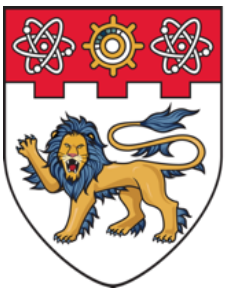


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

KKBOX CHURN PREDICTION CHALLENGE

CE4041 – MACHINE LEARNING

Sem. 2 - Academic Year 2017/18

Kaggle score: 0.11859
Ranked position: 48 of 575 (TOP 8.3%)

GROUP 6
Pablo Javier de Paz Carbajo
Rick Schneider
Maxime Kayser
Ragnhild Skirdal Froehaug
Martha Honganvik Andersen

ABSTRACT

Accurate prediction for the churn-rate of users plays a crucial role in the development of business strategies for digital companies that aim to create a long-lasting relationship with their customers.

To this end, as part of the Kaggle competition "KKBOX Churn prediction challenge", we predict the churn rate of the Asian music streaming-service's customers. This report presents our approach to solve this predictive modelling task.

To make an accurate prediction, the data of three data-sets was explored. These files contain information about the user activity, registration and subscription history, as well as the users themselves. In addition, the report introduces our approach to feature engineering as well as feature selection.

Our final prediction model using extreme gradient boosting (XGBoost) yields a final score of 0.11859, ranking us among the top 8.3% of the competitors.

ACKNOWLEDGEMENTS

First, we would like to thank Nanyang Technological University and especially, the School of Computer Science and Engineering for the opportunity to take part in the course CE4041.

We want to thank Prof. Sinno Jialin Pan for conducting interesting and educational lectures and tutorials, which helped us to grow a strong foundation in Machine Learning concepts.

The datasets used in this project are provided by KKBOX and the competition platform is hosted by Kaggle.

TABLE OF CONTENTS

1 INTRODUCTION

KKBOX is Asia's leading music streaming service, holding more than 40 million tracks as in April 2018. It was established in Taiwan in 2004 and has more than 10 million users. They offer 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.

Kaggle is a platform for predictive modelling and analytics competitions in which statisticians and data miners compete to produce the best models for predicting and describing the datasets uploaded by companies and users. There are countless strategies that can be applied to any predictive modelling task and it is impossible to know beforehand which technique will be most effective.

In this competition, we are asked to build an algorithm that **predicts** whether a **user will churn or not** after his/her subscription period expires. To be more specific, we want to forecast if a user makes a new service subscription transaction within 30 days after the current membership expiration date.

Note that this competition is finished by the time we started working on it (February 2018).

To develop our prediction model, Kaggle and KKBOX provided the following files:

- **Train**. Train set (user ID + whether they churn or not).
- **Sample submission**. Test set (user ID + prediction to be done on churn).
- **Transactions**. Data about user transactions (payment method, plan days, price, transaction date…).
- **User logs**. Daily user logs describing listening behaviors of a user (date, total seconds played, number of unique songs played…).
- **Members**. User information (city, birthday, gender…).

The report proceeds as follows. First the team and the roles of each member are briefly introduced. Next, the data-sets are explored in detail. Every feature and its relationship to the response variable *is_churn* (provides information on whether the user has churned or not) is analyzed. Section 4 introduces the features that have been engineered for this competition. In section 5, four different Machine Learning algorithms (i.e. Logistic Regression, XGBoost, AdaBoost and Random Forrest) are examined. In section 6, the methodology used to implement our prediction model for the Kaggle competition is explained. Finally, Section 7 & 8 present the results of our project, a conclusion and suggestions for further improvements of our model.

> Churn: "no new valid service subscription within 30 days after the current membership expires".

2 TEAM

Our team consists of five members, and the roles and responsibilities are stated in table 1 and table 2.

Table 1: Team Roles

Team member	Role
Pablo Javier de Paz Carbajo	Data analyst, data scientist, report writer
Rick Schneider	Data analyst, data scientist, report writer
Maxime Kayser	Data analyst, data scientist, report writer
Ragnhild Skirdal Froehaug	Data scientist, presenter, report writer
Martha Honganvik Andersen	Presenter, business analyst, report writer

Table 2: Roles and responsibility

Role	Responsibility
Data analyst	Interpret the comprehensive datasets, and analyze the results using statistics. Responsible for data pre-processing and transformation.
Data scientist	Responsible for model building and evaluation.
Presenter	Responsible for the project presentation.
Report writer	Responsible for writing the report and explain the procedures carried out.
Business analyst	Gather technical and business requirements and, documenting technical and business requirements.

3 DATA ANALYSIS

KKBOX provides three datasets with information about the transactions (*transactions*), the users listening behavior (*user_logs*) and the users themselves (*members*). Before developing a prediction model, we have to develop a clear understanding of the different data-sets and their features.

For each of the datasets above, we have two different versions: *version_1*, which contains data until February 28th, 2017 (used later for training part) and *version_2*, which contains data for the month of March 2017 (used for train and testing part).

This section proceeds as follows. First, the different features of each data-set are briefly explained and visualized. Next, we elaborate on the relationship between the features and the response variable *is_churn*. This step is particularly important for the feature engineering (see section 4), as it is crucial to understand in what way the features relate to the response variable. Finally, we compare the features to each other to get an overview of the correlation that may exist between them. This could eventually help us to reduce the number of features by dropping those that are highly correlated.

3.1 TRANSACTIONS

The transactions dataset contains 9 columns (each one representing an independent feature). In *version_1*, there are 21,547,746 rows whereas *version_2* contains only 1,431,009 rows.

Table 3 shows the different features and their meaning.

Table 3: Features

Features	Type	Description
msno	object	The user ID linking each session to a user.
payment_method_id	int64	Each integer represents a payment method.
payment_plan_days	int64	Describes length of membership subscription in days.
plan_list_price	int64	Price of a subscription in New Taiwan Dollar(NTD).
actual_amount_paid	int64	Actual amount paid for a subscription in NTD.
is_auto_renew	int64	Shows if a subscription was automatically renewed.
transaction_date	int64	Date when transaction took place.
membership_expire_date	int64	Date when subscription expire.
is_cancel	int64	Shows if a user actively cancelled a subscription.

3.1.1 UNIVARIATE DESCRIPTION

In order to get an idea of how the transactions dataset could be used to create a prediction, we looked into the different features and their distribution.

Payment_method_id

The different payment methods in KKBOX are represented by an integer from 1 to 41, where the most popular payment method is 41. Looking at the distribution of the data in a count plot, it can be observed that methods above 30 are used by a noticeable number of users. In other words, methods below 30 can be neglected for our model as they only constitute 5,3% of the total users and can be considered as noise for our future prediction model. Figure 1 shows *payment_method_id* values above 30 (as already mentioned, payment methods with a label below 30 only represent a small part of the users).

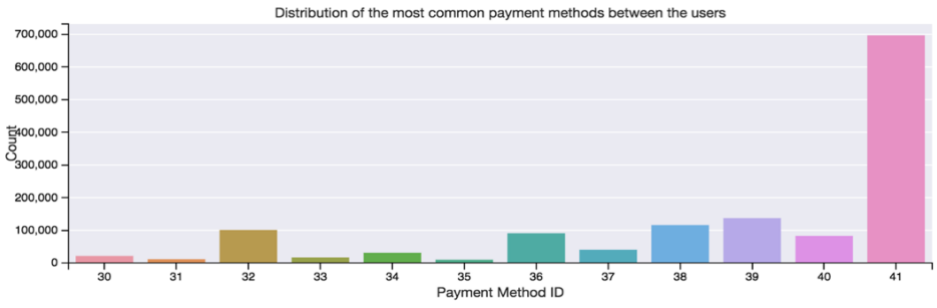


Figure 1: Payment Method ID

Payment_plan_days

This feature represents the length of the membership subscription for each user (in days). The majority of the users choose 30 days as their payment plan, as shown in figure 2. There are many outliers in both values below 30, as well as values above it. After the 30-days payment plan, the most chosen payment plans are 410, 195, 180, 90, 7 days (in descending order of importance).

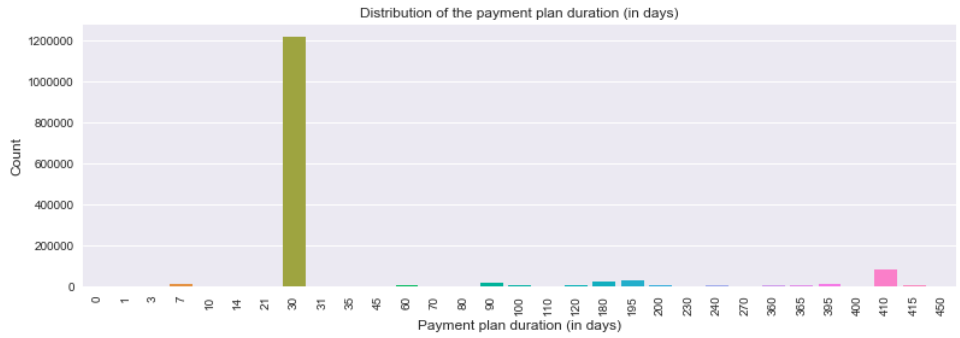


Figure 2: Payment plan duration

Plan_list_price and actual_amount_ paid

This feature describes the price that should be paid for and the price that is actually paid for a subscription. The mean is a bit higher in *plan_list_price* than in *actual_amount_paid*, which shows that some users get discounted subscriptions. Only values below 210 New Taiwan Dollar(NTD) are significant for the distribution of these two features. Values 99 and 149 are the most common ones, regrouping 70% of the total data.

In addition to these two features, we created a new one as the subtraction of them (refer to section 4.1). It can be observed that most of the people paid what they owe but there are some mismatches for individual users. Interestingly, in some cases, users end up paying more than they actually should.

Auto_renew

The feature *auto_renew* shows if a transaction was automatically renewed (value 1) or not (value 0). The distribution of the values of this feature is shown in the bar-plot in figure 3.

Most of the users, around 80%, choose to automatically renew their subscription whereas only 20% want to manually renew their subscription.

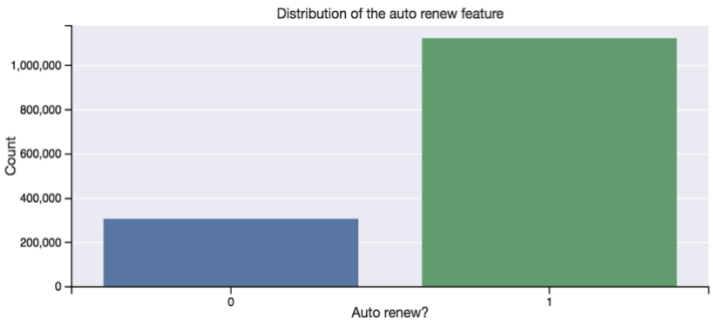


Figure 3: Auto Renew

Transaction_date and membership_expire_date

These two features show the date of the transaction and the membership expiration date respectively. The number of transactions is increasing every year. Thus, there are more transactions done now than a few years ago. This might be caused by an increased number of subscribers (even though the number of new subscriptions is decreasing in 2017, see section 3.3.1).

Transaction on a monthly basis tend to be high in February and March (a huge increase) followed by January and December. We need to take into account that the given data is mainly of the previous months to April 2017. Transaction date day-wise trends are higher in Friday. These trends can be observed below in figure 4.

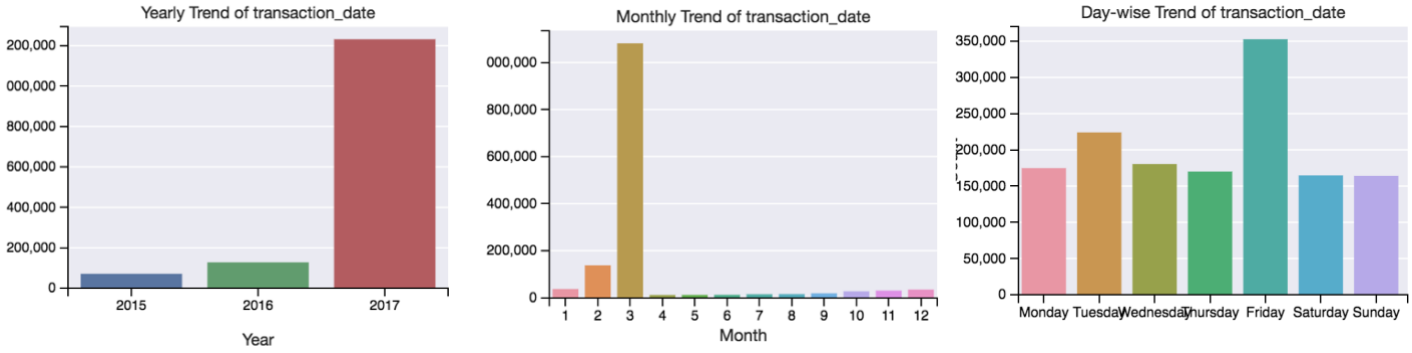


Figure 4: Transaction date

Most memberships expire in 2017, followed by 2018. However, some subscriptions expire after several years, and as such, they can be considered as outliers.

The membership expire date monthly trends are high in March, April and May. Indeed, the transaction date was higher in February and March, and as a result, the membership will expire in a short period of time (unless it is renewed).

These trends can be seen below is shown in figure 5 and 6.

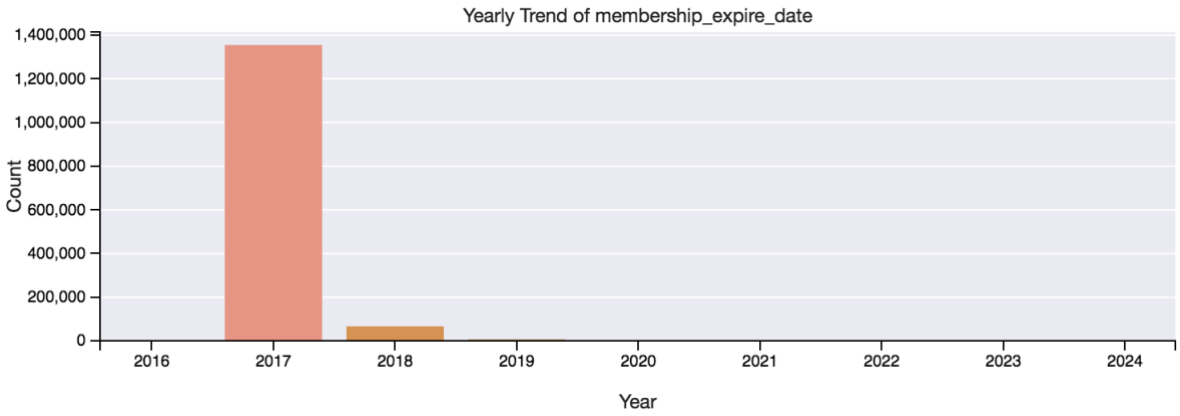


Figure 5: Membership expiry year

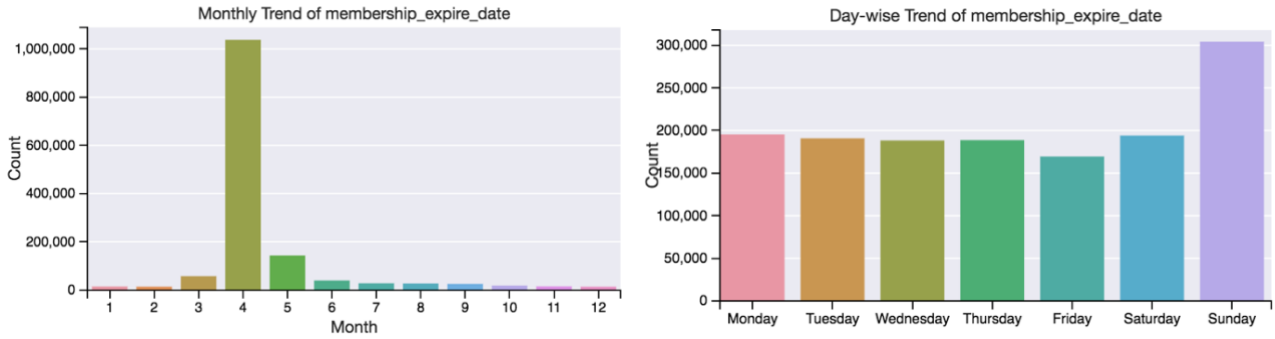


Figure 6: Membership expiry date

Is_cancel

The feature *is_cancel* shows whether a user manually cancelled a transaction or not. If so, he/she is more likely to churn. 96% of the users do not cancel their subscription.

3.1.2 CHURN RATE AND FEATURE RELATION

Is_churn and Payment_method_id

Payment_method_id compared to the response variable is_churn, shows that 96% of people with payment method 32, 86% of method 35 and 30% of method 38 churn. These payment methods are chosen by a smaller number of users and then, it is easier that most of them (or a large amount of them) churn because they are a small group compared to others such as payment method 41.

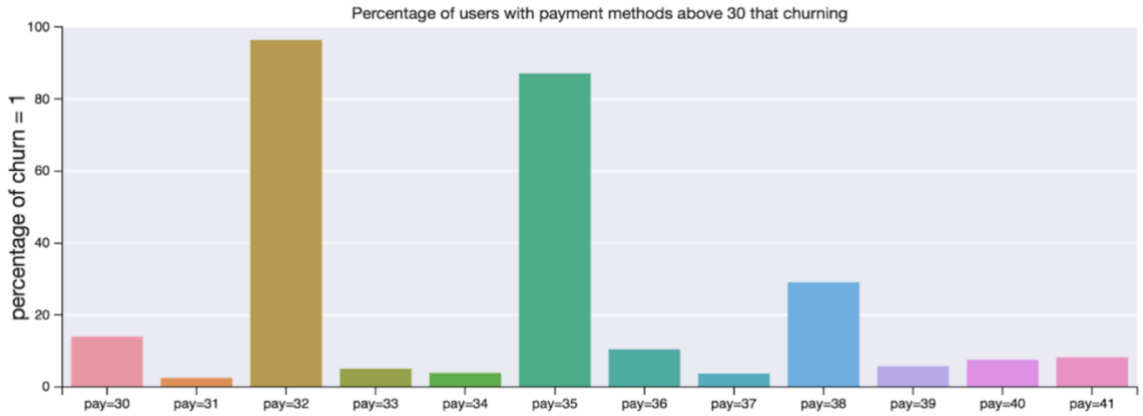


Figure 7: Is_churn and Payment_method_id

Is_churn and is_auto_renew

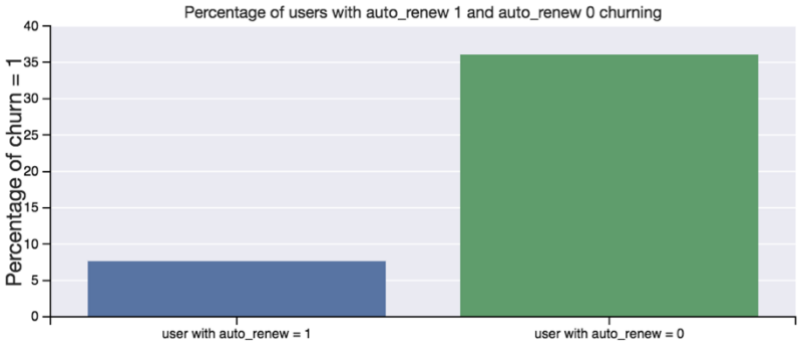
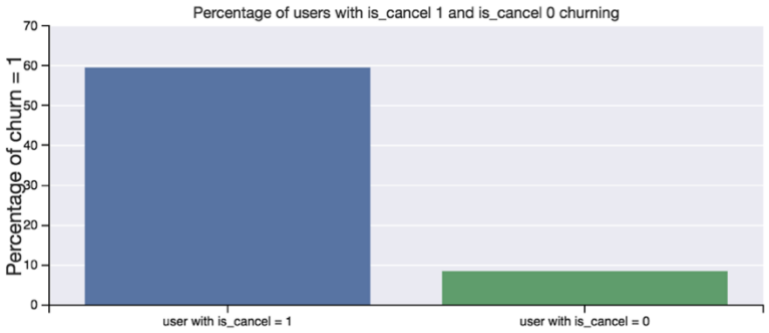


Figure 8 shows that 7,7% of the people with auto renew churn, while 36,1% of the people without auto renew churn. This means that people that choose auto renew maybe do not care much about the subscription, so they will not churn.

Figure 8: Is_churn and is_auto_renew

Is_churn and is_cancel



Subscribers who have canceled their membership are more than six times as likely to churn as the ones who have not canceled their membership. This is shown in figure 9. Users that do not cancel will probably continue and not churn.

Figure 9: Is_churn and is_cancel

3.1.3 CORRELATION ANALYSIS

Table 4 shows the correlation between the two features that contain price information. They are almost perfectly positively correlated (correlation = 0.99). As such, both features are strongly related to each other, and the prediction model might not need both features as an input. In fact, the best way would probably be to represent them both in one feature by calculation their difference.

Table 4: Correlation of features

Feature 1	Feature 2	Correlation
plan_list_price	*actual_amount_paid*	0.99

3.2 USER LOGS

The initial dataset has 392 million rows (excluding user_logs_v2.csv) and 9 columns. It contains information about the listening behavior of the users from January 2015 to April 2017. Table 5 describes the type and meaning of each initial feature of the dataset. Due to the large amount of data, we first explored a sample of 5 million rows to get an idea of trends in the dataset.

Table 5: Features

Features	Type	Description
msno	object	The user ID linking each session to a user.
date	int64	The day where the listening session took place.
num_25	int64	The amount of songs that were skipped before 25% of the song was played.
num_50	int64	The amount of songs that were skipped when 25-50% of the song was played.
num_75	int64	The amount of songs that were skipped when 50-75% of the song was played.
num_985	int64	The amount of songs that were skipped when 75-985% of the song was played.
num_100	int64	The amount of songs that were played entirely.
num_unq	int64	The amount of individual songs that were played.
total_secs	float64	The total duration of one listening session in seconds.

3.2.1 UNIVARIATE DESCRIPTIONS

In data exploration, we plot, amongst others, the distribution of the features *entries per user* and *total_secs* as well as the *number of songs played in each skipping category*. Our Jupyter Notebook on GitHub contain the remaining plots.

Total_secs

This feature contains information about the total duration of a listening session. It contains values in the range [-9e+15,9e+15], indicating that outliers had to be removed. Hence, we removed all rows containing negative seconds. In total we removed 820 rows, which is 0.02% of the sample. From figure 10 we see that most users listen between 0 and 10 000 seconds per session.

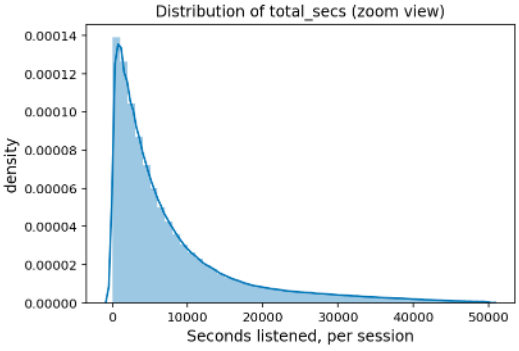


Figure 10: Distribution of total seconds

Entries per user and skipping behavior

Figure 11 shows that in the extracted user log sample, each user has between 1 and 16 entries. For the complete dataset, we show in part 4.3 (Feature Engineering) that the number of entries per user is actually much larger.

Figure 12 shows the general trend in listening behavior. It shows that most songs are played entirely. For the different skipping patterns, playing only up to 25 % of the song is the most common one.

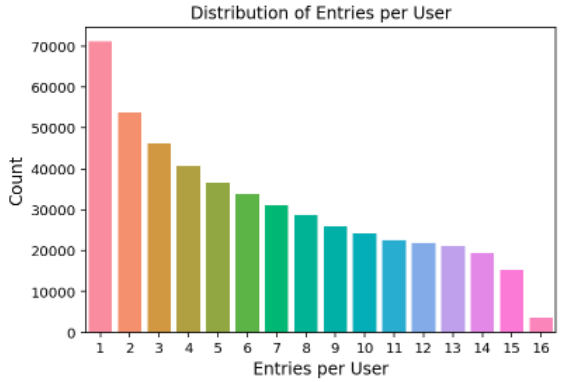


Figure 11: Entries per user

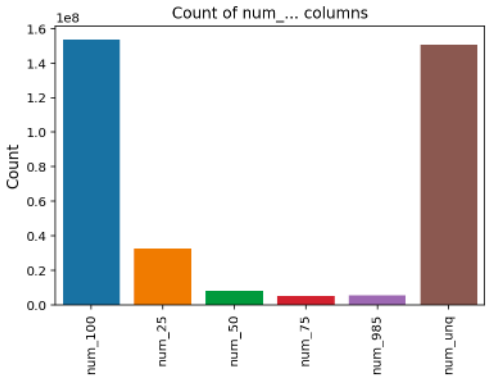


Figure 12: Number of songs played in each skipping category

3.2.2 CHURN RATE AND FEATURE RELATION

Is_churn and entries per user

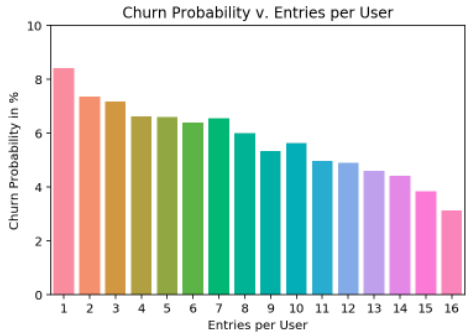


Figure 13: Entries per user compared to churn

We have also investigated how each feature affects the probability of churning. Figure 13 shows that the probability of churning increases as the number of entries per user decreases. On average, churning users have 13.9% fewer entries.

Is_churn and skipping categories

Table 6: Listening behavior

Num_xx	Mean churn	Mean non-churn	Mean difference
num_25	37.55	41.27	-9.0%
num_50	9.52	10.27	-7.3%
num_75	5.84	6.42	-9.0 %
num_985	6.55	7.14	-8.3 %
num_100	169.32	194.75	-13.1 %
num_unq	169.09	191.39	-11.7 %

Regarding number of songs played in each skipping category, table 6 shows that, in general, churning customers listen to fewer songs than non-churning customers. Differences are larger for songs played

entirely than songs skipped prematurely. This indicates that churning users have a slightly different listening behavior, which validates the use of these columns in our models.

3.2.3 CORRELATION ANALYSIS

The heat map in figure 14 shows how the initial features are correlated. The strongest correlation is between num_100 and total_secs, hence we can consider leaving total_secs out in the input data of our prediction model. Other strongly correlated features are num_unq and total_secs and num_unq and num_100.

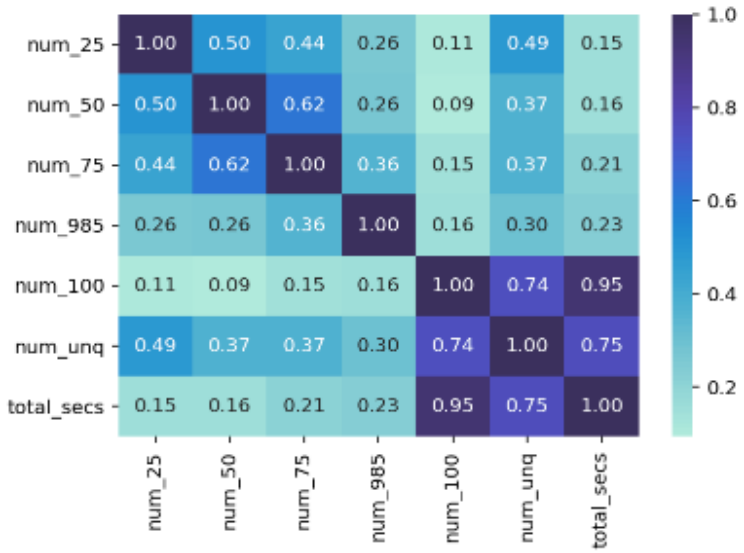


Figure 14: Heat map showing correlation between initial features

3.3 MEMBERS

The members dataset contains 6 different features (see table 7).

Table 7: Features

Features	Type	Description
msno	object	The user ID linked to a member.
city	int64	The city a user lives in.
bd	int64	The age of the user
gender	int64	The users gender
registered_via	int64	A number indicating how the user registered.
registration_init_time	int64	The date the user performed the registration on KKBOX.

3.3.1 UNIVARIATE DESCRIPTION

In this section, the different features for the members data-set are introduced and visualized.

City

This feature provides information about the cities the users live in. The anonymized city variable can take values from 1 to 22, indicating that the users come from 22 different cities. As we can see from figure 15, ⅔ of the entries are from city 1. This is followed by city 4, 5, 6, 3, 15 and 22. It should be noted that the dominance of city 1 can be explained in two ways. First, city 1 could simply represent a very large and popular city, which would explain why there are so many entries for this value. Second, city 1 could be the go-to value if there is actually no information on the city available. This would mean that many users did not provide any information on the city they live in, and that in this data set, instead of having a NaN entry, they are labeled as city 1.

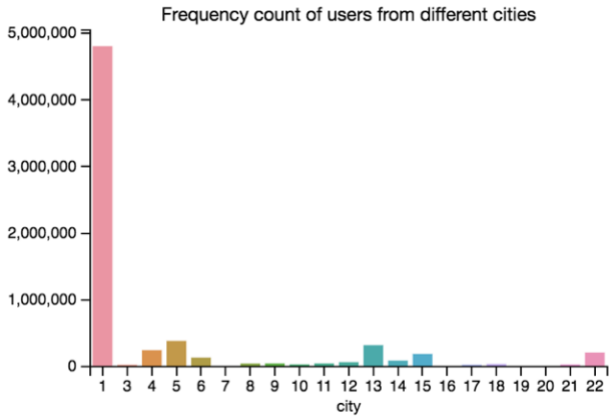


Figure 15: Histogram of users from different cities

Birthday

The birthday feature shows the age of the different users. The values range from -700 to +1000, which clearly indicates that there are many outliers. Figure 16 shows the distribution of the birthdays. It can be observed that ⅔ of the entries are equal to 0. This is probably due to the fact that KKBOX does not have data on the birthday for every single user, and that the go-to value for missing birthdays is 0. To get rid of outliers we plot ages larger than 0 and less than 110. Only 33% of the birthday entries are valid, and their distribution is shown in figure 17.

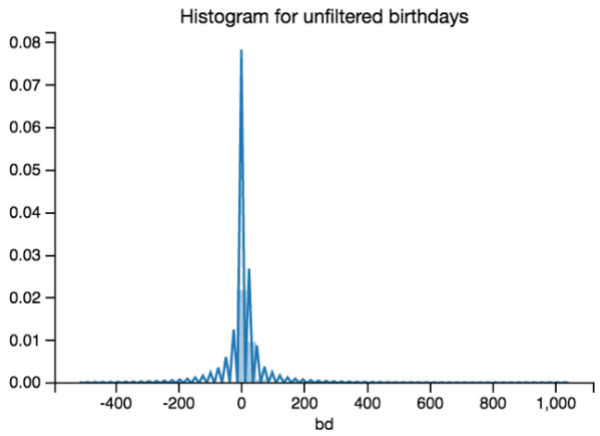


Figure 16: Histogram of unfiltered birthdays

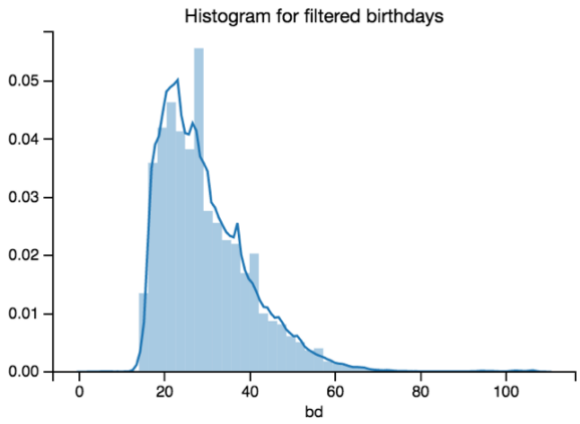


Figure 17: Histogram of filtered birthdays

Gender

The gender column contains information about the gender of the user, i.e. if the user is female or male. 35% of the values related to this feature are missing (*NaN* values). The histogram in figure 18 shows that there are a similar number of females and males (slightly more males).

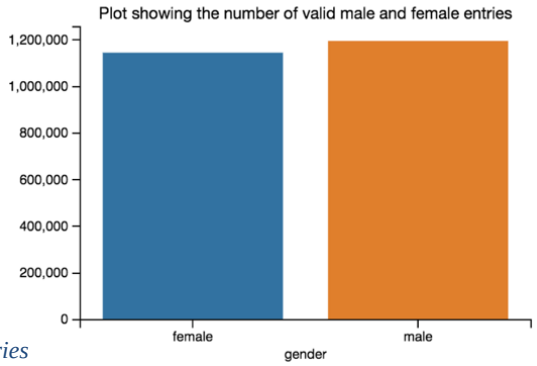


Figure 18: Distribution of valid male and female entries

Registered_via

The feature registered_via provides information on the registration method the user chose to sign up with KKBOX. It is encoded as an integer from [1, 19], so there seems to be 19 different methods for registration.

As figure 19 shows, most people are registered via method 4, followed by 3, 9 and 7. The other methods are not used frequently.

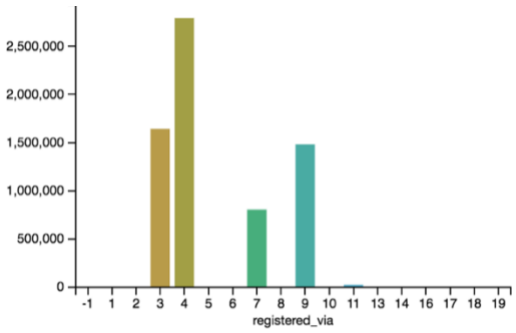


Figure 19: Method members registered via

Registration_init_time

Registration_init_time contains information about the registration date of the users. The data is stored as follows: *YearMonthDate* as an integer (for example, 20180926 represents 26.09.2018). We need to transform the integer entries to a date type. All the entries in the dataset have a registration_init_time.

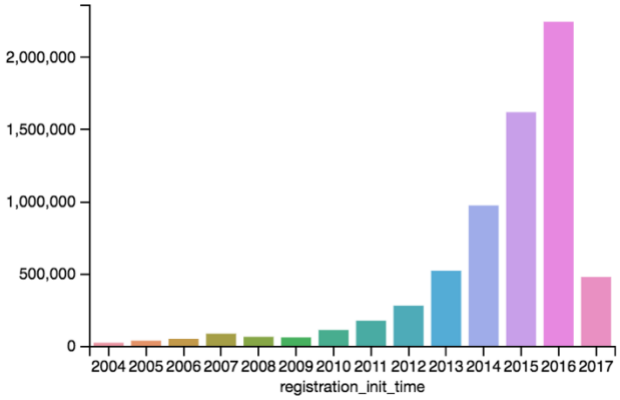


Figure 20: Registration year

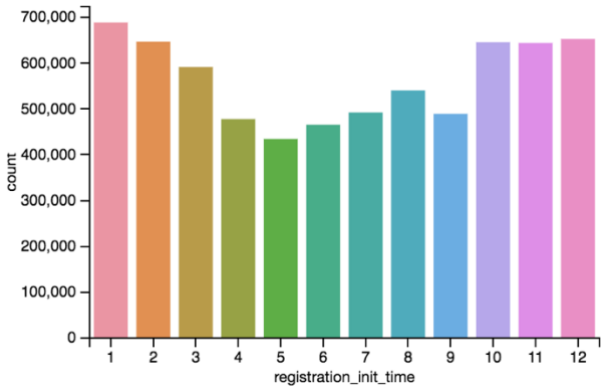


Figure 21: Registration month

Figure 20 demonstrates that the new memberships constantly increased and reached their peak in 2016. In 2017, there is a decline in the rate for new memberships. It is possible that strategy of KKBOX is not that aggressive anymore and expanding the business is not the main focus. People tend to get registered more often in the first and last quarter of the year, as shown in figure 21. There is also a slight increase in new registrations during weekends. During the weekdays, the number of people who start using the service is constant.

3.3.2 CHURN RATE AND FEATURE RELATION

In this section, we show the correlation between individual features of the members file and the response variable *is_churn*. 6,6% of the users that are mentioned in the members data-set churn their subscription.

Is_churn and gender

The gender seems to have almost no influence on whether a user churn or not. Figure 22 shows how many males and females churn respectively. Figure 23 visualizes the actual churn rate (in %) for females and males. Both genders churn around 91% of the time.

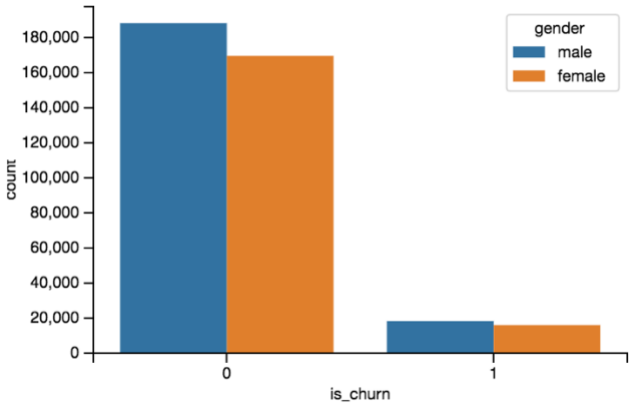


Figure 22: Churn partitioned in gender

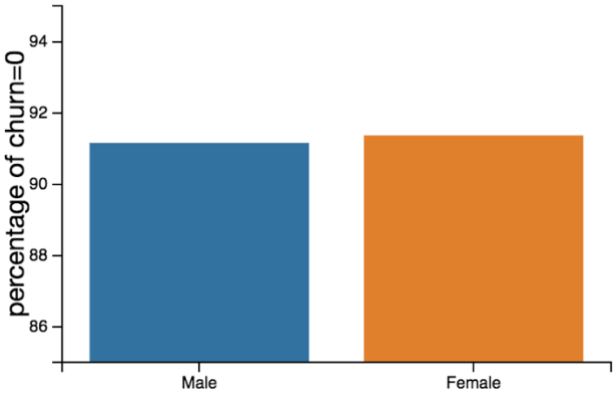


Figure 23: Gender compared to no churn

Is_churn and registration_init_time

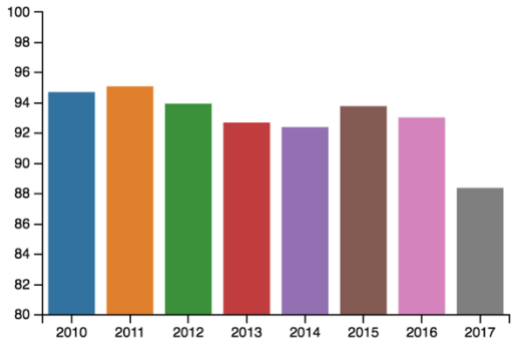


Figure 24 shows that the percentage of churn=0 is decreasing (especially in 2017), suggesting that a subscription made in 2017 is more likely to churn than a subscription made in earlier years. It makes sense that long-term users are indeed less likely to churn than new users.

Figure 24: Churn in relation to registration year

Is_churn and bd

There seems to be a relationship between the age and the churning rate. Indeed, figure 25 shows that the old people are less likely to churn than young people. The churning rate seems to be particularly high for users aged between 18 and 30.

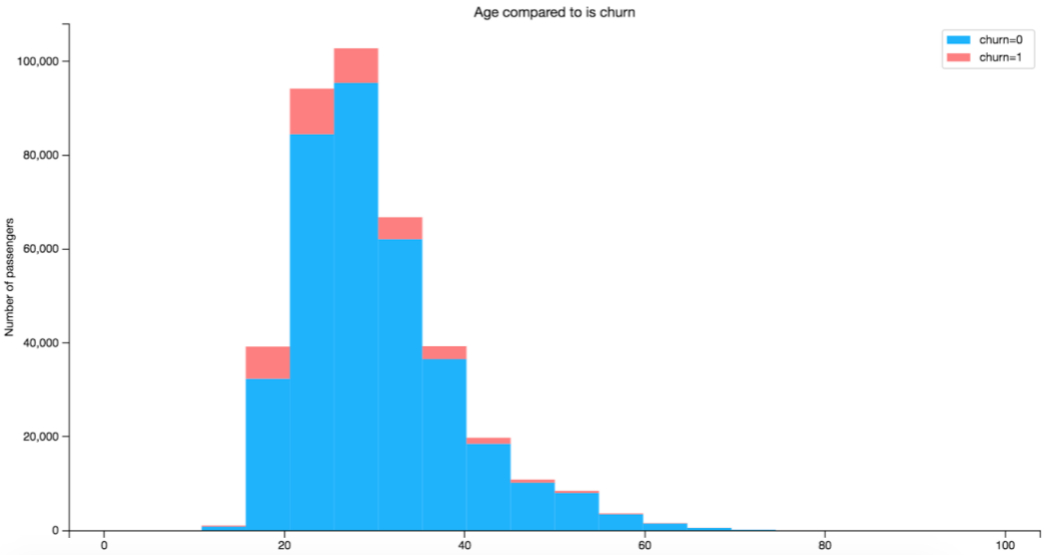


Figure 25: Age compared to churn

3.3.3 CORRELATION ANALYSIS

Table 8 shows the correlation between city=1 and birthday=0. As already mentioned before, we assume that they are the go-to values if there is no information available (instead of *NaN* values). A correlation of 0.95 seems to reinforce our assumption.

Table 8: Correlation

Feature 1	Feature 2	Correlation
city = 1	bd = 0	0.95

4 FEATURE ENGINEERING

This section introduces the features that act as inputs for the predictive modelling algorithms. Feature engineering is the process in which domain knowledge is used to extract relevant information out of raw data that is important to make an accurate prediction. Here, the domain knowledge is acquired though a detailed data exploration shown in section 3.

In our feature engineering, we distinguish between features that show change in user behavior over time and those that are absolute (i.e. none time-related features). Features that measure change in user behavior are generally those that describe its listening habits over time *(user_logs*) and its purchasing history (*transactions)*. Absolute features are categorical features such as gender, registration method, payment method, plan days, etc.

In order to correctly make use of the features showing change in user behavior, we had to train the data under the exact same conditions as we tested them on. KKBOX provided churning data for February 2017 (*train.csv*) and March 2017 (*train_v2.csv*). Hence, we had to divide the time related features into 3 datasets for training, testing and predicting. The 3 datasets are listed below:

- **Training**: input data until January 31st, 2017, churning data from February 2017
- **Testing**: input data until February 28th, 2017, churning data from March 2017
- **Prediction**: input data until March 31st, 2017, predict churn for April 2017

Hence, to create *Training,* we had to remove all elements from the datasets that where marked with a date past January 2017. To create *Testing,* we simply used all the v1 datasets. Finally, for *Prediction,* we included both data from the v1 and v2 (March 2017) sets.

For further reference, we will refer to *Training* as "*data-feb*", to *Testing* as "*data-mar*" and to *Prediction* as "*data-apr*".

We had to make sure that date related features where relative to the month currently looked at for churning. Indeed, a feature called *entries per user Feb-17* should have the same meaning for *is_churn* in March 2017 as the feature *entries per user Mar-17* for *is_churn* in April 2017 and thus should have the same name (e.g. *feature_m1)*.

For categorical features, we have used one-hot encoding. If an attribute is one-hot encoded, a new column is generated for every possible value that the attribute could take.

4.1 TRANSACTIONS

The transactions dataset contains information about transactions from January 2015 up to April 2017. As we have predicted churn for April 2017, the most interesting transactions are those in the months before April. In feature engineering of transactions, we have grouped the dataset on *msno*, resulting in a dataset with one entry for each *msno*.

The initial features *membership_expire_date* and *transaction_date* have been dropped and replaced with the features *membership_duration* and *diff*. The features *actual_amount_paid* and *plan_list_price* was also dropped because they did not add any improvements to our prediction.

4.1.1 FEATURE CREATION

Listed below are the features we created from the initial *transactions* dataset.

Membership duration (*mem_duration*)
mem_duration is a feature obtained from *membership_expire_dates* and *transaction_dates* and represents the value of membership duration for each row. Recall that a single user can appear in many rows. To have a single value for each user, this is then grouped by *msno*, adding the values of this feature for each given user. The resulting difference is in terms of days, represented by an integer.

No auto renew and cancel (*notAutorenew_&_cancel*)
We also extracted a new binary feature from *is_cancel* and *auto_renew*. This feature is set to 1 if ($\overline{auto\_renew}\ AND\ is\_cancel$) is true, thus, if *auto_renew* = 0 and *is_cancel* = 1.

This feature can be useful to ease the algorithm the search of a proper model for our data as we are only setting to 1 the rows that, a priori, have more probability to churn due to the nature of *auto_renew* and *is_cancel*.

Number of cancellations (*is_cancel_number*)
This feature has been extracted from *is_cancel*, simply counting the number of times a user has cancelled a subscription. After grouping, we obtain for each unique user, the number of times he/she cancelled. As shown in section 3.1, the data exploration of the transactions, rows with is_cancel = 1, has a higher probability of churning. By counting up the number of cancellations of each user, we assume that users with a higher number of cancellations are more likely to churn.

Difference (*diff*)
The feature diff has been computed for each of the three datasets January 2017, February 2017 and March 2017. It is an integer that shows the number of days since the last expiration date (if the number is negative) or until the last expiration date (if the number is positive) for each *msno*, with respect to the last day of the month.

This feature is directly related to the definition of churn. If days since last expiration date is -30 or more, we know for certain that the user has churned because there is no valid subscription within the 30 days since the last expiration date.

Payment method churn (*pay_method_churn*)
There are three payment methods that, according to data pre-processing, give users a higher probability to churn. These are: 32, 35 and 38.

To take into account this and give importance to this fact, we created this feature that is set to 1 when one of these methods are found for each row. Finally, it will be added when grouping by *msno*.

4.1.2 ONE-HOT ENCODING

Payment method id
We one-hot encoded *payment_method_id*. Indeed, payment method is a categorical integer variable. As such, the value in "*payment_method_id*" has no actual numerical meaning. In this case, it is used to describe the payment method of the user. Alternatively, the payment methods could also have been designated by their actual name, or a different categorical value, such as "A", "B", etc. Summing up the values would make no sense. In addition, a payment method containing a high value does not have a higher importance.

In our data exploration, we observed that only 5,3 % of the users use payment methods below 30. Hence, we have one-hot encoded the payment methods from 30 to 41 and dropped those below 30. Since we now only have 11 different payment methods, one-hot encoding generates 11 columns containing a 1 or a 0 based on the payment method. There will only be one "1" in each row, and the rest will be "0".

Payment plan days
Here we used the same procedure as for *payment_method_id*. In our data exploration, we identified the most used payment plans and after one-hot encoding, all payment plans except 410, 195, 180, 90, 7 and 30 were dropped. These payment methods are the most common ones and most common between the whole set of users.

All the feature above (from *transactions* data frame) were **grouped by *msno***, adding the values of all features for each given user and dropping features that are not useful: *plan_list_price*, *actual_amount_paid*, *transaction_date*, and *membership_expire_date*.

4.2 USER LOGS

4.2.1 DATA PROCESSING

The *user_logs.csv* file is 30.51 GB and thus, due to memory restrictions, cannot be read in by normal computers in one go. Therefore, we read in the file by chunks of 10^8 bytes and append them iteratively. Some of our features are derived from the whole dataset and thus need to be generated during the loop. This, in particular, includes all date-related features.

4.2.2 TIME RELATED FEATURES

Our data exploration showed that the number of entries, listening time and listening patterns (in terms of skipping songs) all have an effect on churning. To capture changes over time and relate those to the month we have to predict *is_churn*, we created a set of date-related features.

Entries per user in a given month
Besides adding the overall amount of *entries per user*, we also added a feature that gives the number of *entries per user* for a certain month. In the data exploration, we only generated *entries per* user for 5 million rows and got a range of 1-16; this time the maximum value is 790. E*ntries per user per month* is given for the months Mar-17, Feb-17, Jan-17, Dec-16 and Nov-16. This allows us to quantify changes in user activity in the months prior to the month where we should predict churn. For each of our 3 datasets, we added the 3 months before the output month and labeled them *entries_m1, entries_m2, entries_m3.* In the data exploration, we have seen that the likelihood of churning depends a lot on the number of entries of user, so we believe this feature will be promising.

Total seconds in a given month
Similar to the feature above, we have summed monthly total_secs for the months Nov-16 to Mar-17 to inspect for drops in listening time in the months prior to the output month.

Behavior
This feature describes the ratio of skipped songs to songs played fully out. 'Behavior' is given by: $\frac{\text{num}_{25}+ \ldots + \text{num}_{98.5}}{\text{num}_{100}}$. Thus, 0 would mean that a user listens to every song entirely. Again, in Data Exploration we have seen that churning and non-churning customers have a slightly different skipping behavior. So, this feature allows us to effectively quantify that behavior.

Ratios
In order to quantify the trends in listening behavior, we have created different ratios comparing the behavior of the user in different months. We created ratios for number of entries in a month and number of seconds played in a month. For both features we created the ratios current month/previous month and previous month/previous, previous month, giving a total of four ratios. Note that current month always depicts the month before *is_churn* is given.

4.3 MEMBERS

In the data exploration of the members-file, we concluded that certain variables, such as the birthday and the city users live in, have an influence on whether people churn or not. As the quantity as well as the quality of the features have a large influence on the quality of the outcome of the algorithm, it is important to carefully generate and test these features.

4.3.1 ONE-HOT ENCODING

City
To include information about the city of the user as a feature, we one-hot encoded the variable "city". Before we handled information about the city to the algorithms, we needed to make it a binary. The

variable "city" contains 21 different cities. As a result, one-hot encoding generates 21 columns containing only 1's and 0's. If a customer lives in city 3 for example, only the newly generated column "city_3" will contain a 1, while the other 20 columns contain a 0. Section 6.1 provides details on how different features are selected for the final model, as it is not necessary to include every single one of them.

Year of registration

In this part, we one-hot encoded the year of registration. As long as a date is not calculated in relation to another date, the value can be considered as a categorical one. First, we need to convert the integer date to the data type "date". This allows us to extract the year. As we show in the data exploration, the chance of churning is higher for short-term users. We consider the dates until the year 2012. This is due to the fact that we don't have a lot of data on the years before 2011. Adding these years as well will only make our algorithm slow and generate unwanted noise. It is crucial to provide only important data to the algorithm and avoid spamming it with non-relevant information or columns that do not contain data about many users.

Registration method

Next, we one-hot encoded the registration method. Similarly, to the "city"-variable, this feature contains integer values that have no numerical meaning. The algorithm will not be able to understand what the value in "registered_via" represent, since has no actual meaning. It will add a higher weight to large values, which would be wrong. Taking the data exploration into account, we decide to get rid of certain values for "registration_via" as they only represent a very small fraction of the data.

4.3.2 NORMALIZATION

Birthday

The age of the users also plays a role in the likelihood of churning, as it was mentioned in the data exploration. First, we get rid of the outliers by setting all the values that are smaller than one and larger than 90 to 24. 24 corresponds to the median of the valid birthdays. We made the assumption that a user is between 1 and 90 years old. Next, we normalize the data (min-max normalization). The data is normalized, because the range of the values for our raw data varies widely between different features. To prevent a bad effect on the performance of the algorithm, we took the precautionary step to normalize it. In fact, some objective functions will not work properly if data is not normalized.

5 ALGORITHMS

We have tested four different supervised learning algorithms for training a model on the datasets, namely Logistic Regression, XGBoost, AdaBoost and Random Forest. This section presents the theory behind these algorithms. We also present the evaluation metric *logloss* that we used to evaluate the performance of the different algorithms.

5.1 LOGISTIC REGRESSION

Logistic regression is a regression model that can be used to create classifiers. The base of this model is the logistic function that takes any real-valued number and maps it to a number between 0 and 1. The input is linearly combined using weights and coefficients and then transformed with the logistic function to predict an output value. The coefficients of the linear regression model need to be learned from the training data. To determine the best coefficients, maximum likelihood estimation is used. This procedure seeks to minimize the errors in the predicted probabilities ("Logistic Regression for Machine Learning", 2016 [1]).

5.2 XGBOOST

XGBoost is an extreme gradient boosting library that uses a tree ensemble model to make predictions. The training model creates Classification and Regression Trees (CART) to generate a prediction for each input. In a CART-tree each node gets a real score associated with it. One tree will most likely not be enough to make a prediction. The tree-ensemble model combines the trees together and make final score to create a prediction.

As for supervised learning in general, the goal of XGBoost is to optimize an objective function. In XGBoost, the objective function consists of a loss function and a regularization function. An additive training strategy is used to optimize the objective function. Because of the complexity of each tree, all trees cannot be trained at once. Instead, one tree is added at a time, and the algorithm adjust after what it has learned from that tree. Next, the algorithm add a new tree to the model. In order to pick the next tree while building the model, each tree is given a structure score computed from the loss function. This score describes how well suited the tree structure is. Pruning is then used to optimize the trees and the regularization function keeps the complexity of the tree under control and prevents over-fitting. ("Introduction to Boosted Trees", 2016 [2]).

In our prediction we have used the Python API for the XGBoost library. This enables us to fine tune parameters. The parameters of XGBoost can be divided into three categories, general parameters, boosting parameters and learning task parameters. To improve the model, we have focused on the boosting parameters such as the learning rate (named eta), max tree depth and the number of boosting iterations. By fine tuning the parameters, we are able to fit the model to meet the requirements of our prediction.

5.3 ADABOOST

AdaBoost, or Adaptive Boosting as is its full name, is a meta-estimator that tries to create a strong classifier from multiple weak classifiers. Weak learners are models that have accuracy just above random guess at a classification problem. AdaBoost starts by fitting a classifier to an original dataset, then it fits additional copies of the classifier on the same dataset. It is adaptive in the fact that the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases. AdaBoost is sensitive to noisy data and outliers.

AdaBoost was the first successful boosting algorithm developed for binary classification. ("AdaBoostClassifier", 2017 [3]).

5.4 RANDOM FOREST

Random Forest is an ensemble-learning method. It constructs multiple decision trees during training time and output the mean prediction of the individual trees. The algorithm brings randomness into the model when it grows the trees, which are generated based on a random subset of features. The algorithm uses voting on classifying new instances. In the end, it chooses the classification result having the most votes over all the trees in the forest (Pan, 2018 [4]).

Random Forest and Boosted trees are not different in model, but in how they are trained. ("RandomForestClassifier", 2017 [5])

5.5 EVALUATION METRIC

To evaluate the performance of the different models, we use *logloss* as our evaluation metrics. This evaluation metrics shows how much the predicted probability diverges from the actual label. For this Kaggle competition, it is particularly important to evaluate our algorithms with the *logloss* evaluation metrics, because a very high percentage of the users do churn, and thus the problem have an unbalanced class distribution. This is also the reason why accuracy is not used as an evaluation method. The accuracy would be misleadingly high even for a bad prediction, and it would be very difficult to see if a new feature or a change to the algorithm parameters actually improved the prediction. Kaggle also uses *logloss* as evaluation metric for our submission. Equation 1 shows the evaluation metric used to evaluate our predictions.

Equation 1 The logloss evaluation metric.

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

6 METHODOLOGY

6.1 FEATURE SELECTION

This section introduces the process of selecting important features for the final prediction model. First, features that according to our data exploration have a small or non-existent impact on the churning rate are removed. Next, after merging the features of the three files (i.e. *transactions*, *members* and *user-logs*), every feature is tested for its influence on the *logloss*.

For the members file, a total of 61 different features have been generated. This large number of features is mainly due to one-hot encoding, which creates a new column for every value that is encoded. However, the data exploration of the members file (see section 3.3) shows that many features contain variables that have no impact on *is_churn* or represent only a very small part of the users. As such, these features can be removed such that 33 features (including *is_churn* and *msno*)

are kept. The features for the transactions file are preselected in a similar way. Out of 42 generated features (including the 9 original ones), 28 features are selected (including *is_churn* and *msno*). None of the 21 features of the *user_logs* file are removed.

Next, the three files are merged (on msno), which generates one large data frame containing a total of 78 preselected features. To decide which features we should keep and which ones we should drop, we need to test their efficiency. As such, we run loops that remove one distinct feature in every iteration and test the *logloss* for each set. This way, we will find out if leaving aside a particular feature will improve our result. We test these sets with XGBoost.

Once we find features that, when removed, indeed improve the test score, we also test them by submitting them to Kaggle. That is because sometimes the test score might improve but the Kaggle score doesn't (due to differences in training data size). If we find the feature whose removal improves the score the most, we repeat the process by iteratively dropping each individual feature from a set where the previous feature has already been removed. We repeat this as long as the score keeps improving.

By doing the above we achieved the following sequence of Kaggle score improvements:

Initial Dataset	Removing *mem_duration*	Removing *is_auto_renew*	Removing *payment_method_id_36*
0.12319	0.12215	0.12195	0.12120

Table 9: Kaggle Score Improvements

After removing *payment_method_id_36*, no additional feature that, when removed, gave a better test score, actually improved the Kaggle score and thus we stopped our effort here.

Note that we also tried pre-processing method, such as max-min-normalization. But, as XGBoost does not require data normalization when using trees, this did not improve our result.

6.2 GENERAL METHODOLOGY

Before we could actually start working on the competition, we had to import the three data-sets. This was particularly challenging for the user-log file reaching a size of over 30 GB. In fact, our personal computers did not have enough random-access memory to load the file and trying to do so resulted in a memory leakage. To overcome this problem, we needed to implement a loop that loaded the file chunk by chunk. As in the end, we would have one row per user (summarizing all his up to 790 listening sessions), the file would eventually become much smaller. This size reduction was realized during the loop, where we also created all the features that relied on a per-log-basis, such as *entries* and *total_secs* for certain months.

The next step in addressing the churn prediction challenge of KKBOX was gaining a solid understanding of the data. This was achieved by extensive exploration of the three datasets. Here, we laid our focus on the four cornerstones of data exploration, i.e. a detailed inspection of distribution, composition, relationship and comparison of the features.

Next, we applied the conclusions from data exploration part to feature engineering. In particular, we removed outliers and generated various new features. At this point, the datasets were still separate and testing the efficiency of different feature combinations yielded outcomes that where not coherent with testing them on the whole dataset and thus were suspended.

Merging the datasets with the output *is_churn*, which is contained in the train files, required special due diligence. We had two train files, containing *is_churn* for February and March 2017, respectively. Our task was to predict churn for April 2017. As our input data only extended to March 2017, we had to make sure that for each train/test set, we only use data that extends to the month prior of the month where *is_churn* is given (to create the same conditions than for the final prediction).

After merging the three datasets and creating the *data-feb*, *data-mar* and *data-apr*, which are based on the three different months of churn prediction, and intelligently replacing all Null values, we could start testing features and algorithms. As we already have 2 datasets with output labels for different months, we didn't have to create train and test sets manually but could simply train on *data-feb* and try to predict *is_churn* of *data-mar*.

To address the problem of over-fitting, we compared the test score of prediction of unseen data versus prediction of the training data. While the predictions were always better for the training data, XGBoost's over-fitting was not excessive and we further reduced it by fine-tuning its parameters.

Finally, after having selected an optimal batch of features, we created a new train set to create our prediction for April 2017. Indeed, having an output label for two different months as well as a set of features that are relative to the date, allowed us to generate a trainset where we combined both the outputs from March and April 2017. Thus, by simply appending *data-mar* and *data-feb*, we generated a dataset practically double the size of the original *train.csv* set. This has resulted in a substantial decrease of our *logloss*. Indeed, training on *data-feb* and *data-mar* yielded a *logloss* of 0.13091 and 0.13411, respectively. Appending the datasets notched the score up to 0.12120.

While we could already see that XGBoost yielded the best predictions, our last step was to tweak the parameters of the XGBoost function. This has further improved our score to a final 0.11859 (see section 7.2).

7 RESULTS

7.1 RESULT HISTORY

Our final prediction shows that XGBoost yields the best result and that logistic regression performs the worst for our prediction model. The bad performance of logistic regression might be due to the fact that it is sensitive to the distribution of the class labels in the training and the test set. The algorithm assumes that the distribution of the class labels of the two sets are the same and performs poorly when they are not.

Furthermore, our results show that the algorithms making use of boosted trees, i.e. AdaBoost and XGBoost, perform the best. Both of them combine several weak learners into a strong prediction model. They make use of a boosting technique where several weak learners are trained in sequence, and the current trained tree is informed about the wrong predictions of the previous tree. The ability to propagate information about error increases the probability of making a correct prediction. However, XGBoost has larger computational capabilities and exploits the capabilities of the CPU to a lager extent than AdaBoost. Hence, its effectiveness results in a more precise prediction than AdaBoost. Lastly, the XGBoost library offers a great amount of customization when it comes to parameter fine-tuning. In section 7.2 we show how we reduced the error by optimizing the set of parameters.

Random forest performs worse than XGBoost and AdaBoost. This might be due to the foundation technique of random forest is bagging rather than boosting. Hence, the wrong predictions made by one tree is not propagated onto the next tree. Instead, the algorithm outputs the mean prediction of the random built trees.

Table 10 shows the Kaggle scores yielded by the four different algorithms on the original dataset containing 78 features.

Table 10: Kaggle scores of 78 features.

XGBoost	Ada Boost	Logistic Regression	Random Forest	XGBoost after fine-tuning
0.12120	0.12936	1.22667	0.41332	0.11859

As shown in section 6.1, the Kaggle score improved by removing features that made XGBoost perform worse. Figure 26 shows the screenshot of our final submission to Kaggle.

Figure 26: Final Kaggle score.

The most important features in our final XGBoost prediction model is notAutorenew_and_cancel, is_cancel and diff. Their respective importance is shown in table 11, listing the top ten features split on by XGBoost. We see that the top tree features are closely related to churn. A user has to actively renew their subscription if it is cancelled and they don't have auto renew. Hence, if they don't renew, they become a churned user.

Table 11: XGBoost Feature importance

XGBoost	Importance
notAutorenew_and_cancel	21.90%
is_cancel	19.05%
diff	18.10%
cVp_entries	5.24%
payment_plan_days_30	4.76%
payment_method_id_41	4.29%
pay_method_churn	3.81%

secs_m1	3.33%
payment_method_id_39	3.33%
entries_m1	2.86%

7.2. XGBOOST PARAMETER FINE-TUNING

This section focuses on the fine-tuning of four different parameters of XGBoost. By modifying the values of some parameters of the algorithm, we were able to improve our Kaggle score from 0.12120 to 0.11859 (top 8.3% of the participants of the competition). XGBoost uses many different parameters that guide the overall functioning and the optimization of the algorithm, as well as the boosting at each step. Here, we focus on the following parameters: max_depth, eta, num_class and num_round. The max_depth parameter specifies the maximum depth of each tree and is used to control over-fitting. A large depth allows the model to learn relations in a very specific manner for particular samples. Eta is the training step for each iteration and can make the model more robust by shrinking the weights of each step. The parameter num_class represents the number of classes for the data-set. Finally, num_round represents the number of training iterations.

Table 12 provides an overview of different parameter settings that were tested. In total, 27 different settings for the parameters were implemented. This empirical test shows that the best Kaggle score can be obtained for the following set of parameters: *{num_round=15, num_class=10, eta=0.3, max_depth=3}*. The number of classes and training iterations have the largest impact on the improvement of the score. In addition, the maximum depth of each tree is also a very important parameter for our model since we encountered the problem of over-fitting. We need to limit the tree depth to 3, as a larger value leads to a worse score (probably caused by a larger impact of over-fitting).

Table 12: Parameter fine tuning results.

Num_round	Num_class	Eta	Max_depth	Kaggle Score
20	3	0.3	3	0.12120
50	3	0.3	3	0.13998
13	3	0.3	3	0.12609
15	3	0.3	3	0.12136
15	5	0.3	3	0.12022
15	**10**	**0.3**	**3**	**0.11859**
15	15	0.3	3	0.12123
15	10	0.2	3	0.12160
15	10	0.4	3	0.12422
15	15	0.3	5	0.12939
15	15	0.3	2	0.13123

8 CONCLUSION AND OUTLOOK

In summary, we used the extreme gradient boosting library XGBoost to make a classification model for KKBOX's Churn Prediction Challenge. Our final prediction yields a score of 0.11859 and is ranked in top 8.3% of the participants.

To achieve this score, it was crucial to understand the data given by Kaggle and KKBOX. As such, we analyzed and visualized every single feature, and we looked into the relationships between the features and the response variable *is_churn*. This process allowed us to engineer relevant features that were used as inputs for XGBoost. One of the biggest challenges of the competitions was related to the large size (over 30 GB) of the user-logs file. This required extra care, as most of the features had to be generated during the loops that iteratively loaded the file.

Future work includes noise reduction in the training data and further fine-tuning of the parameters to reduce the negative impact of over-fitting. Even though we took certain measures against this problem, such as increasing the amount of training data by merging the data of the month of March and February, we sill detected over-fitting. However, with its build-in capabilities to reduce the impact of over-fitting, XGBoost outperformed every other algorithm that we tested. Nonetheless, we are certain that the score could be improved by applying measures that further reduce the impact over-fitting.

In addition, the score could be improved by generating additional features. Those could especially include new features that use the data of the 3 different datasets concurrently. To further improve the score ensemble learning between the different algorithms we have used can be implemented.

For the whole team this was our first hands-on experience with Machine Learning. It has been an exciting experience to apply the theory learned in the lectures and tutorials. We have improved both our problem-solving skills and teamwork abilities. The project has given us new insight into the field of Machine Learning. Also, the competition has given us motivation to constantly improve our score.

9 REFERENCES

[1] Brownlee, J. (2016, September 22). Logistic Regression for Machine Learning. Retrieved 23 April 2018, from https://machinelearningmastery.com/logistic-regression-for-machine-learning/

[2] Introduction to Boosted Trees. (2016). Xgboost.readthedocs.io. Retrieved 23 April 2018, from http://xgboost.readthedocs.io/en/latest/model.html

[3] AdaBoostClassifier. (2017). Scikit-learn.org. Retrieved 23 April 2018, from http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[4] Pan, S. J., (2018). Ensemble Learning, lecture slides, CZ4041 Machine Learning, Nanyang Technological University

[5] RandomForestClassifier. (2017). Scikit-learn.org. Retrieved 23 April 2018, from http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html