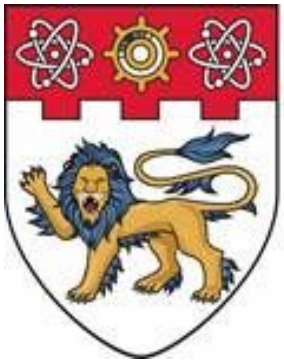


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CZ4041 - MACHINE LEARNING

PROJECT REPORT

Kaggle KKBox's Churn Prediction Challenge

STUDENT NAME	MATRICULATION NUMBER
DERRICK PEH JIA HAO	U1621219F
LIM DONG LI, TONY	U1620970K
LIM BOON LENG	U1621831E
ANG POH KEONG	U1621005E

Introduction

Kaggle KKBox's Churn Prediction Challenge

For a subscription business, accurately predicting churn is critical to long-term success. Even slight variations in churn can drastically affect profits.

KKBOX is Asia's leading music streaming service, holding the world's most comprehensive Asia-Pop music library with over 30 million tracks. They offer a generous, unlimited version of their service to millions of people, supported by advertising and paid subscriptions. This delicate model is dependent on accurately predicting churn of their paid users.

In this competition we were tasked to build an algorithm that predicted whether a user will churn after their subscription has expired. Currently, the company uses survival analysis techniques to determine the residual membership life time for each subscriber. By adopting different methods, KKBOX anticipates they will discover new insights to why users leave so they can be proactive in keeping users dancing.

The evaluation metric for this competition is Log Loss:
$$\textit{logloss} = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) + (1-y_i)\log(1-p_i))$$

where N is the number of observations, loglog is the natural logarithm, yiyi is the binary target, and pipi is the predicted probability that yiyi equals 1.

For each user id (msno) in the test set, we must predict the probability of churn (a number between 0 and 1).

Problem statement

KKBOX is a music streaming service that has many paid users who subscribe to them in order to utilize their unlimited services. A churn occurs when a user does not renew his/her subscription within 30 days after their subscription expires. The purpose of this project is to predict the churn rate of the user, given his digital footprints of his/her usage of the service.

Role of each member

All members were actively involved and participated in all meetings. The roles of each member were undefined as everyone contributed equally and ran our solutions on our own computers to save time by doing parallel processing of the data.

Presentation slides

https://prezi.com/view/wXUaXvmbFDuESN0EUab5/

Kaggle evaluation score and rank

We submitted multiple results and each submission got us a better score. We finally ended up with the following score and ranking after countless attempts of data processing and features extraction., as well as model parameters tuning.

Attempts	Results			Remarks
1	submission_results.csv 8 days ago by Boon Leng add submission details	0.61095	0.61626	1st attempt on local machine
2	Kernel25c3a88ddf (version 4/19) 2 days ago by Boon Leng From "Kernel25c3a88ddf" Script	0.15683	0.15740	1st attempt on Kaggle's Kernel using the average of all 4 boosting models
3	lgb_results_filtered.csv 20 hours ago by Boon Leng add submission details	0.13336	0.13451	Best attempt of LGB ran using tuned parameters, and filtered features
4	cat_results (fold = 3, sample = 10000 , iteration = 1000, lrate = 0.01, res... 11 days ago by StrawyTony add submission details	0.13398	0.13499	Best attempt of CAT ran using tuned parameters, and filtered features
5	xgb_sub_v5.csv 10 days ago by Jonathan add submission details	0.13386	0.13473	Best attempt of XGB ran using tuned parameters, and filtered features
6	merged.csv 2 minutes ago by Boon Leng xgb+lgb+cat	0.12996	0.13090	Best attempts of LGB + XGB + CAT, merged and averaged

Private Leaderboard: **Score: 0.12996** **Rank 110/575 (Top 19.2%)**

109	▼1	MOKA		0.12940	38
110	▲1	Preacher		0.12999	7

Public Leaderboard: **Score: 0.13090** **Rank 112/575 (Top 19.5%)**

111	▼46	Preacher		0.13056	7
112	▼42	kqhoffmann		0.13112	37

Methodology

Workflow

1. Preprocessing
    1.1. Datasets given by the organisers were analysed to see how they can be preprocessed and used accordingly.

2. Feature Extraction
    2.1. Features were selected and extracted out from various datasets and were combined with the training dataset to form the base training data.

3. Training
    3.1. The training dataset was split into 2 subset groups and used for cross validation. x contained the features while y contained the churn result of each user. (x1, y1) were used as the training dataset while (x2, y2) were used as the test dataset.
    3.2. These subset groups were used by a selected classifier model to train on.
        3.2.1. For cross validation, (x1, y1) were used as the training datasets for the classifier model selected.
        3.2.2. Parameters such number of iterations for the classifier (boosting algorithms) , learning rate and early stoppage was selected. This was to prevent overfitting of the training data.
        3.2.3. Prediction was done on (x2,y2) dataset and accuracy of the model was calculated using the the log loss function.
        3.2.4. This process was fine-tuned by repeating with other classifier models using different parameters for the model to observe any improvements.

4. Prediction
    4.1. The best model with the fine-tuned parameters was selected to predict the official test dataset given by the organisers.

Experiments were done on a suite of complex models and performance ceiling was established for each model.

1. Random Forest
    1.1. It is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve predictive accuracy and overfitting. For our model, we adapted the classifier from Scikit Learn.
    1.2. The sub-sample size of the random forest classifier is always same as input size. On the other hand, samples can be drawn with replacement using parameter bootstrap=True.
    1.3. There are 2 stages during the implementation of the classifier, namely random forest creation and prediction.
        1.3.1. Randomly select k features from total m features, k<m
        1.3.2. Amongst k features, calculate d using best split point
        1.3.3. Split into daughter nodes using best split point
        1.3.4. Repeat 1-3 until x number of nodes reached
        1.3.5. Build forest by repeating 1-4 until n times to create n number of trees
    1.4. It can be used for both classification and regression predictions.
    1.5. It contains a function to help identify important features from the training dataset.

2. Cat Boost
    2.1. It is an open-source algorithm from Yandex. It is able to work with diverse data types and provides best-in-class accuracy.
    2.2. A type of gradient boosting, but based around multiple categories of data.
    2.3. It works well with relatively less data unlike other algorithms.
    2.4. It can be used without any explicit pre-processing.
    2.5. It is robust, has low chances of overfitting and reduces the need for extensive hyper-parameter tuning although tuning is still available

3. XGBoost
    3.1. It is an extreme gradient boosting framework that implements gradient boosted decision trees, designed for speed and performance.
    3.2. There are 3 forms of gradient boosting
        3.2.1. Gradient boosting with learning rate
        3.2.2. Stochastic gradient boosting with sub-sampling at row, column and column per split levels
        3.2.3. Regularized gradient boosting with L1, L2 regularization
    3.3. Its algorithm features include
        3.3.1. Sparse Aware implementation - auto handling of missing data
        3.3.2. Block Structure - parallelization of tree construction
        3.3.3. Continued Training - further boost already fitted model on new data
    3.4. It is relatively fast when compared to other gradient boosting implementations and dominates structured or tabular datasets on classification and regression

predictive modeling problems

4. Light GBM
    4.1. It is a gradient boosting framework that uses tree based learning algorithm.
    4.2. It grows trees vertically (leaf-wise) unlike other algorithms that grows horizontally (level-wise)
    4.3. It is the preferred algorithm for big datasets as it can handle large datasets with low memory usage and focuses on the accuracy of results.
    4.4. It is not recommended to be used on small dataset as it is more sensitive to overfitting

Proposed solution

1. Dataset Preprocessing (Feature Engineering)

Transactions Dataset

Below are the default features given in “transactions.csv” and “transactions_v2.csv”.

Features	Description
msno	User ID
payment_method_id	Payment Method
payment_plan_days	Length of membership plan in days
plan_list_price	Listed plan price in New Taiwan Dollar (NTD)
actual_amount_paid	Actual amount paid for the membership in New Taiwan Dollar (NTD)
is_auto_renew	Auto renew of membership
is_cancel	Member cancellation transaction

Below are the features derived from the given features in “transactions.csv” and “transactions_v2.csv”.

Features	Derived Method
transaction_date	Converted to date time format
membership_expire_date	Converted to date time format
difference	Difference in days between the membership expiry date and last transaction date
discount	Amount of discount derived from the original plan list price and the actual amount paid
renewB4ExpirationDate	Renew before expiration date is true when difference is more then or equal 0 and the transaction is not a cancel transaction
renewAfterExpirationDate	Renew before expiration date is true when difference is less then 0 and the transaction is not a cancel transaction

trialUser	The user is a trial user when there is a transaction where the plan list price is 0 and the actual amount paid is 0
is_cancel_rate	The percentage of cancellation transactions over the total number of transactions performed
is_auto_renew_rate	The percentage of membership auto renew transactions over the total number of transactions performed
numOfTxn	Total number to transactions the user performed
numOfDiscountsInDecimal	The percentage of discounted transactions over the total number of transactions performed
churnOrNot1	Last transaction that is churn (1 or 0)
churnOrNot2	Second last transaction that is churn (1 or 0)
churnOrNot3	Third last transaction that is churn(1 or 0)
churnOrNot4	Fourth last transaction that is churn (1 or 0)
churnOrNot5	Fifth last transaction that is churn (1 or 0)

User Logs Dataset

Below are the default features given in “user_logs.csv” and “user_logs_v2.csv”.

Features	**Description**
msno	User ID
date	Date of the user log
num_25	Number of songs played less than 25% of the song length
num_50	Number of songs played between 25% to 50% of the song length
num_75	Number of songs played between 50% to 75% of the song length
num_985	Number of songs played between 75% to 98.5% of the song length
num_100	Number of songs played over 98.5% of the song length
num_unq	Number of unique songs played
total_secs	Total seconds of songs played

Below are the features derived from the given features in “user_logs.csv” and “user_logs_v2.csv”.

Features	Derived Method
usage_above_985	Average daily usage of songs played over 98.5% of the song length (in seconds)
total_days	Total number of days the user used the application
total_secs_mean	Mean number of the total number of seconds the user used the application (in seconds)
num_unq_mean	Mean number of the total number of unique songs played (in seconds)
total_secs_sum	Total number of seconds the user used the application (in seconds)

Members Dataset

Below are the default features given in “members_v3.csv”.

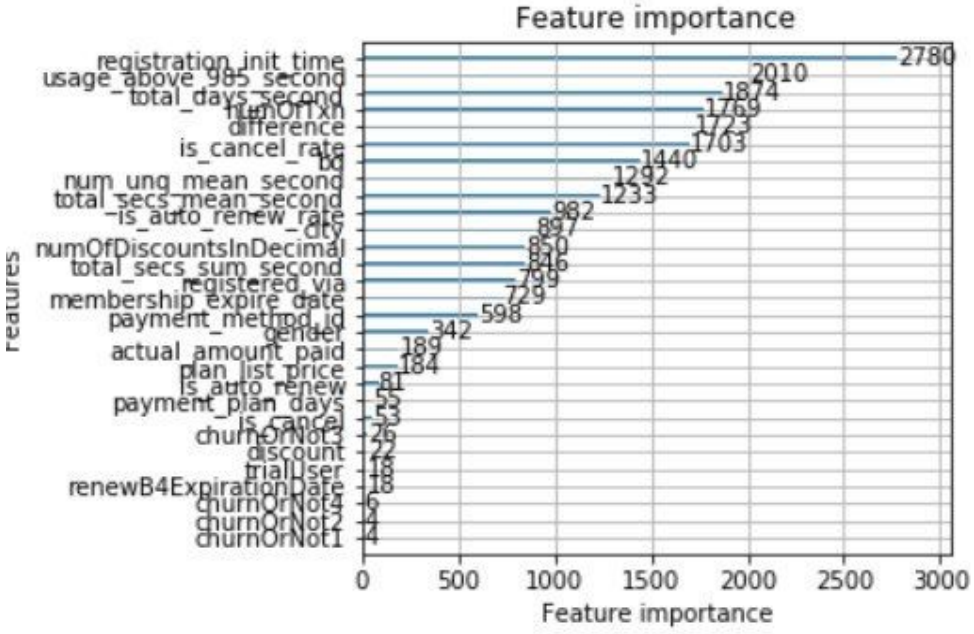
Features	Description
msno	User ID
city	City of the user
bd	Age of the user
gender	Gender of the user (male - 1, female - 2)
registered_via	Registration method
registration_init_time	Registration time

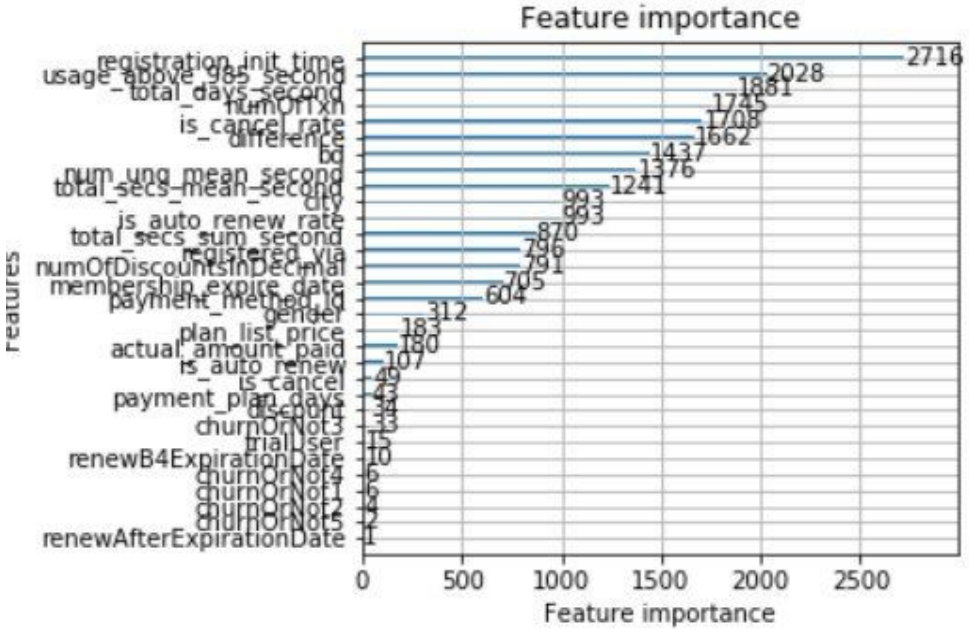
2. Features Importance

Features importance using LGB

Below is the graph of feature importances derived from the following code snippet:

```
model = lgb.train(lgb_params, train_set=d_train, num_
ax = lgb.plot_importance(model)
plt.tight_layout()
plt.savefig('lgb_feature_importance.png'.format(i))
```





Features importance using RandomForestRegressor

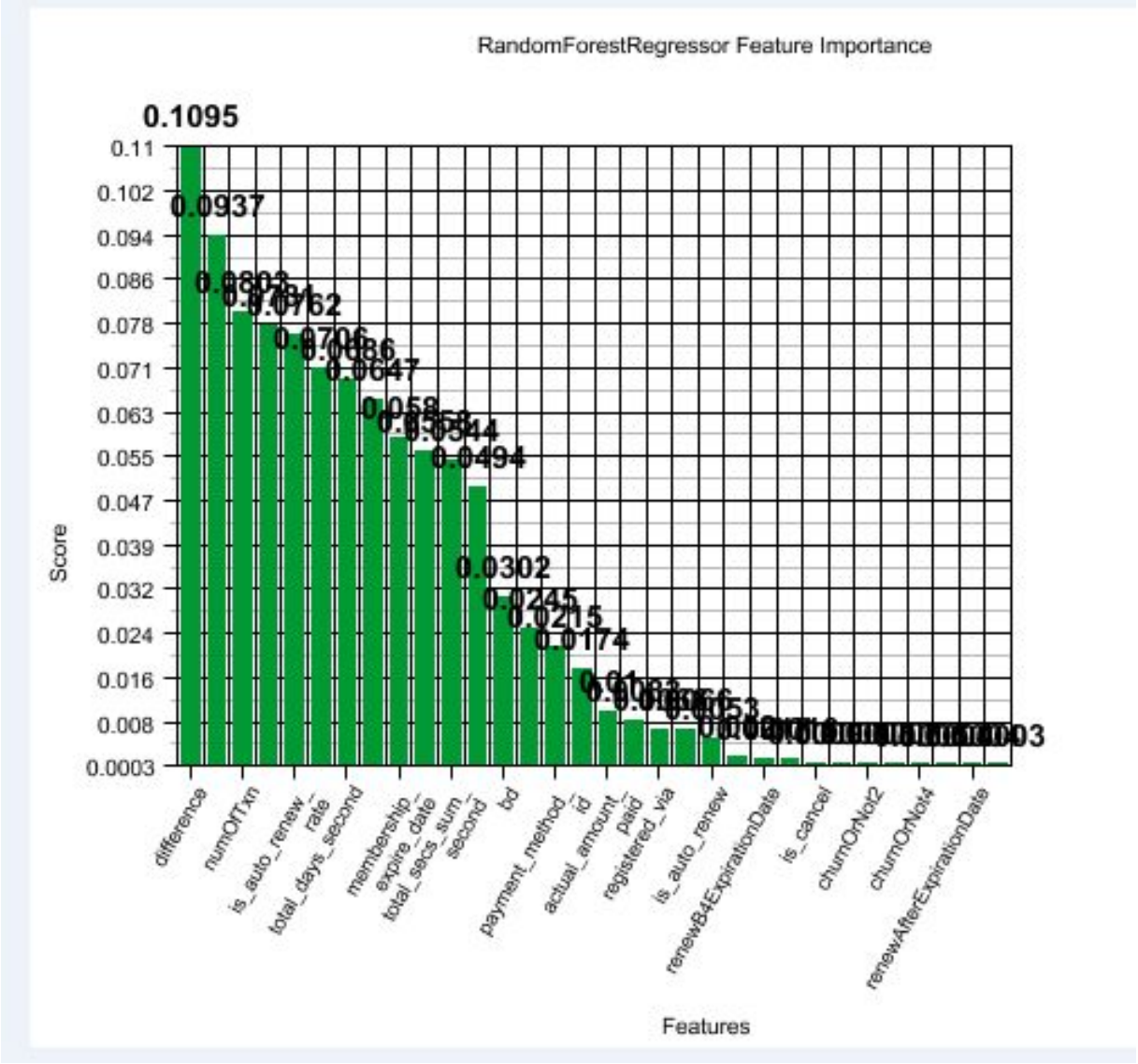
For classification, it is typically either *Gini impurity* or *information gain/entropy* and for regression trees it is *variance*. Thus when training a tree, it can be computed how much each feature decreases the weighted impurity in a tree. For a forest, the impurity decrease from each feature can be averaged and the features are ranked according to this measure.

Below is the graph of feature importances derived from the following code snippet:

```
from sklearn.cross_validation import ShuffleSplit
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from collections import defaultdict

import pandas as pd
import numpy as np

X = pd.read_csv('train_input.csv')
X = X.drop(['is_churn'], axis=1)
Y = X['is_churn']

rf = RandomForestRegressor()
scores = defaultdict(list)

#crossvalidate the scores on a number of different random splits of the data
for train_idx, test_idx in ShuffleSplit(len(X), 100, .3):
  X_train, X_test = X[train_idx], X[test_idx]
  Y_train, Y_test = Y[train_idx], Y[test_idx]
  r = rf.fit(X_train, Y_train)
  acc = r2_score(Y_test, rf.predict(X_test))
  for i in range(X.shape[1]):
    X_t = X_test.copy()
    np.random.shuffle(X_t[:, i])
    shuff_acc = r2_score(Y_test, rf.predict(X_t))
    scores[names[i]].append((acc - shuff_acc) / acc)

print("Features sorted by their score:")

print(
    sorted(
        [(round(np.mean(score), 4), feat) for feat, score in scores.items()],
        reverse=True))
```

Output:

[(0.1095, 'difference'), (0.0937, 'registration_init_time'), (0.0803, 'numOfTxn'), (0.0781, 'is_cancel_rate'), (0.0762, 'is_auto_renew_rate'), (0.0706, 'usage_above_985_second'), (0.0686, 'total_days_second'), (0.0647, 'num_unq_mean_second'), (0.058, 'membership_expire_date'), (0.0558, 'total_secs_mean_second'), (0.0544, 'total_secs_sum_second'), (0.0494, 'transaction_date'), (0.0302, 'bd'), (0.0245, 'city'), (0.0215, 'payment_method_id'), (0.0174, 'numOfDiscountsInDecimal'), (0.01, 'actual_amount_paid'), (0.0083, 'gender'), (0.0066, 'registered_via'), (0.0066, 'plan_list_price'), (0.0053, 'is_auto_renew'), (0.002, 'payment_plan_days'), (0.0017, 'renewB4ExpirationDate'), (0.0016, 'churnOrNot3'), (0.0009, 'is_cancel'), (0.0008, 'discount'), (0.0008, 'churnOrNot2'), (0.0007, 'churnOrNot5'), (0.0005, 'churnOrNot4'), (0.0004, 'trialUser'), (0.0004, 'renewAfterExpirationDate'), (0.0003, 'churnOrNot1')]



As visible from the graph results, both LGB and RFR have ranked the importance of the features quite similarly. Using both results, we *dropped the features* deemed to be less important to purify the input dataset in an attempt to achieve a better logloss result (which actually did). The following pages will show that the drop in unimportant features improved our logloss scores.

3. Model Experimentations

Random Forest Classifier

```
# rfc
print('rfc training')

model = RandomForestClassifier(n_estimators=200,n_jobs=-1,random_state=3225+i)
model = model.fit(x1, y1)
rfc_valid = model.predict_proba(x2)[:,1]
print('rfc log loss = {}'.format(log_loss(y2,rfc_valid)))
rfc_pred = model.predict_proba(test[cols])[:,1]
```

- We chose this model during our first few runs of predicting the dataset and results were not ideal, as the logloss scores were very high, at an average of 0.6
- We tried tuning the parameters as well as ran it on the filtered dataset but the results were around the same
- We gave up on Random Forest Classifier and directed our attention to the other 3 models in attempts to improve the logloss score

submission_results.csv 8 days ago by Boon Leng add submission details	0.61095	0.61626

Cat Boost Classifier

```
model = CatBoostClassifier(cat_params)
# training dataset
model = model.fit(x1, y1,eval_set=(x2,y2),logging_level='Silent')

# actual test dataset
cat_pred = model.predict_proba(test[cols])[:,1]

# training-test dataset
cat_valid = model.predict_proba(x2)[:,1]
print('cat valid log loss = {}'.format(log_loss(y2,cat_valid)))

cat_preds += cat_pred
```

Attempt:

```
cat training
cat valid log loss = 0.2699528030238153
```

Parameters: 200 iterations, learning rate 0.15

XGBoost

```
watchlist = [(xgb.DMatrix(x1, y1), 'train'), (xgb.DMatrix(x2, y2), 'valid')]

# training dataset
model = xgb.train(xgb_params, xgb.DMatrix(x1, y1), 200,  watchlist, feval=xgb_score,
maximize=False, verbose_eval=100, early_stopping_rounds=10)

# actual test dataset
xgb_pred = model.predict(xgb.DMatrix(test[cols]), ntree_limit=model.best_ntree_limit)

# training-test dataset
xgb_valid = model.predict(xgb.DMatrix(x2))
print('xgb valid log loss = {}'.format(log_loss(y2,xgb_valid)))

xgb_preds += xgb_pred
```

Attempt 1:

```
[0]     train-logloss:0.661834  valid-logloss:0.662606  train-log_loss:0.661834 valid-log_loss:0.662606
Multiple eval metrics have been passed: 'valid-log_loss' will be used for early stopping.

Will train until valid-log_loss hasn't improved in 50 rounds.
[100]   train-logloss:0.133196  valid-logloss:0.158255  train-log_loss:0.133196 valid-log_loss:0.158255
[199]   train-logloss:0.110382  valid-logloss:0.151108  train-log_loss:0.110382 valid-log_loss:0.151108
xgb valid log loss = 0.15110775040084262
(12996118,)
[Finished in 425.951s]
```

eta: 0.04, max_depth: 5, early_stopping_rounds: 50, min_child_weight:7

Attempt 2:

```
[0]     train-logloss:0.661061  valid-logloss:0.661587  train-log_loss:0.661061 valid-log_loss:0.661587
Multiple eval metrics have been passed: 'valid-log_loss' will be used for early stopping.

Will train until valid-log_loss hasn't improved in 50 rounds.
[100]   train-logloss:0.126357  valid-logloss:0.152452  train-log_loss:0.126357 valid-log_loss:0.152452
[199]   train-logloss:0.105329  valid-logloss:0.148144  train-log_loss:0.105329 valid-log_loss:0.148144
xgb valid log loss = 0.14814416665985483
(12996118,)
[Finished in 332.927s]
```

eta: 0.04, max_depth: 5, early_stopping_rounds: 50, min_child_weight:9

Attempt 3:

```
[0]     train-logloss:0.661516  valid-logloss:0.661525  train-log_loss:0.661516 valid-log_loss:0.661525
Multiple eval metrics have been passed: 'valid-log_loss' will be used for early stopping.

Will train until valid-log_loss hasn't improved in 50 rounds.
[100]   train-logloss:0.13305   valid-logloss:0.141489  train-log_loss:0.13305  valid-log_loss:0.141489
[199]   train-logloss:0.111287  valid-logloss:0.131708  train-log_loss:0.111287 valid-log_loss:0.131708
xgb valid log loss = 0.1317075153623591
(12996118,)
[Finished in 344.444s]
```

eta: 0.04, max_depth: 5, early_stopping_rounds: 50, min_child_weight:8

LightGBM

```
d_train = lgb.Dataset(x1, label=y1)
d_valid = lgb.Dataset(x2, label=y2)
watchlist = [d_train, d_valid]

# training dataset
model = lgb.train(lgb_params, train_set=d_train, valid_sets=watchlist, verbose_eval=100)

# feature importance of lgb
ax = lgb.plot_importance(model)
plt.tight_layout()
plt.savefig('feature_importance_{}.png'.format(i))

# actual test dataset
lgb_pred = model.predict(test[cols])

# training-test dataset
lgb_valid = model.predict(x2)
print('lgb valid log loss = {}'.format(log_loss(y2,lgb_valid)))

lgb_preds += lgb_pred
```

Attempt 1:

```
start fold 1
lgb training
Training until validation scores don't improve for 10 rounds.
[100]   training's binary_logloss: 0.111667     valid_1's binary_logloss: 0.112391
[200]   training's binary_logloss: 0.104218     valid_1's binary_logloss: 0.105334
[300]   training's binary_logloss: 0.0991529    valid_1's binary_logloss: 0.100642
[400]   training's binary_logloss: 0.0942924    valid_1's binary_logloss: 0.0960306
[500]   training's binary_logloss: 0.0909984    valid_1's binary_logloss: 0.0931337
[600]   training's binary_logloss: 0.0877892    valid_1's binary_logloss: 0.0902544
Did not meet early stopping. Best iteration is:
[600]   training's binary_logloss: 0.0877892    valid_1's binary_logloss: 0.0902544
lgb log loss = 0.09025445991461725
(12996118,)
```

Parameters: 600 iterations, learning rate 0.1

Attempt 2:

```
start fold 1
lgb training
Training until validation scores don't improve for 10 rounds.
[100]   training's binary_logloss: 0.120261     valid_1's binary_logloss: 0.120692
[200]   training's binary_logloss: 0.111294     valid_1's binary_logloss: 0.111916
[300]   training's binary_logloss: 0.107536     valid_1's binary_logloss: 0.10837
[400]   training's binary_logloss: 0.10413      valid_1's binary_logloss: 0.105123
[500]   training's binary_logloss: 0.101293     valid_1's binary_logloss: 0.102473
[600]   training's binary_logloss: 0.0985576    valid_1's binary_logloss: 0.0998928
Did not meet early stopping. Best iteration is:
[600]   training's binary_logloss: 0.0985576    valid_1's binary_logloss: 0.0998928
lgb log loss = 0.09989279026090211
(12996118,)
```

Parameters: 600 iterations, learning rate 0.05

Noted that when learning rate is decreased to 0.05, the valid log loss value increased *HOWEVER, when submitted to Kaggle, the log loss decreased from ~0.17 to ~0.15!

5. Thought Process

Why these features

- We first derived more features by manipulating the original features and made attempts to reduce the huge user logs dataset (29GB)
- The user logs dataset was reduced to ~500MB doing averaging and summing on the existing data, as well as reducing the 6 columns of num_xx into a single feature called 'usage_above_985' which is actually the ratio of the number of songs played above 98.5% of the song duration to the total number of songs he played
- The transactions dataset was the more important dataset as visible from the feature importance graphs
- We derived several more features that we thought would be useful as it demonstrates user transaction activities.
- An example is deriving the number of transactions; the thought is that the more the transactions a user has, the more likely that the user will not churn and continue his subscription renewal.
- Another example is the discount and number of discounts; the thought is that the higher the discount is for a plan, and the number of times the transactions had a discount, the more likely that the user is a discount seeker and will (not) churn whenever there are discounts
- All features that were derived were thought to depict user activity/behaviour of KKBox, and might improve training results due to variations in human behaviour. The idea of deriving such new features were so that the training models would be able to identify a specific group of behaviours that might imply that the user is likely to (not) churn

Why these models

- We first referred to Kaggle's wiki site (https://www.kaggle.com/wiki/Algorithms) and noticed a few algorithms that we are familiar with (taught in Lectures), and many new and interesting algorithms
- After that we looked through the discussion board of the challenge and chose a few algorithms that gave good results based on others' leaderboard rankings and came up with the final few as listed above
- We researched each algorithm and noted down their pros and cons before implementing them in Python language

Why Python

- Python has many extensively machine learning library (scikit-learn, etc.) that contains many machine learning algorithms for various problems
- Python has an advanced module extension (plotly) to do data plotting
- There are lots of extensive learning materials that guided us throughout our machine learning journey with Python

Challenges faced

We encountered many challenges throughout the project due to unfamiliarity with Python and Machine Learning. Below is the list of problems we faced multiple times and some were unable to be overcome due to certain limitations.

1. Memory errors - Limitations of computer, dataset too big
    a. All of our computers only have 8GB RAM. The biggest dataset of this competition is 29GB. Due to the limitation of our computer, many hours were wasted as our program would result in a 'MemoryError' during the processing of the data features. This scenario happened multiple times throughout the processing, and training of the data and was a major bottleneck to our progress.
    b. This problem was not solved certainly as some of the machine learning models do not support incremental learning, and thus the training data had to be processed as a whole. This was a major issue as our final training data was 7.9GB, and not enough RAM was available to do the training. Ultimately, after filtering features using features importance, the data was reduced to 5GB and running the program became successful.
    c. This problem was solved in some scenarios whereby split processing can be done. An example would be the 29GB of user_logs, where we were able to process the data in chunks, so that the RAM would not be used up.
    d. We tried to run the program on Kaggle's Kernel and it still failed us occasionally, with the kernel returning us an error message of 'Exceeded memory limit of xxx'.
2. Time and resource consumption
    a. Due to the huge dataset, time taken to run the processing and training is extremely long and as each of the team only has one computer, running the program(s) render our computer useless as we cannot perform other tasks as well as risk the program catching memory errors, then wasting our time and efforts.
    b. As we are unable to run the script to fold more than 1 time, we have to run the script multiple times to simulate multiple foldings in order to get the final results. This meant that while the script is running, we were unable to perform other tasks on our computers.
3. Fine-tuning of training model
    a. In order to further improve the score of the training model, fine-tuning of the training model is required. This includes including certain parameters that are available in the model. Using an early stoppage parameter for example minimizes the chance of over fitting.
    b. Tuning of the training model requires lots of trial and error as some parameters may prove not to be useful in improving the overall score. Furthermore, though tuning the parameters may result in a better score, over-tuning the parameters may be detrimental to the training model. Having too much iterations for example may result in overfitting of the model.

Conclusion

Throughout the project, we have learnt that features engineering is very important in machine learning. Having more features does not mean that the prediction will turn out to be more accurate, same for vice versa. It is essential to have more quality data, than quantity.
Feature importance is very helpful in helping to filter out less useful data, so that the input features will be more diverse and less-skewed for the predictive models.

We have also learnt that running machine learning algorithms on huge datasets essentially require a very huge amount of memory space, and takes up alot of resources and time on our computers.

Fine-tuning of the parameters is also another factor in improving predictions, and can actually help prevent overfitting or underfitting, in contrast to the features input. Factors include early stoppage, ensembling and more.

Overall, this project has largely improved our knowledge on machine learning and has been very enriching and interesting on how so many factors can affect the final predictive results.