turns out to be a massive file that is difficult to work with. Even the transactions file can present memory problems at times at a size of 1.6 GB, so the user logs file at size of 28.4 GB certainly requires special treatment. The simplest work-around is to just take the first X million rows of the user logs file and work with that data. The information in the columns of the users log file includes counts of songs listened to grouped by percentage of song played, number of unique songs played, and total seconds played for that day.

So the scope of the user information available both seems somewhat limited but also overwhelming in quantity and shape. Perhaps KKbox has other data available about its users such as what variety of music users listened to, but they did not present that and it would be proprietary so my analysis will be limited to these datasets. Instead of looking for other data I focused on organizing what I have and subsetting it into a usable size dataframe.

An approach that I took was to focus the user log data instead of the transaction data. The rationale being that it should be easier to discern differences in the listening behavior of churners vs non-churners and that listening behavior would have more predictive power. I tried cleaning and wrangling the data in a variety of ways and the latest iteration went as follows in the next paragraph.

First, I read the members csv and apply a boolean filter to exclude members with nonsensical ages. Then merged the members and train set with an inner join on the user id column to create a dataframe called df. Then I read the first 40 million rows of the user logs csv and selected only the entries for users that existed in df. I changed the string values in the date column of the user log into datetime objects so I could filter out specific date ranges of logs. Then as a shorthand way to limit memory usage I saved that abridged version of the user logs as a csv file that could be opened in a new notebook with a fresh kernel.

Abridged example of wrangled and aggregated data to be used for model training

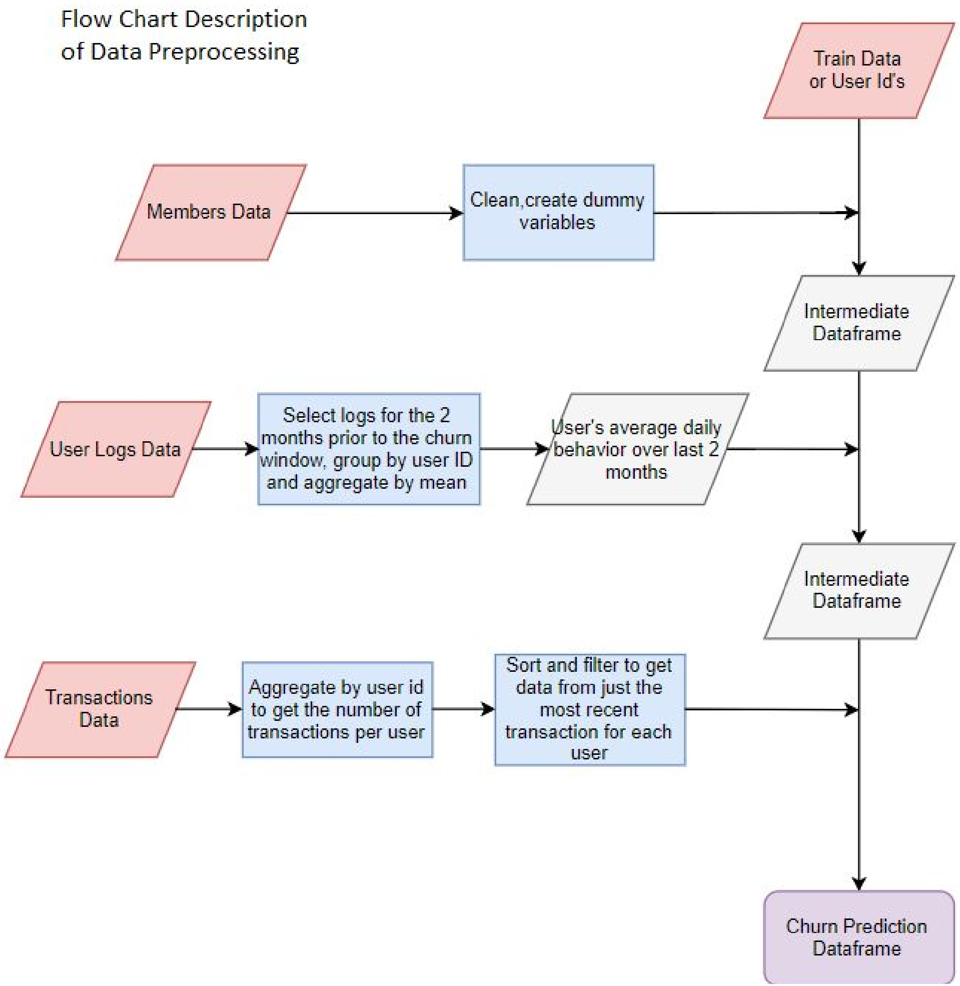


msno	bd	is_churn	num_25	num_50	num_75	num_985	num_100	num_unq	total_secs	...	city_20	city_21	city_22	registered_via_3	regis
3y31I1/dBaGuHDfdII=	39	0	0.0	0.0	0.0	0.0	12.0	5.0	38202.32900	...	0	0	0	0	
1tbi9shGF/egYLbed4=	22	0	27.4	4.6	1.4	4.6	28.6	50.8	8129.65220	...	0	0	0	0	
Jt2SdzOrWGH1tNIYI=	24	0	1.5	1.5	0.5	1.5	5.5	6.5	1570.91825	...	0	0	0	1	
JWkbFz9LL8m9G46k=	23	0	9.0	2.0	2.2	2.0	79.2	66.6	18840.85160	...	0	0	0	0	
InpgLFirbSZbL6EyeE=	32	0	15.2	3.4	2.4	3.4	43.4	43.6	11660.45860	...	0	0	0	1	

In order to convert the daily user logs into a form that can be used with some of the sklearn machine learning algorithms, ie. one row for each user with multitude of feature columns, I used the group-by method on each data column, grouping by the user-id and using the mean as the aggregation function. So this turned the daily user behavior into the average daily user behavior for whatever time period I filter for. I then took the 'mean user logs' then did another inner merge on user id with df. Since this resulting dataframe a few column of categorical data I used the pandas 'get dummies' function to convert then into additional columns with binary values.The dummy variables also dealt well with instances incomplete categorical data. For example if there was an NaN value in the gender column, it was replaced by a zero and then became the 'gender_0' binary dummy variable.

It became clear that the transactions data would be necessary to create meaningful features. The way that I ultimately dealt with this datafile was first to create one feature for the number of transactions on file a user has. Then I took the most recent transaction that each user had and added each column of that transaction as a feature for that user.

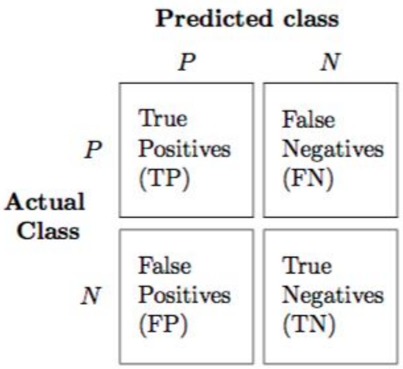
Since the totality of the user data was too large to process and model using the methods taught in this course and the memory constraints of my machine I settled on using a sample of the user logs data which was about 25% of the total records and filtering out by date to look at the user behavior in just a 2 month period before churn window. The rationale being that churners and nonchurners would be likely to behave differently leading up to their decision on whether to continue their service. This average behavior over the last two months is not an ideal approach from a feature engineering standpoint, but should be enough to extrapolate out a significant model.



Model Selection and Performance

After wrangling comes the business of constructing a prediction model. Since the goal was binary classification my first choice of algorithms was logistic regression. Additionally, since the evaluation metric KKbox chose was the log loss this again suited logistic regression since a logistic regression fit is essentially a probability curve. However I knew that random forest classifiers also tend to work very well with binary classifiers since with only two classes, the branching and computational strain does not become too extreme.

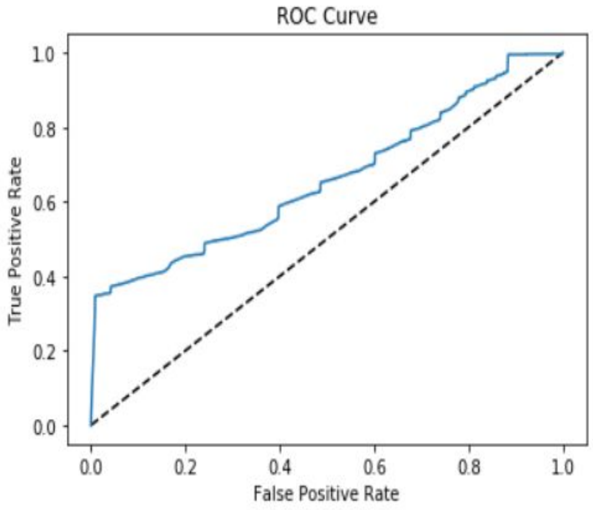
For reference here are what the values in a confusion matrix for binary classification correspond to



		Predicted class	
		P	*N*
Actual Class	*P*	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

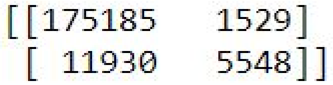
My default logistic regression model scored on the holdout set with a ROC AUC of 0.66 and log loss of 0.244. On the competition submission scoring set its log loss was **0.160**. Bear in my mind that the log loss of just randomly guessing would be 0.693 and an absolutely perfect model (no such thing) would be 0.0.

Model Performance: Unoptimized Logistic Regression



```
AUC: 0.6603846389869884
```

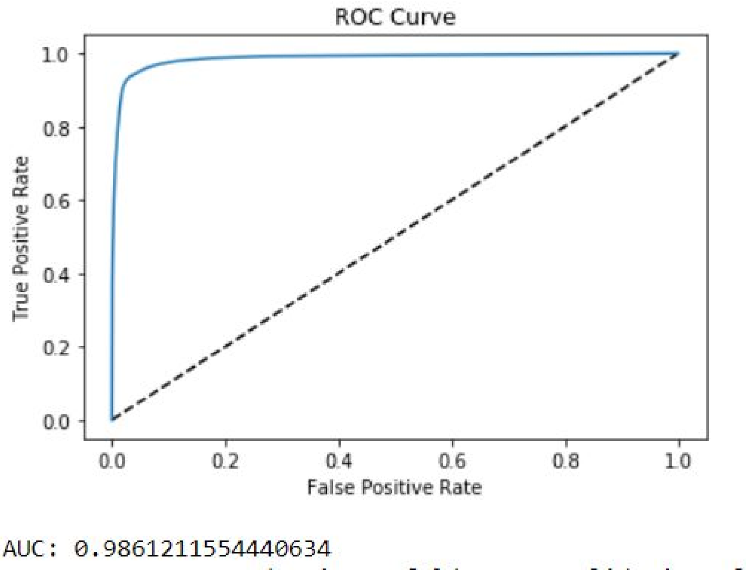
Confusion Matrix



```
[[175185   1529]
 [ 11930   5548]]
```

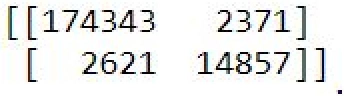
Next I tried a random forest classifier with 100 classifiers and got a very high ROC AUC of 0.986 and a log loss of 0.09 on the holdout set. However, on the competition scoring set it had a log loss of **0.307**.So this was clearly an example of overfitting since it did extremely well on the train set but not so well on the test (contest submission not hold out) set.

Model Performance: Random Forest Classifier



```
AUC: 0.9861211554440634
```

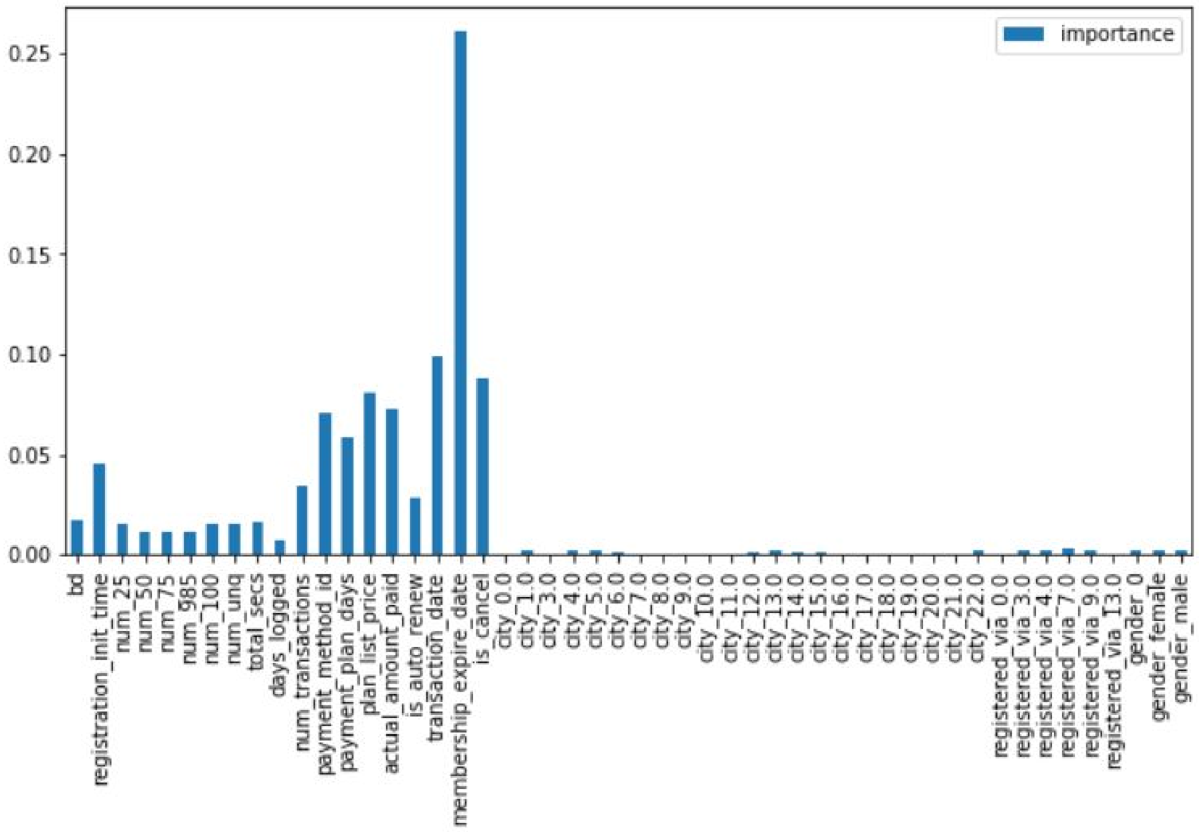
Confusion Matrix



```
[[174343   2371]
 [  2621  14857]]
```

One thing you should always look at with tree based classifiers is the feature importance. How exactly feature importance is calculated will vary depending on which package/ algorithm you’re using but essentially, random forest and gradient boosting algorithms use an ensemble of decision tree classifiers and the feature importance is approximately how often that feature factored into all the models in the ensemble. When trying analyze which variables are significant for users to churn or not churn it can be worthwhile to review this importance metric.

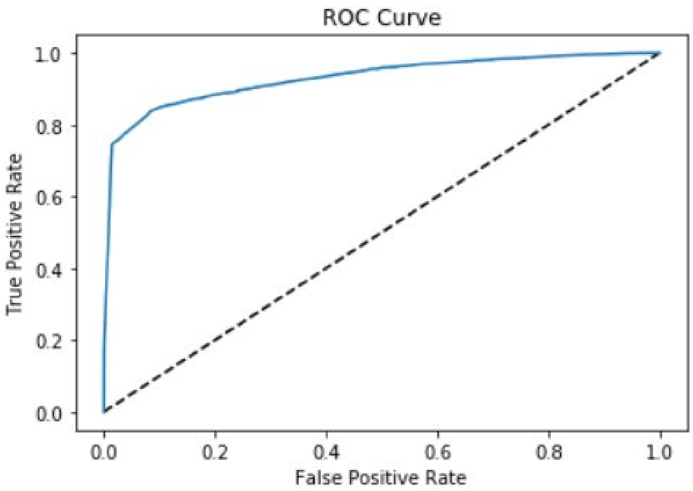
Random Forest Feature Importance

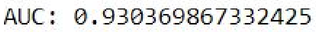


This also brings up the issue of feature selection. It is standard practice when constructing regression models, particularly linear regression, to attempt to avoid multicollinearity, that is, eliminate features that are highly correlated with each other. A standard approach is to calculate the variance inflation factor (VIF) for each feature and iteratively eliminate high VIF feature from the model and re-run the regression. I was instructed to do this but it did not seem to help and since this was a situation where I could check my competition score I could see definitively what did or did not not improve my model. It turned out high VIF feature are not a concern for dummy variables.

Next I tried using a cross validated grid search on logistic regression to attempt to find the optimal hyperparameters. Again this resulted in some overfitting where the model had ROC AUC and log loss of 0.93 and 0.134 on the hold out set but log loss of **.0270** on the test set.

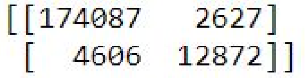
Model Performance: Logistic Regression with Grid Search Hyperparameter Optimization





```
AUC: 0.930369867332425
```

Confusion Matrix

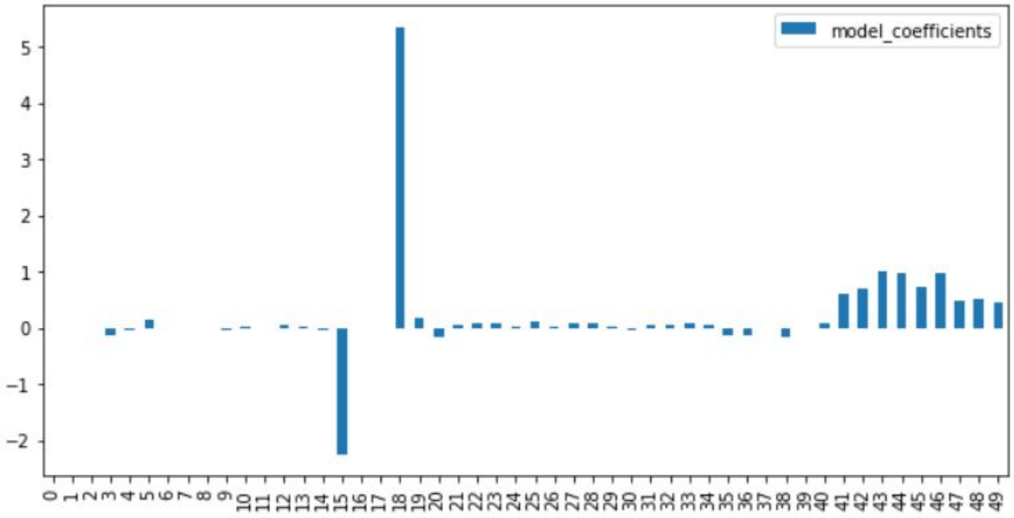


```
[[174087   2627]
 [  4606  12872]]
```

One thing the grid search did highlight though is how using the ‘L1’ as the choice of the penalty parameter can improve fitting. These parameters deal with how the loss term is calculated with L1 referring to least absolute shrinkage and selection operator (LASSO) regression and L2 referring to ridge regression. I remembered a technique briefly described in this course where LASSO regression can be used for feature selection. An aspect of LASSO is with its penalization it shrinks the coefficients of some variable toward zero.

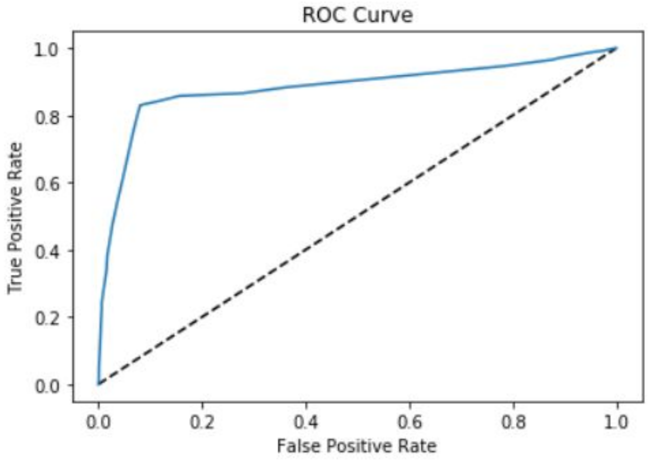
LASSO Regression Feature Coefficients

(numbers correspond with above figure of RF feature importance)



So what I did is use LASSO regression then just filtered out the variables with significant magnitude coefficients and then ran ridge regression on just those variables. This resulted in model that had ROC AUC and log loss of 0.88 and 0.174 on the holdout set and a log loss of **0.137** on the test set. This was my best model thus far and I decided this would be the final one I use for the purposes of this project.

Model Performance: Final Model



```
AUC: 0.8817120310353784
```

Confusion Matrix

```
[[173671   3043]
 [ 10788   6690]]
```

Conclusion

The model I constructed can be used by the hypothetical client KKbox to identify users with a high probability to churn. This would be of great use to marketing team for deploying targeting marketing campaigns to increase customer retention, which is one of the primary goals of any subscription based business. Additional analysis of which features relate to customer churn could prove quite valuable for implementing product improvement measures.