KKBox Churn Rate Prediction: A Special Focus on Imbalanced Dataset

Zihao Xu, Alex Gui, Minh-Quan Do

December 16, 2017

Abstract

In this paper, we aim to document our methodologies in approaching the KKBox Churning Prediction Challenge. This challenge is essentially a classification problem, but the response variable is highly imbalanced. In the below sections, we will describe and visually explore the data sets. Then we will talk about several machine learning models we employed that are highly suitable for handling imbalanced data. Our current ranking on Kaggle is 136 out of 535, achieved by the XGBoost model.

Contents

1 Introduction

Churn rate prediction is a topic of major focus in today's marketing strategy, as many data-driven businesses look for insights into their customers' behavioral patterns, hoping to discover ways to retain their customers based on this data. In the context of customer base, churn rate refers to the proportion of customers or subscribers who leaves their suppliers or services during a given time period.

In this paper, we investigate different ways to tackle challenges in churn rate prediction analysis, such as messy and unwieldy data and imbalanced response variables, as well as implementing different predictive techniques like Weighted Random Forest, Autoencoder, and Extreme Gradient Boosting to come up with the most effective model for our dataset.

2 Problem & Motivation

The problem we try to address comes from a Kaggle Competition entitled KKBox Churn Rate Prediction Challenge. In this competition, we obtain 10 datasets from the company KKBox, a music streaming service in Asia. Our objective is to come up with a statistical model that would best predict whether or not a customer will churn after subscribing to KKBox's services.

Besides, handling and building effective machine learning models with imbalanced data is a common problem faced by data scientist in different industries such as credit risk analysis and churn prediction. Exploring and understanding the techniques and mechanisms behind the new machine learning models will also be beneficial to our future data science endeavors.

3 Data Description

3.1 Overview

The data is donated by KKBox which can be downloaded from the following link:
https://www.kaggle.com/c/kkbox-churn-prediction-challenge/data

The details are given in the tables below:

Data	Description
members.csv members_v2.csv	Member information Latest member information update
train.csv train_v2.csv	The training set. Contains User ID and whether they have churned New churn data in March 2017
transactions.csv transactions_v2.csv	Transactions of users up to 02/28/2017. New transaction data until 3/31/2017
user_logs.csv user_logs_v2.csv	Daily user logs that describe user listening behavior New user log data until 3/31/2017
sample_submission_zero.csv sample_submission_v2.csv	Test set that contains user ID New test data until April 2017

Table 1: Descriptions

Data	Variables
members.csv members_v2.csv	user_id,city, age, gender, registered_method, registration_init_time
train.csv train_v2.csv	user_id, is_churn
transactions.csv transactions_v2.csv	user_id,payment_method_id, payment_plan_days, plan_list_price, actual_amount_paid, is_auto_renew, transaction_date, membership_expire date, is_cancel
user_logs.csv user_logs_v2.csv	user_id, #of songs played less than 25/50/75/98.5/100 % of song length, num_unq,total_secs
sample_submission_zero.csv sample_submission_v2.csv	user_id, is_churn

Table 2: Variables

Note:

1. All the version 2 data are in the format as the original ones but provide additional data on the latest month.
2. The features in members, transaction and user logs will be our predictors while 'is_churn' is the label that we are predicting.
3. A member can have multiple transaction and log data

3.2 Imbalanced Data

The most important problem as well as the focus of this project is the imbalanced nature of the data. In our training set, the response class has a ratio of 15:1 (no churn v.s churn). Therefore, conventional models (decision trees, random forests) and evaluation metric(prediction accuracy) will fail our purpose since it is the minority class (churn users) that we truly care about and with the majority of training data on majority class, the model will very much likely be unsuccessful. We will go deep into how we deal with this problem in the coming sections.

is_churn	percentage
0	93.607713
1	6.392287

Figure 1: Churn Rate in Training Data

4 Exploratory Analysis

4.1 Data Handling

With some exploratory analysis, we have found numerous problem with the data [1]:

1. All the categorical variables are encoded numeric. They need to be transformed into factor variables for our visualization.
2. There exist significant outliers. For instance, age range from as low as negative to over 1000. The min of total seconds played goes as low as to -9.223e+15.
3. Half of the gender information is missing
4. The data sets are tremendous. user-log particularly has 30 GB which makes processing difficult.

To handle the big data, we read 1 million rows from each data set which is about 20% of the member data, 5% of transaction and 1% of user log. To clean the data for visualization, we have factor coded all the numeric categorical variables, imputed missing values and outliers with 0 and the average. In the visualization of a particular variable, we filtered out outliers associated with that variable. Additionally, all the dates have been transformed into date format. Finally, we join the data sets that contain user feature information to explore the relationship between certain features and churn rate.

4.2 Visualization of Individual Data Sets

4.2.1 User Demographics

Recall that 'members.csv' has features: user id, city, age, registration method and registration initial time. In this section, we want to explore user demographics: who are KKBox's user? What is their age and gender? Where do they come from and through what channel they get to subscribe to KKBox? We are also curious about the development of KKBox as a company: what is their popularity through out the time and at that time they gained their users bases.

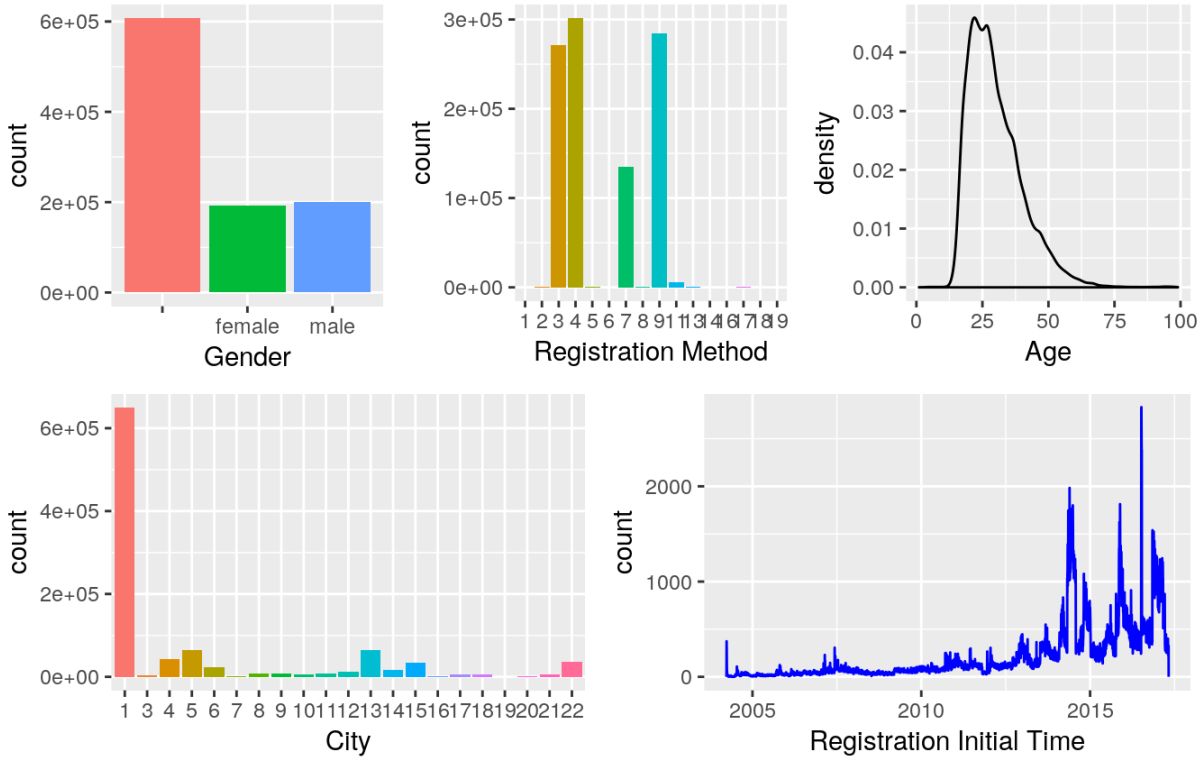


Figure 2: User Demographics

[1] Only version 1 of the data are used there except for members which is v3

A quick glance at age, city, birthday distribution and time plot of user registration time. We observe that

1. Gender distribution is pretty equal
2. The majority of users come from city 1.
3. The major registration method is 4,9,3
4. a spike of user accumulation took place in 2014

Additionally, we want to explore the relationships among the features

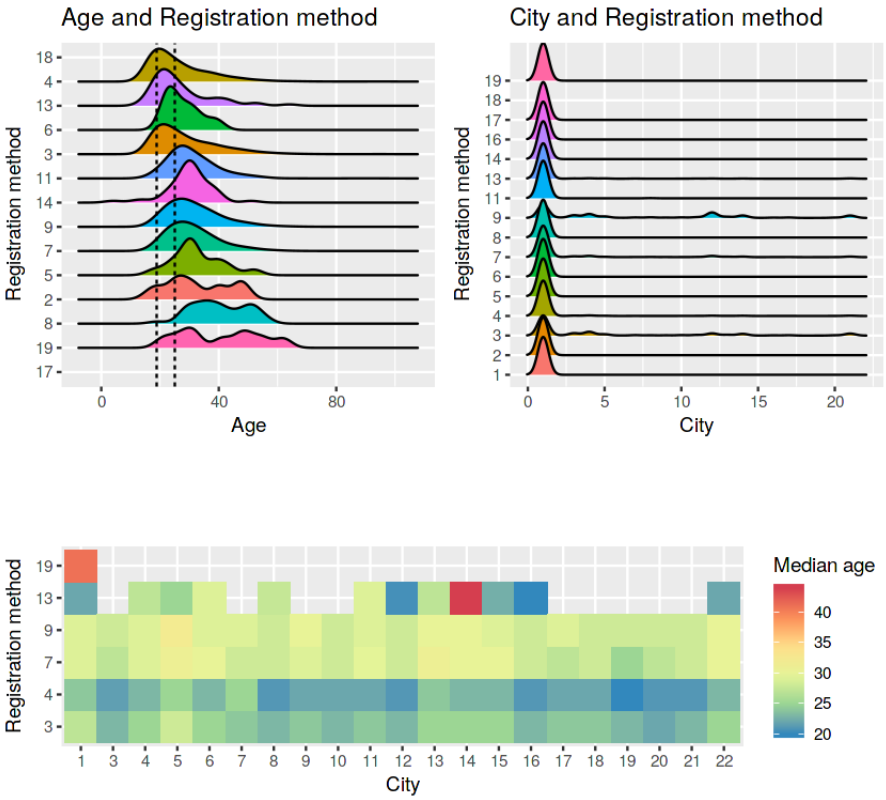


Figure 3: Age-City-Registration Method

Observe that

1. Young people mostly register through method 4 and 13. Older users mostly prefer method 7 and 9.
2. Registration distribution does not vary much among the cities (the reason might be city 1 users dominate the data)

4.2.2 Transaction Data

Recall that transaction data has features: payment method, payment plan, list and actual price, whether the user uses auto-new, transaction and expiration date and whether they have canceled.

Similar to what we've done to member data, we first use bar plot to visualize the distribution of transaction behavior.
However, note that this visualization used unprocessed data which means each observation is a transaction not a user. Therefore, some effects might be exaggerated: for example the percentage of auto renew is higher than that of actual users because users with more transaction are more likely those with auto-renew.

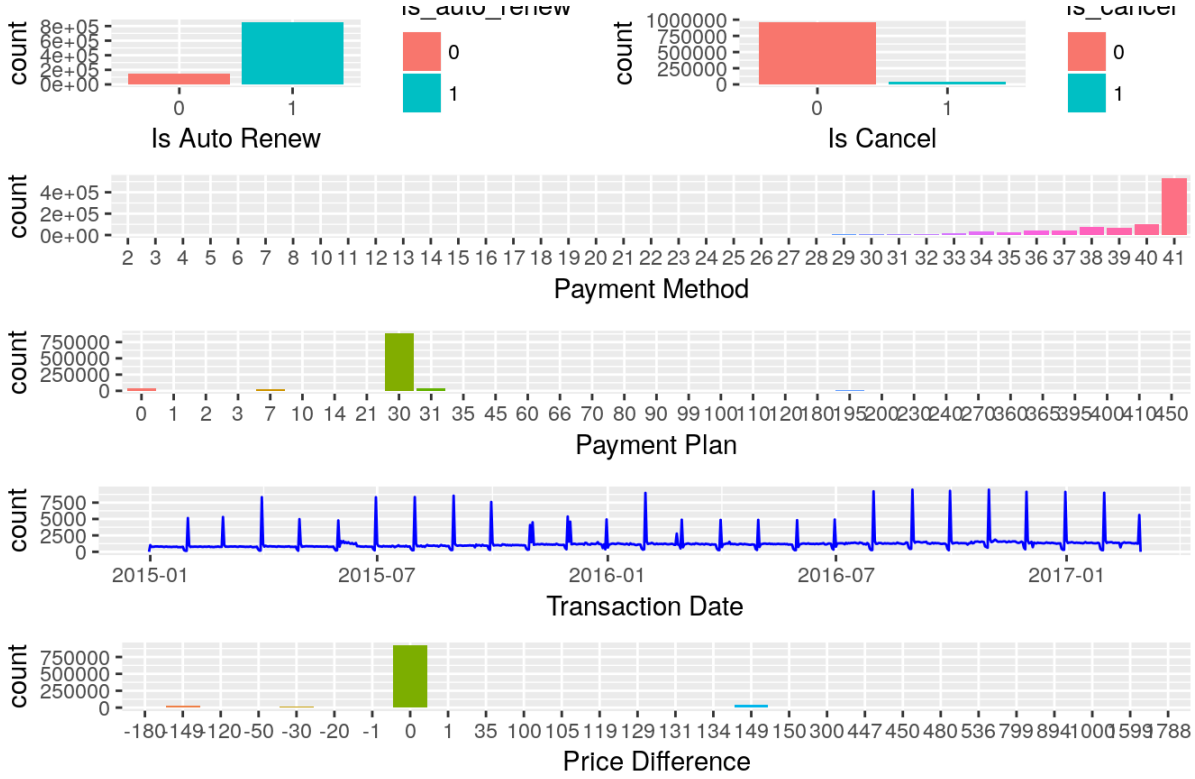


Figure 4: Transaction Behavior

Observe that

1. Monthly subscription is the most popular plan
2. Payment method 41 is the most popular and dominant.
3. There are numerous cases when the actual amount paid and list price is different.
4. The pattern of transaction data fits the monthly subscription picture. Note that an increasing amount of transaction took place in 2017 which corresponds to the user spike in 2017 shown in the member data set.

In the coming sections, we will explore how these variables are related to churn rate.

4.2.3 User Log

User log has features: user id, number of unique songs played, number of less than 25%-100% song length played and total seconds played.

Observe that users on average have 1-16 log entries while most users have one entry per day. Total seconds of songs played is skewed but normal in the right tail. The number of unique songs played centers around 20.

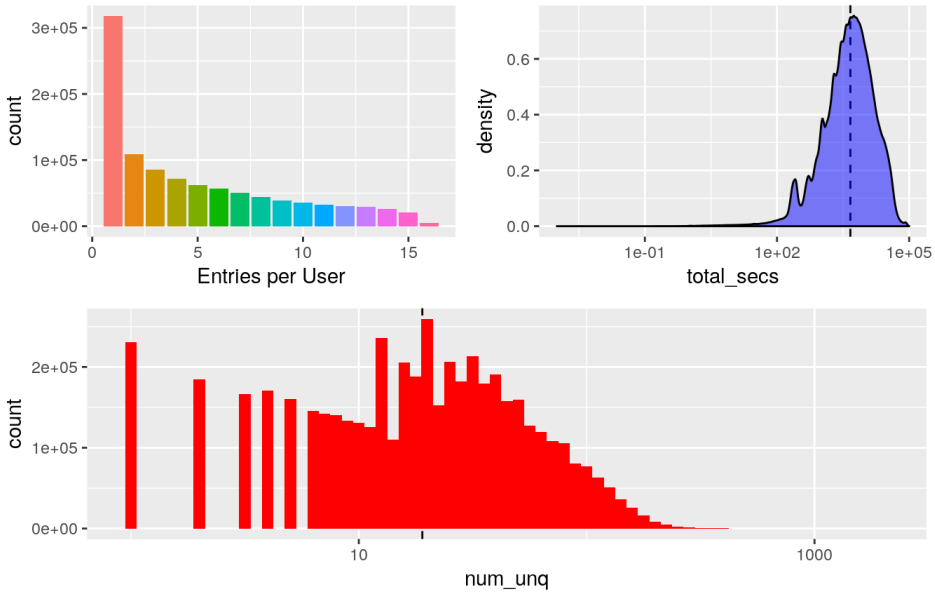


Figure 5: User Listening Behavior

4.3 Relationship Between Predictor Variables and Churn Rate

In this section, we want to explore: what kind of users are more likely to churn? This analysis will provide important foundation for our model building as it gives us an idea of what features are closely related to our prediction. In this section, we merged the data sets with user features into one training set for our analysis.

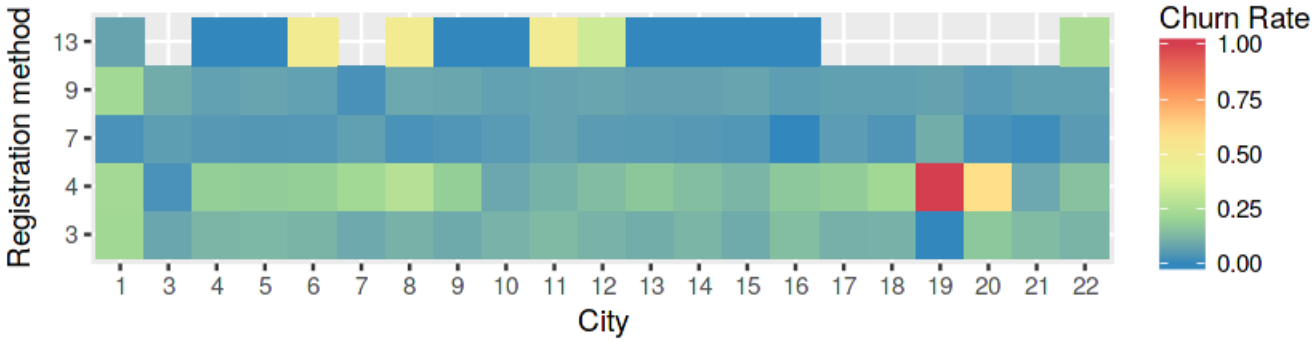


Figure 6: City, Registration Method and Churn Rate

Observe that users registered from method 4 and 3 seem to have higher churn rate. City 19 method 4 probably has only one user which explains churn rate of 100%.

Gender doesn't seem to effect churn while young people (18-28) are more likely to churn.

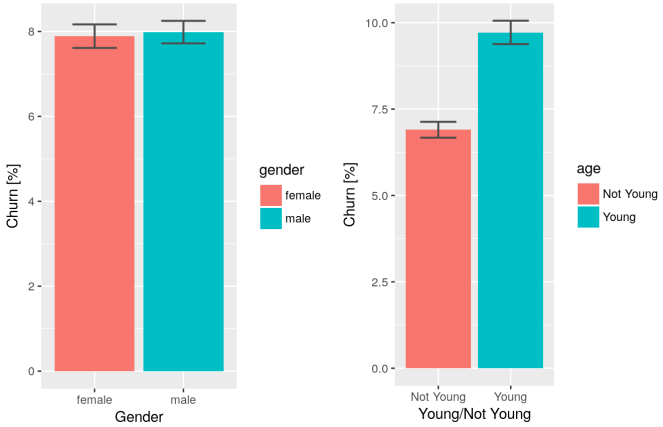


Figure 7: Gender, Age and Churn Rate

Users with more log entries are less likely to churn

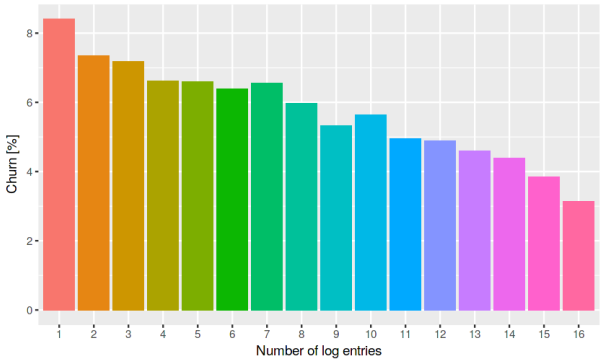


Figure 8: Number of Log Entries and Churn Rate

The plot of payment plan is really interesting: on one hand, it's empirically true that users with standard 30-day subscription rather than 1-day trial are far less likely to churn. However, users with super long subscriptions (400 days) have very high churn rate. This might be due to the fact that very few people choose long payment plan options and one churn will make the effect dramatic?

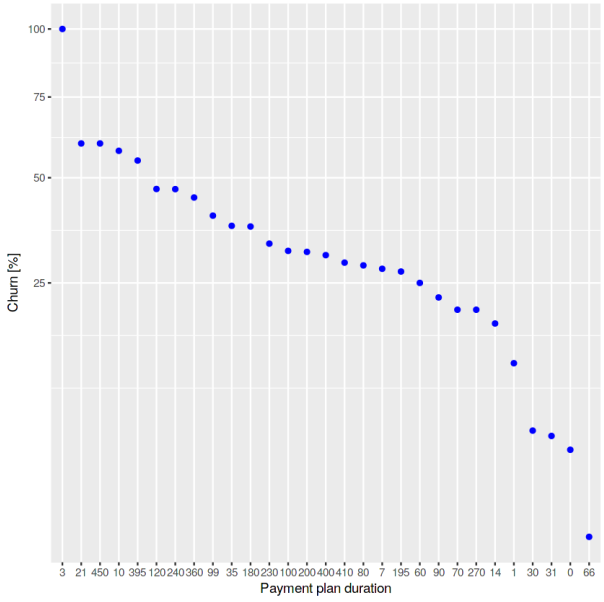


Figure 9: Payment Plan and Churn Rate

Users who do not auto renew (marked as "0") and cancel subscription (marked as "1") are significantly more likely to churn.

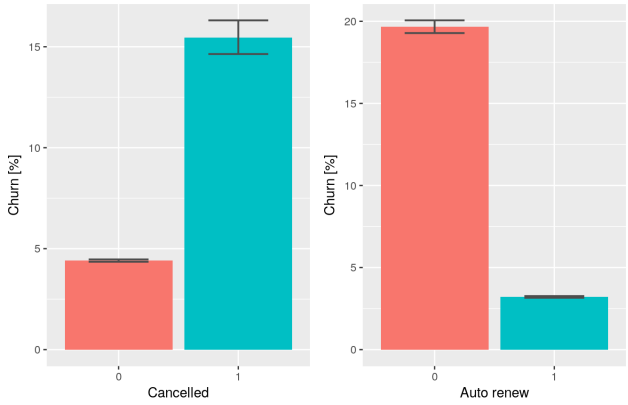


Figure 10: Auto Renew, Cancellation and Churn Rate

4.4 Insights Summary

Exploratory analysis shows us a big picture of KKbox's users and their behaviors. We know that the majority of users are under 30 years old, come from city 1, prefer registration method 4,3, payment method 41 and 30-day standard payment plan. They usually log in once a day, listen to a total of 1000 secs of songs(16.6 minutes) and a median of 20 unique songs. We also learned that there are predictors that are promising to tell us something about churn rate: age, number of log entries, payment plans, and whether the user has chosen auto renew. This foundation is important for the coming model building.

There are lots of space for improvements: what percentage of users who cancel churn eventually? What type of listening behavior is more likely to churn? The data is super interesting and we believe there is a lot more potential than what we have discovered.[2]

[2]Our visualization is inspired by the work of a Kaggle user, heads or tails. The link is attached: https://www.kaggle.com/headsortails/should-i-stay-or-should-i-go-kkbox-eda. It is an amazingly comprehensive analysis which is better than our original plots at many level. Therefore in our analysis, we mostly reproduced his analysis and reflected what way we can improve and make our arguments more concise. The way plots are coded are also different. In a word, we have no intention to take credit for his work but just believes it's a great education

5 Building Machine Learning Models

After discussing the insights we gained from the data sets, we are now ready to move onto building machine learning models to make predictions. Due to the imbalanced nature of the response variable, *is_churn*, machine learning models that works well in normal (predictions of balanced response variables) do not necessarily work well for our purposes. To fully appreciate the imbalancedness of our data set, we have explored three alternative machine learning algorithms: Weighted Random Forest, Autoencoder and Extreme Gradient Boosting.

In the following sections, we will cover how we performed data pre-processing, feature selection and engineering and finally model building and evaluation.

5.1 Data Pre-processing

As mentioned in Section 3, the data sets are readily available on the Kaggle kernel. However, the data sets are not readily usable for our purposes. The problems mainly comes from four areas: time-related, memory consumption, accurate information representation and imputation of missing values.

5.1.1 Time-related

Since the entries in all of our data sets are labeled with a time stamp, working with time is necessary for this data challenge. For time-related pre-processing, we mainly did three manipulations:

First, there exist two versions for all the input data sets (e.g. *members.csv* and *members_v2.csv*), representing independent data from two distinct time periods. This requires us to append the latter data frames to the previous ones to obtain all information available. Second, the dates are represented in a $float64$ format, which we converted into *datetime*64 and from it we extracted the year, month, day and weekday as separate columns. Third, we converted the time representation of date in $float64$ (e.g. 20170228) into a more accurate numeric representation using the following equation:

$$new\_time \;=\; year \;+\; \frac{month - 1}{12} \;+\; \frac{day}{365}$$

In this way, we were able to accurately represent time in a continuous and accurate[3] way.

5.1.2 Memory Consumption

Another common problem for all the input data sets is they are all too big to be directly loaded into memory. To solve this problem, we used the Kaggle kernel to perform our computations and furthermore, we performed memory reduction by changing the data types of each column to its smallest possible format. More specifically, all of the numeric values in the data frame were stored in float64 format, which occupies a relatively large space in memory. To reduce memory, for each column, we look at the range of the values in that column and converted the values to their corresponding smallest possible format (e.g. Int16 $\in$ (-32,768, +32,767), Int32 $\in$ (-2,147,483,648, +2,147,483,647)). In this way we were able to reduce memory consumption drastically.

On top of this, one data set, *user_log.csv*, has size of around 20GB and even the Kaggle kernel could not handle data of such a size: it is too big to be loaded and even if it could be loaded, the time consumption would be quite significant. To tackle this problem, we first applied parallel processing techniques to load in the data from this file in chunks. Then we dynamically append the useful portion[4] of the newly loaded chunk of data, discard this chunk and finally output only the most useful information. This will be further discussed in the following sub-section.

5.1.3 Accurate Information Representation

Besides, *transaction.csv* and *user_log.csv* both contained duplicated entries for each individual user, representing multiple transactions and multiple log-ins. To keep the data set tidy (each row

reference.

[3]Note that the minus 1 for month is because, for example, when we are in January, the first *month* has not actually passed, but only the *days*.

[4]We define useful as being the most recent entry of a particular user. This will be further discussed in the next section.

represents an unique user), we have to find ways to represent a user's information accurately. Ideally, the most accurate representation of a particular user's transaction and log-in data is probably the average of all the entries of the respective columns. However, due to computational limitations, we took the most-recent (in time) transaction and log-in information to represent a user's data. Such representation might be limited and could be further improved if time permits.

5.1.4 Imputation of Missing Values

The problem with missing values is also present in our data sets. we simply imputed the missing values for a particular column by its mean. We also experimented with imputation by median, but did not produce better model performance. We believe that more advanced techniques such as group-by imputation and KNN for imputation could be used to better this process.

5.2 Feature Selection and Engineering

Besides the pre-processing, two more important steps before model training is feature selection and engineering. With them, we are able to eliminate the variables that have no predictive power while creating new variables that are able to extract information hidden in the data sets.

5.2.1 Feature Selection

The data set contains, in total, 27 features and 1 response variable. However, upon further examination, several of the numeric columns, such as *registered_via*, *city* and *payment_method_id* are treated as positive discrete numbers. We first attempted to apply one hot encoding[5] to all such cases but the resulting feature space has more than 100 dimensions. Therefore, we would like to find an efficient way to identify the important values to encode. To this end, we tried several different methods:

Correlation Matrix
First we plotted the correlation matrix (Figure 11) of all the variables, including features and the response.

[5]For more information on one-hot encoding, visit this link.

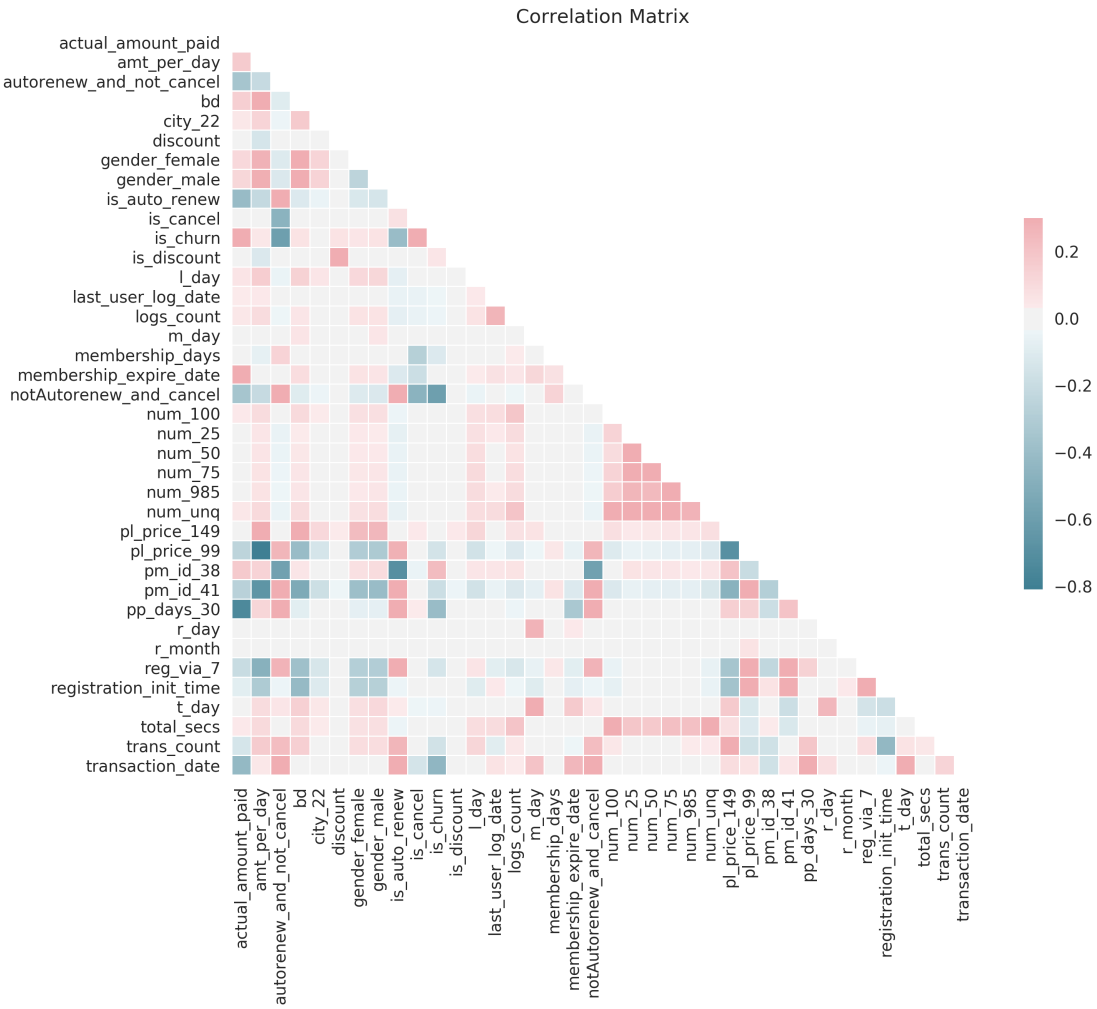


Figure 11: Correlation Matrix Plot

From the correlation matrix plot, we can visually observe that variables such as *actual_amount_paid* (positive correlation) and *auto_renew_and_not_cancel* (negative correlation) are strongly correlated with the response variable, *is_churn*, whereas other variables like *r_month* have basically no correlation with *is_churn*. In this way, we are able to filter out some of the non-correlated variables and reduce the feature space[6].

Counts and Average Churn Rate Plots

We further analyzed the categorical columns to be transformed by looking at the counts of individual values of these columns as well as the average churn rate for each value.

As an example, Figure 12 and Figure 13 show the value counts and average churn rate for each value of the column *registered_via*.

[6]Most of the ones that are filtered our are not in this plot

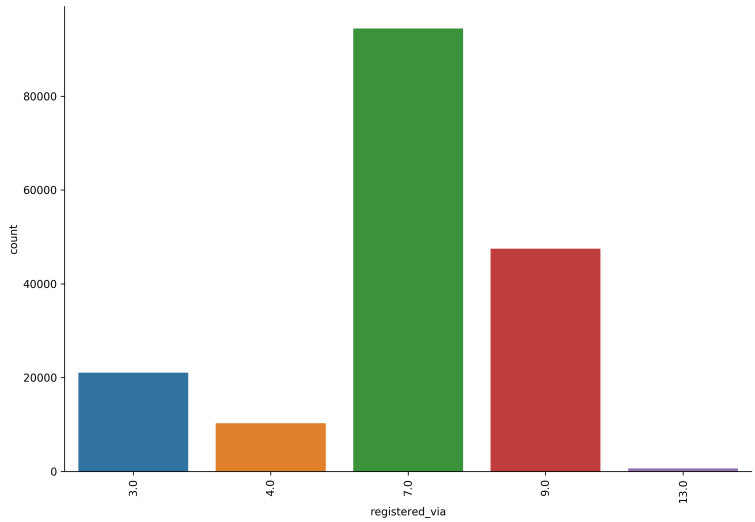


Figure 12: Counts for Unique Values of Column *registered_via*

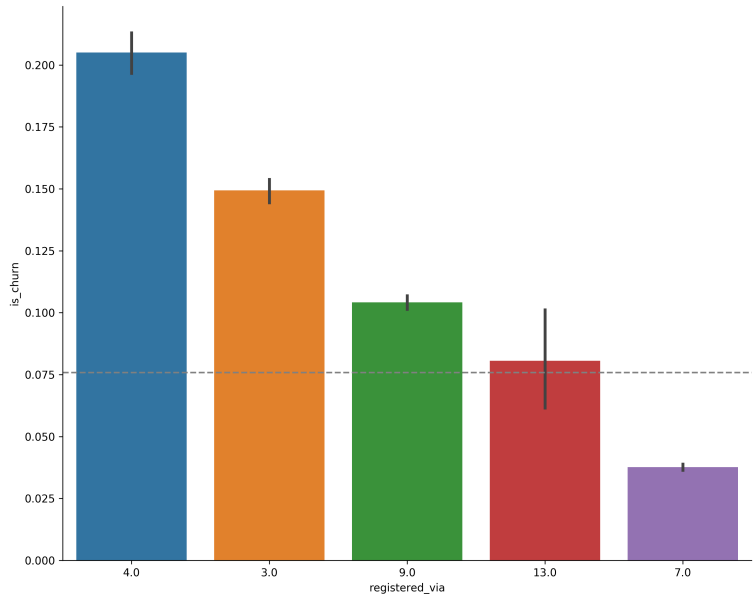


Figure 13: Average Churn Rate for Unique Values of Column *registered_via*

We observe that 7 is the most popular channel through which users register KKBox's service, while users registered through channel 4 tend to have very high average churn rate. Therefore, for this particular variable, we only one-hot-encode *registered_via* equal to 7 and 4. The encodings for other columns are similar to this procedure.

5.2.2 Feature Engineering

Upon further analysis, we have discovered that based on heuristics, we are able create some new variables that extract hidden information in the data. Below is a complete break down of all the new variables that contains predictive power:

discount: Plan list price subtract actual amount paid.
is_discount: Whether the user received a discount, of **discount** > 0.
amt_per_day: Actual amount paid for service divided by days of the plan.
membership_days: Number of days between membership expire date and transaction date.
trans_count: Number of transaction that a user has on record.

logs_count: Number of log-ins that a user has on record.
pm_id_41 & 38: Dummy encoding for *payment_method_id* equal to 41 and 38.
pp_days_30: Dummy encoding for *payment_plan_days* equal to 30.
reg_via_7 & 4: Dummy encoding for *registered_via* equal to 7.
autorenew_&_not_cancel: If a user is using auto-renew and is not a canceled user.
notAutorenew_&_cancel: If a user is not using auto-renew and is a canceled user.

To illustrate the predictive power of the new variables, we have also plotted the average churn rate for individual values for each of the variable. One example, *autorenew_and_not_cancel*, is shown in Figure 14 below:

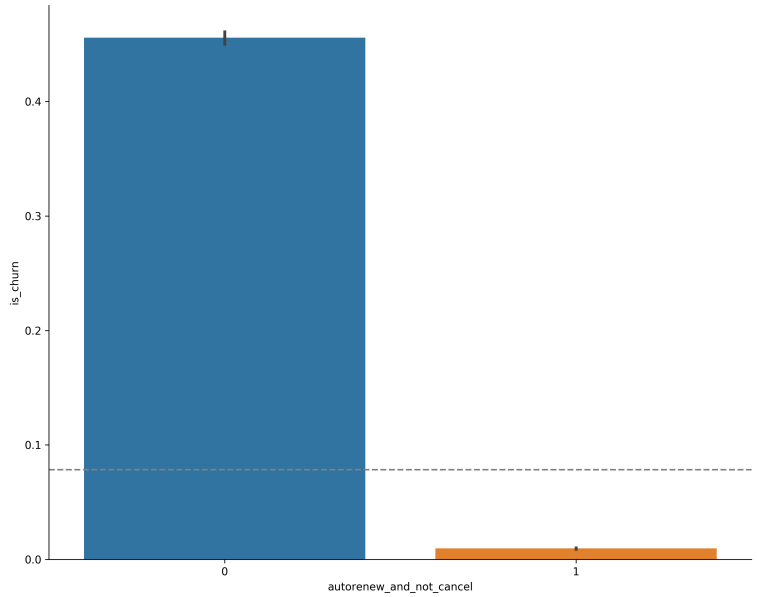


Figure 14: Average Churn Rate for Unique Values of Column *autorenew_and_not_cancel*

Observe that the users that is auto-renew and not cancel has average churn rate far below average while the other group's average churn rate is far above average. Finally, there are, in total, 37 variables[7] included in the machine learning models.

5.3 Model Training

With all the data sets pre-processed and features selected, we divided up the data into training (80% of 1,963,891 observations) and testing sets (20% of 1,963,891 observations). After that, we experimented with three models that can theoretically handle the imbalancedness of the response variable. In the following subsections, we will go into each of the models and its underlying mechanisms.

5.3.1 Weighted Random Forest

In class we have talked about Random Forest in detail, so in this section, our discussion will be restricted into the difference, or enhancement that Weighted Random Forest (WRF) (Chen et al., 2004) has over the standard Random Forest (RF).

WRF is suitable for learning from extremely imbalanced data because of its idea of cost sensitive learning. The essential idea of WRF is to to place a heavier penalty of misclassifying the minority class. Before training the WRF model, we first assigned a weight to each of the observation, and this weight is inversely proportional to the size of the respective class. This is to say that, observations in the minority class are assigned larger weights (i.e. higher misclassification cost), which is then used in the calculation of node homogeneity (e.g. Gini index) in model training (Chen et al., 2004). Furthermore, the prediction of WRF differ from that of standard RF in that,

[7]Sadly, not 47 anymore... Some of the less predictive ones were dropped.

the final prediction in given by a weighted voted - trees voting in favor of the minority class are given more weights (Chen et al., 2004).

The combined effects of assigning higher misclassification cost and more prediction weight to the minority class is that, WRF is more likely than RF to correctly predict observation from the minority class while incorrectly predicting observation from the majority class[8]. And it is confirmed by experimental results that WRF performs better than RF in terms of a weighted accuracy (Chen et al., 2004).

5.3.2 Autoencoder

Another algorithm we explored is called Autoencoder (Tan, 2009), a form of anomaly detection technique that is widely applied to the field of fraud detection. Essentially, the idea of anomaly detection is that we will train the model using only the "normal", or the majority class, observations so that it can capture the patterns of the majority class as much as possible. Then in the prediction phase, the output of each observation can be of two forms: probability of occurring (probability of being "normal" given the input features) or, for algorithms like Autoencoder, reconstruction error, which we will discuss further below.

The training of Autoencoder is of two-folds: in the encode phase, the Autoencoder attempts to encode the input features into a lower dimensional representation (the number of dimensions is a tuning parameter) of the inputs; in the decode phase, the Autoencoder tries to reconstruct as well as possible the original input features based on its lower dimensional representation (Tan, 2009). Figure 15 represents an intuitive way of illustrating this encode-decode process:

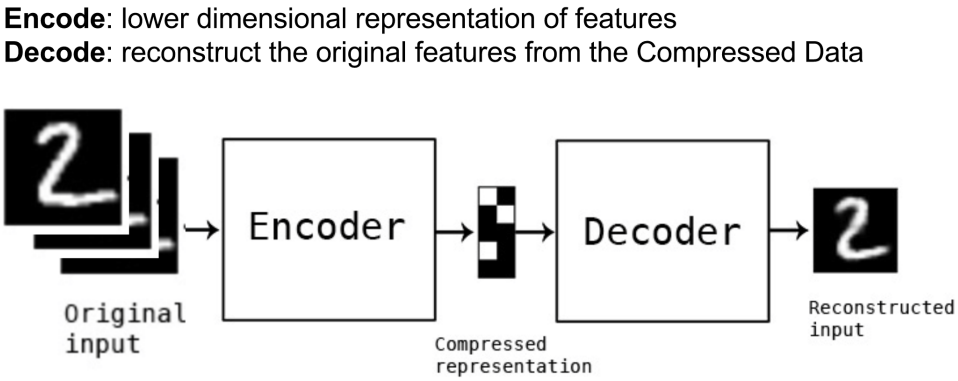


Figure 15: Autoencoder Training

After the model training, we then feed new, unlabeled observations to the Autoencoder with the hope that it is able to recreate observations from the "normal" (in our case, "no churn") class better than observations from the "abnormal" (in our case, "churn") class. A numeric measure called "reconstruction error" is assigned to each prediction, with lower error values representing "closer to being a normal class" and higher error values representing "closer to being an abnormal class". Figure 16 shows the prediction logic of the Autoencoder algorithm.

[8]This is Zihao's own inference.

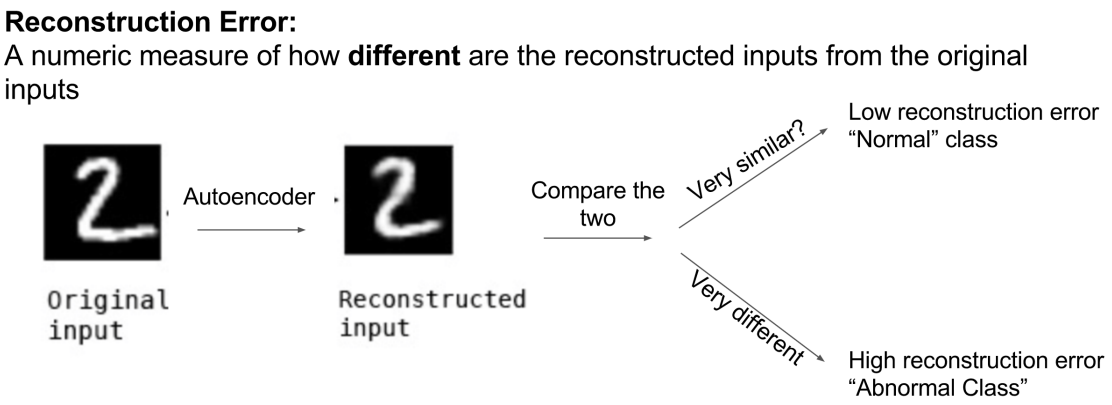


Figure 16: Autoencoder Prediction

In our prediction, we plotted the reconstruction error distribution and discovered that, the distribution of observations from the "churn" class is indeed different from (relatively higher than) observations from the "no churn" class, illustrated below:

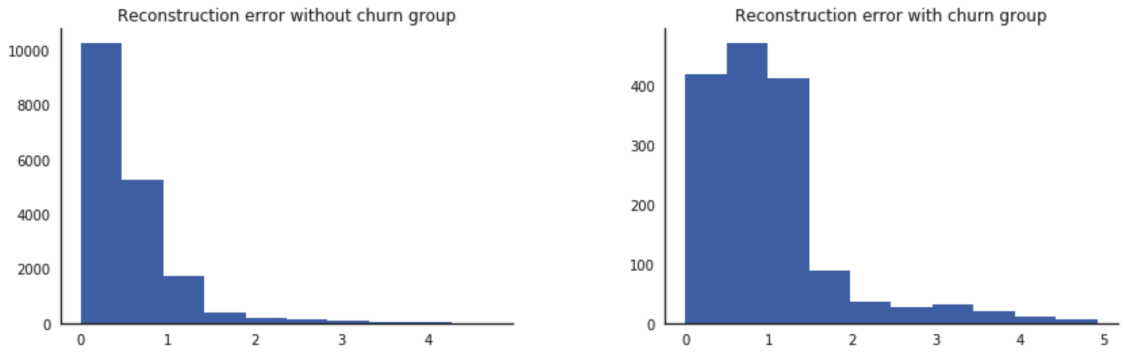


Figure 17: Reconstruction Error Distribution for Observations from each Class

From the difference in distribution of reconstruction error, we are able to set a hard threshold, so that the observations with reconstruction error above the threshold is predicted to be "churn", while the observations with reconstruction error below the threshold is predicted to be "no churn", thereby completing the prediction process. Figure 18 is an graphical illustration of such prediction process:

5.3.3 Extreme Gradient Boosting

One last model that we build is called Extreme Gradient Boosting (XGBoost) (Chen and Guestrin, 2016). Essentially, XGBoost is a particular form of Boosting, an ensemble meta-algorithm that aims to convert a collection of weak models into a strong model. XGBoost usually aggregates two types of weak models: CART (Classification-And-Regression-Trees) and linear regression models. In our particular application, XGBoosted CARTs are used due to their wider applications and superior performance (Chen and Guestrin, 2016). Though the complete algorithm and mathematical properties of this model is beyond our current scope of understanding, it is interesting to note that the response variable for Boosting models in general is not the actual response variable, but rather the residuals of the previous weak learner. This is to say that, each CART (not the first one) is trained so that it is trying to fix the "mistakes" made by the previous CART. It is this fact that makes Boosted models better than the standard RF in that the CARTs inside a RF are independently built (their predictions are not informed by mistakes of each other).

In our analysis, we trained a XGBoost model, using cross validation to determine the point at which the model stops improving (*early_stopping*), but did not use CV to tune the parameters due to lack of understanding of the algorithm.

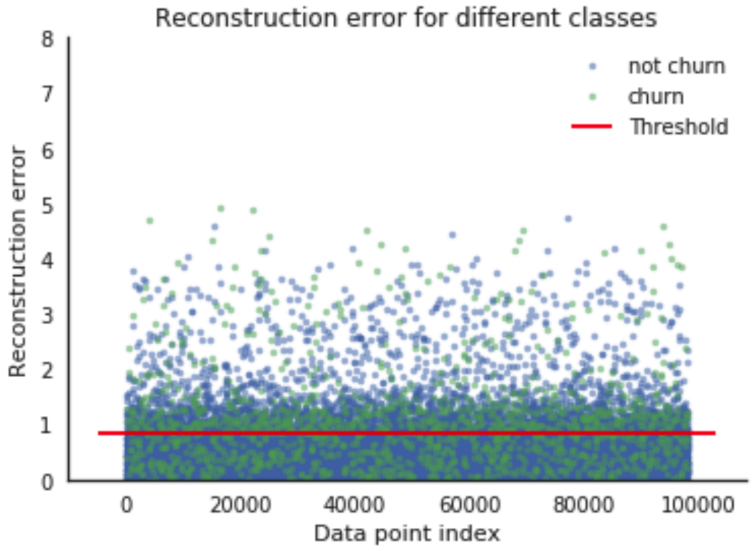


Figure 18: Prediction of Autoencoder using Threshold and Reconstruction Error

5.4 Model Evaluation

After we trained our models, we used several metrics to assess our model performance: log-loss, confusion matrix and AUC score.

5.4.1 Log-Loss

Logarithmic loss, or log-loss, is a evaluation metric used for classification models and a lower log-loss indicate superior performance. It is used by Kaggle competition and therefore should be the primary metric that we aim to optimize. The mathematical definition of Log-loss is as follows:

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij})$$

where N is the number of samples or instances, M is the number of possible classes, y_{ij} is a binary indicator of whether or not class j is the correct classification for instance i, and p_{ij} is the model probability of assigning label j to instance i (Collier, 2015). It is interesting to note that, this evaluation metric assumes the prediction to be in the form of a probability, which means that the prediction does not necessarily have to be a $0/1$. In our particular case, since the response variable is_churn is binary, this formula cam be re-written as:

$$-\frac{1}{N}\sum_{i=1}^{N} y_i log(p_i) + (1-y_i) \times log(1-p_i)$$

This evaluation metric has several properties: 1. the more the misclassification, the larger the log-loss; 2.it does not differentiates between misclassification of an observation from majority class or minority class; 3. it heavily penalizes misclassification with high confidence (which relates back to the fact that predictions are given in probabilities). It is worth pointing out that, due to point 3, it is actually better to assign a prediction of 0.5 when the model is unsure (depend on the context of the algorithm) of a particular prediction.

5.4.2 Confusion Matrix

Confusion Matrix is another common technique for model evaluation. It provides the user with a good break down of predictions versus actual values in the following format:

Actual Class	neg	true negative	false positive
	pos	false negative	true positive
		neg	pos
		Predicted Class	

Table 3: Confusion Matrix

With this tool, we can assess the relative performances across several models in terms of the desired metric (e.g. accuracy, precision, recall and etc.).

5.4.3 AUC Score

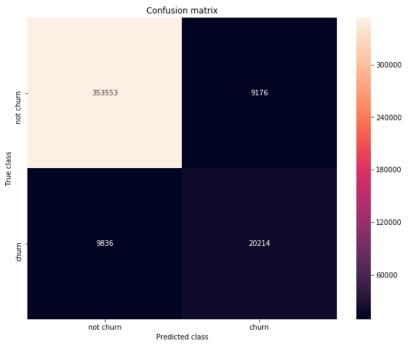
Another useful tool is called the AUC Score, or Area Under the (Receiver Operating Characteristic) Curve. The Receiver Operating Characteristic (ROC) curve plots all possible combinations of True Positive rate and False Positive rate associated with all the possible classification thresholds. In our context, it visually displays the trade-off between proportion of correctly classified churn observations versus the proportion of misclassified churn observations. The large the area under ROC curve, the less trade-off and the better the model.

5.4.4 Final Result

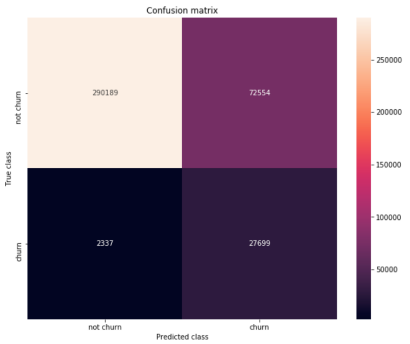
As mentioned in Section 5.3, we have reserved a testing set to evaluate the above three models. Below are the results we obtained.

- Log-loss (Actual log-loss calculated by Kaggle):
  WRF: 1.65977
  Autoencoder: N.A.[9]
  XGBoost: 0.13943

- Confusion Matrix:

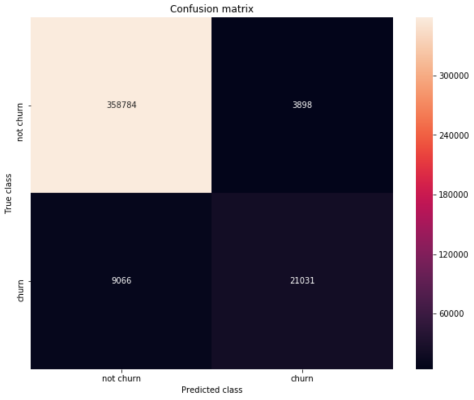


WRF Confusion Matrix



Autoencoder Confusion Matrix

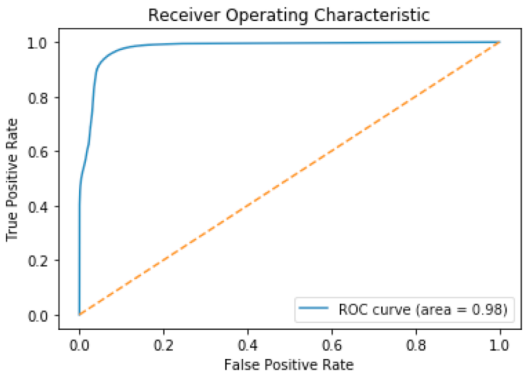
[9]Due to recent update of the *Keras* package, we run into an unknown error, making prediction impossible...



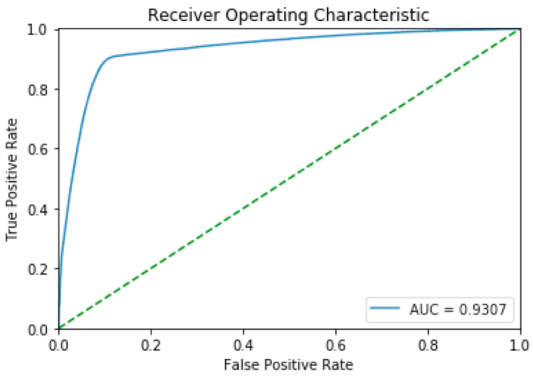
XGBoost Confusion Matrix

Figure 19: Confusion Matrices

- AUC Score:



WRF ROC curve and AUC score



Autoencoder
ROC curve and AUC score

Figure 20: AUC Score

Note that the ROC curve for XGBoost is not included due to coding difficulties.

We can see that, based on the log-loss evaluation metric, XGBoost is our best performing model. WRF is average performing, but still decent, while Autoencoder has great potential as we can see that it correctly classifies most of the churn observations. Finally, we submitted the prediction given by the XGBoost model as our final prediction for this challenge.

6 Conclusion and Future Directions

From this project, we have found that Extreme Gradient Boosting gives us the best prediction when using Kaggle's log-loss evaluation metric. We have submitted our predictions to this competition and is currently ranking at 136 out of 535 teams[10]. The final model is built using XGBoost using heuristically selected parameters, which can be further improved using train-test split and cross validation for parameter tuning. We did not do this step due to lack of understanding of the many parameters that XGBoost has.

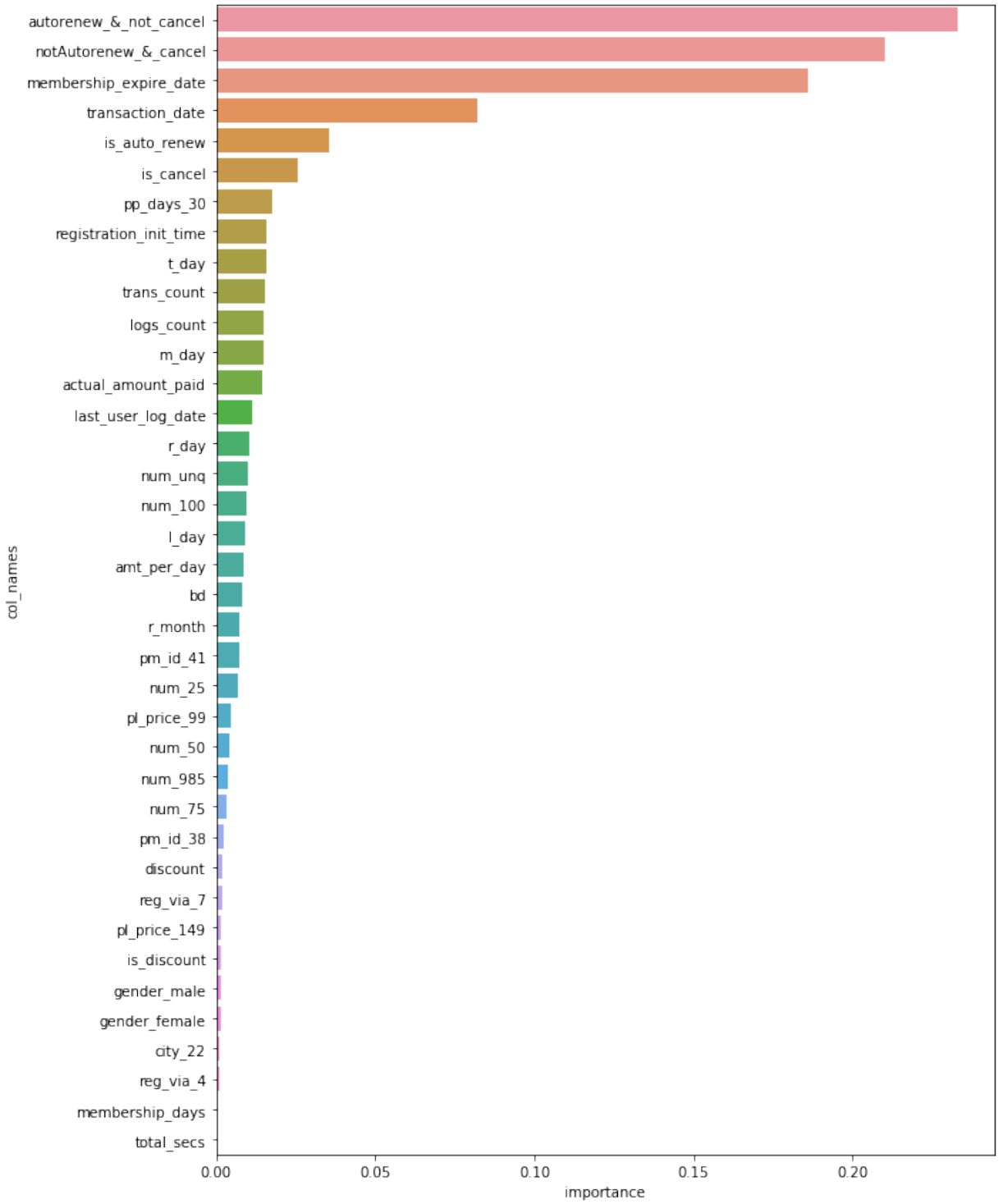
If we were allowed more time, we would spend more time on several things: first, we should perform more careful missing value imputations not only using the sample mean. More accurately re-creating the original values will likely enhance our model performance; second, in our analysis,

[10]Not too bad as a first Kaggle competition...?

we did not attempt to detect nor fix extreme values, which could potentially be helpful to model performance; third, we would aim to gain a more extensive understanding of the hyperparameters the mechanisms behind XGBoost so that we can better utilize this powerful tool.

7 Appendix

Variable importance plot for WRF:



References

Chen, C., Liaw, A., and Breiman, L. (2004). Using random forest to learn imbalanced data. *The Regents of the University of California*. Report ID: 666.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *ArXiv e-prints*.

Collier, A. B. (2015). Making sense of logarithmic loss. *exegetic.biz*.

Tan, C. C. (2009). *AUTOENCODER NEURAL NETWORKS: A Performance Study Based on Image Reconstruction, Recognition and Compression*. LAP Lambert Academic Publishing.