CHALMERS
UNIVERSITY OF TECHNOLOGY

Predicting retention among application users with online ensemble learning models

Master's thesis in Engineering Mathematics and Computational Science

OLOF EKBORG

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018

Predicting retention among application users with online ensemble learning models

OLOF EKBORG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2018

Predicting retention among application users with online ensemble learning models
OLOF EKBORG

© OLOF EKBORG, 2018.

Supervisor: Rebecka Jörnsten, Mathematical Sciences
Examiner: Rebecka Jörnsten, Mathematical Sciences

Master's Thesis 2018
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by the Department of Mathematical Sciences
Gothenburg, Sweden 2018

Predicting retention among application users with online ensemble learning models
Olof Ekborg
Department of Mathematical Science
Chalmers University of Technology

Abstract

Most service providing companies consider customer retention as the most important asset for improving profitability. Even for services and applications without paying customers the retention of users is essential, as more advertisement impressions are generated and the reputation of the brand strengthens. The ability to foresee which users will be retained and which are likely to churn is therefore highly valuable for any expanding company.

Forza Football is one of the world's most popular football live score applications with millions of weekly active users. New data of application activity among users arrives sequentially in the form of a stream. To predict future user activities, a model must be able to adapt to seasonal drifts in activity. The model must furthermore remain scalable and time efficient when analyzing new instance arrivals, given that the size of each instance is several million observations. Motivated by these requirements, this thesis approaches a data stream of previous user activities to predict the activities of upcoming instances. State-of-the-art ensemble classification methods are acclimatized to an online learning environment to incorporate both historical and current information in a computationally low-cost manner.

Various predictive models are proposed which obtains accurate predictions that are efficient in terms of storage and computational time. The models are stable in detecting and adjusting to concept drifts.

Keywords: Online learning, Data stream analysis, Concept drift, Random Forest, Decision tree ensembles, retention prediction, churn prevention.

Acknowledgements

First of all I wish to thank my supervisor Rebecka Jörnsten for her guidance and many insightful discussions throughout this project. I would also like to express my gratitude towards all employees of Forza Football for instantly welcoming me as a member of the team, and especially Andreas Rolén and Gustaf Rasmusson for always havig the time to discuss data related questions. Furthermore, I would like to thank Eva Wilhelmsson for introducing me to Forza, as well as Per Ljung and Jozef Brodala for their feedback. Finally, I wish to express my appreciation towards Pernilla Tanner for always encouraging me and believing in my potential.

Olof Ekborg, Gothenburg, August 2018

Contents

List of Figures

List of Tables

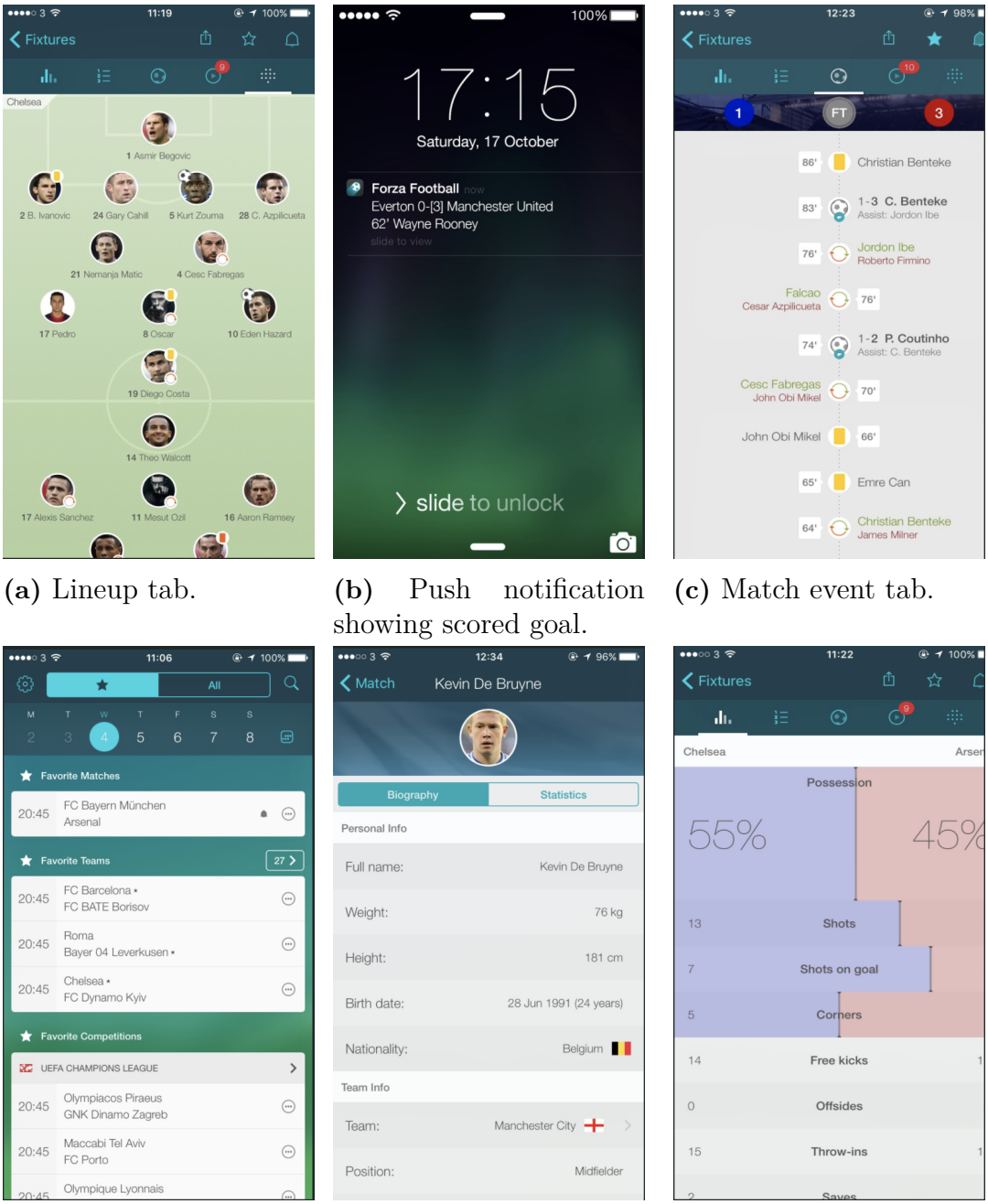
1
Introduction

Many companies today invest much of their time, energy and resources on acquiring new customers. While this is an important strategy for expansion, many argue that customer retention is the most important asset for improving customer profitability [1],[2], and Fred Reichheld claims that in financial services, a 5% increase in customer retention would produce more than a 25% increase in profit [3]. For businesses with subscribing customers, such as banks, insurance companies, telecommunication companies etc., it is indicated that long-term users buy more and cost less to attend (see [4] and [5]), whereas replacing existing customers with new recruits is known to be an expensive and risky strategy (e.g. see [6], [7], [8], and [9]).

Even for services without subscriptions or purchases available, retaining users is essential. For many websites and smartphone applications, a major source of income is selling advertisement spaces to other companies. Having a large population of users available for ad impressions are therefore of importance when negotiating for ad spaces. It is therefore an initial objective for a company to retain the current customer base to be able to grow, and the asset of targeting customers in danger of churning, i.e. becoming inactive, is highly sought after among businesses with the interest of expanding.

1.1 Background

Forza Football [10] is a free smartphone application, hereafter referred to as the app, which is available for both Android and iOS. The app provides the user with live scores, lineups, push notifications, opinion polls and video highlights for more than 800 football leagues and cups around the world. The user customizes which teams and tournaments they are interested in to receive fast, detailed updates and coverage of their favorite teams in particular, or football in general. Some screenshots are provided in Figure 1.1 to illustrate a general idea of the design and main features of the application. The figure includes various screenshots of various essential tabs and pages of the app, such as the tab providing the lineups of the respective team involved in the match (Figure 1.1a), a push notification received when a goal is scored in a match (Figure 1.1b), the tab listing the occurred events of a match (Figure 1.1c), the tab providing the kickoff times and dates organized in calender format (Figure 1.1d), the page presenting information of a certain player (Figure 1.1e), and finally the tab summarizing the statistics of a match (Figure 1.1f).



(a) Lineup tab. **(b)** Push notification showing scored goal. **(c)** Match event tab.

(d) Calender tab. **(e)** Player page. **(f)** Match statistics tab.

Figure 1.1: Various example screenshots of features available in the Forza Football app. Accessed 2018-06-07 from [10].

Since Forza has no payments, churning cannot be defined as an interrupted subscription. For free apps striving towards users returning on a daily basis, such as online games or news applications, a user can be considered as churned when the app has not been activated during a consecutive amount of days. However this definition is irrelevant in the circumstances of football live score, as it might be days, weeks or even months between matches of interest for a user. A user may hence be dormant for a longer period without being at risk of churning, and encouraging a

user to be active in the app during the off-season might cause a poor user experience and decrease the retention time for the user.

As aforementioned the intermissions between matches is not constant, but some obvious seasonalities can be recognized. For national club teams, a yearly seasonality is distinct with matches played regularly during a couple of consecutive months, with start and end date altering for each unique league. During each season of the league, the matches are played on a rather weekly basis, with many leagues playing only during weekends.

In an international perspective, the seasonality commonly follows a quadrennial pattern for major tournaments such as the FIFA World Cup [11], UEFA European Championship [12], and Copa América [13].

1.2 Objective

The main objective of this thesis is to predict the upcoming weekly activity in the app for each unique user. With a reliable prediction model, the user obtaining false positive predictions can be tracked in order to prevent users from churning. Such a model would, given the seasonalities affecting the user activity, optimally be trained on data from multiple quadrennial sets of the past years to cover the full cycle of relevant football tournaments. However the availability of data is limited since only the last few years are covered. Furthermore computational limitations are present, given that several users are active every week and all computations are performed on a MacBook Pro running macOS High Sierra with a 2,3 GHz Intel Core i5 CPU. Given these limitations, an optimal training scenario is not feasible and an additional objective is introduced to provide an online extension of the prediction model with the purpose of adapting the model to the concept drift distinguished in the football season cycles.

The remaining disposition of this thesis is organized as follows: Chapter 2 motivates how the dataset used for the prediction was constructed, and presents the explanatory variables considered. A detailed analysis of the available data is provided and thoroughly discussed. Chapter 3 presents the theoretical foundations of the statistical models used for prediction. In Chapter 4, the models constructed to predict which users will view at least one match during the upcoming week is presented and motivated. The models are essentially various ensembles of decision trees for which poor performing trees will be replaced by new trees trained on the most current instance. Chapter 5 illustrates the results obtained from implementing the models, and provides a detailed analysis and discussion regarding the obtained findings. Finally Chapter 6 concludes the findings and summarizes the thesis.

2
Data analysis

The data considered for this thesis is obtained from the database appurtenant to Forza Football. The database consist of multiple datasets of all trackable events registered during each session in the app, stored metadata (such as team, tournament, and match information etc.), and user specific data, including aggregated activities during different time periods (such as the last day, last week, last 4 weeks, last 90 days and last 180 days, and all time).

Given that every single event performed by each unique user is registered in the database, the amount of available data is massive and a majority of the information will be redundant for the main objectives of this thesis. The selection of explanatory variables will therefore be thoroughly investigated to scale the data from irrelevant additions, and some constraints are invoked in order to reduce the sample size further. The data stream covered in this thesis will ultimately consist of the weekly activities of 8,761,252 unique users during the period from 2016-05-01 until 2018-05-01.

2.1 Response variable

The objective of this thesis is to predict the activity related to match views during the upcoming week for each unique user. A match can be viewed in the app both pre match, during the match, and post match. The reason for a user not being active in the app during a match may not only be restricted to low interest of the match. Various preoccupations, poor internet service, or simply watching the considered match live might restrict the user from being active in the app during the match, even though an interest in the match is existing. Considering all stages of viewing matches are therefore of interest, as the pre and post match information is still valuable for users not being able to follow actively during the match. The response variable treated in this thesis is however restricted to only consider views registered during the same week as the match day, as viewing matches long before or after the match day are too random and irregular events to predict with confidence.

Given that a user's daily activity in the app may differ due to unpredictable circumstances, as aforementioned, predicting the daily activity may cause an unfair and possibly too strict model with an error score that does not reflect the reality fairly. Given the weekly seasonality of most football leagues, with most matches being played during the weekends, the variance of matches viewed from day to day will likely be high for most users, as illustrated in Figure 2.1. By aggregating the activity on a weekly basis, the prediction model will become more robust to daily

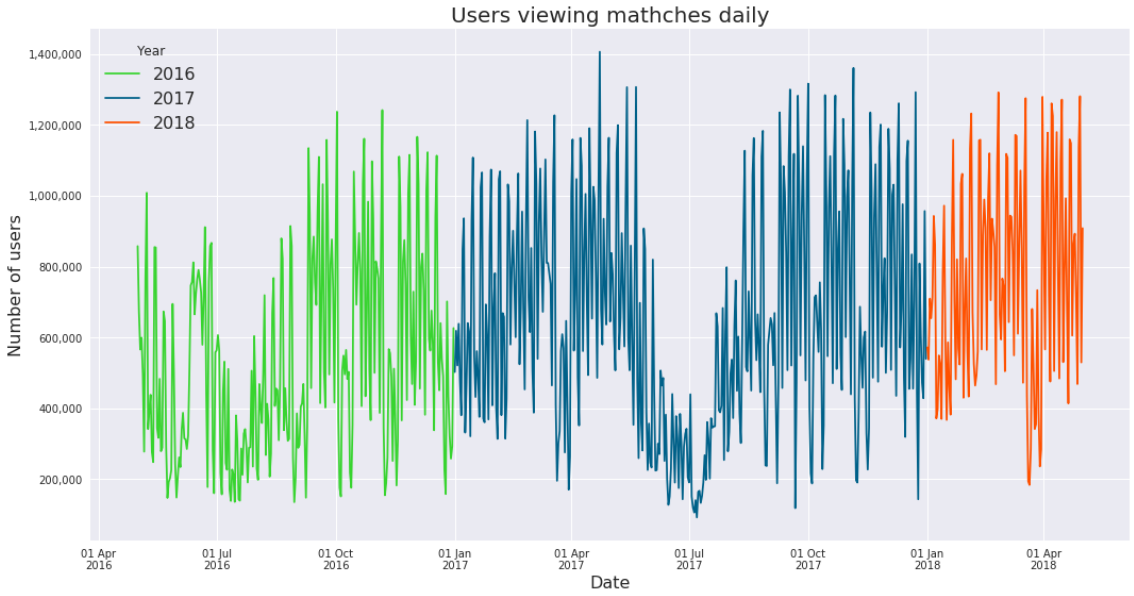


Figure 2.1: Daily amount of unique users viewing at least one match in the app during the time period 2016-05-01 to 2018-05-01.

circumstances and weekly seasonality (see Figure 2.2), generating more confident and reliable predictions. *Weekly active users* (WAU) is hence a key metric in user retention analysis for organizations working with weekly seasonality.

Several approaches on how to interpret the response variable are available. The possibly most intuitive initial approach is to treat the response variable of this thesis as binary classified response, i.e. a boolean indicating whether each unique user will view at least one match in the upcoming week or not, with the activity of the past week and metadata concerning the upcoming week as the given independent variables. Another common approach is to consider a regression model predicting the number of matches each user will be viewing the upcoming week, given the same independent variables as the previous approach. Due to the number of matches being a count variable rather than a continuous one, a Poisson regression model is preferred in order to avoid non-integer predictions. However as shown in Figure 2.3, the distribution of matches viewed daily by each user is extremely long-tailed and are therefore likely to exhibit overdispersion, i.e. produce large error variance caused by extremely active users, and pay less attention to users tending towards churning. The overdispersion could be managed by introducing a mixture of multiple unknown subpopulations of users. An excessive amount zero valued responses are also visible in Figure 2.3, which might be reduced by introducing an extreme zero inflated negative binomial model, which distinguishes between the processes giving a zero valued response or not.

A Poisson regression model predicting number of weekly viewed matches might hence be possible after further preprocessing and analysis, whereas a classification model will, in its simplicity, provide robust and reliable predictions instantly. Furthermore, a classification model will highlight users who show tendencies to approach churning, instead of users with extreme activity.

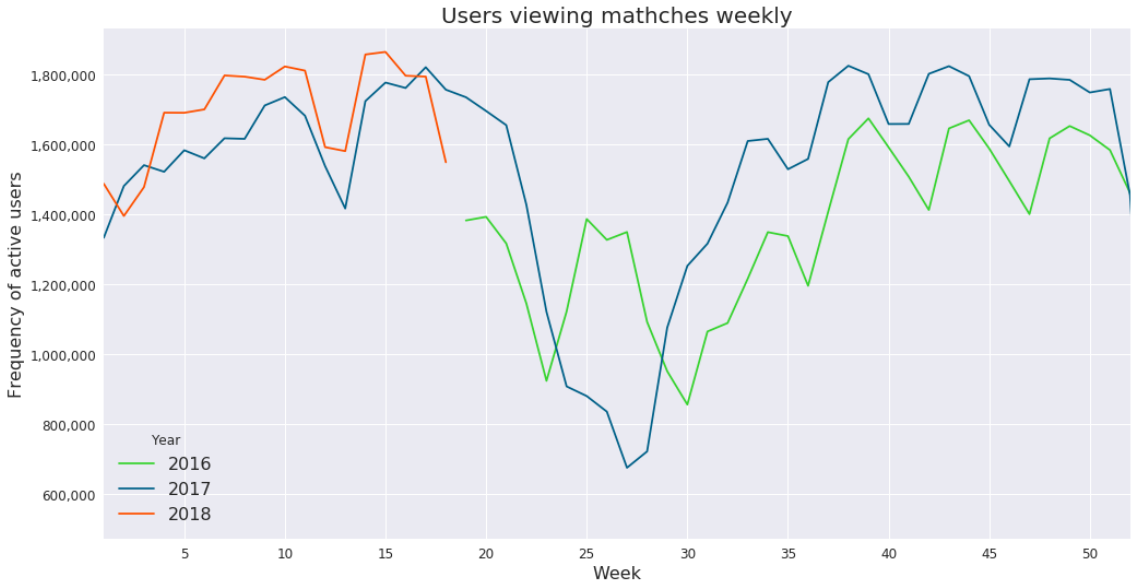


Figure 2.2: Weekly amount of unique users viewing at least one match in the app during the time period 2016-05-01 to 2018-05-01.

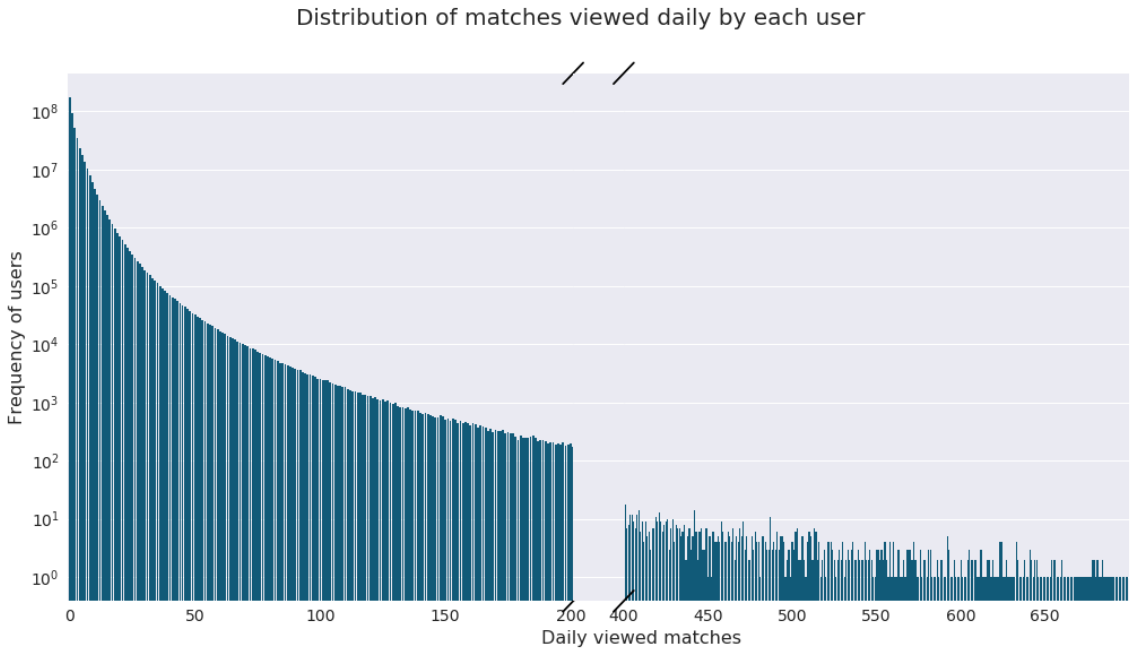


Figure 2.3: Distribution of amount of matches each unique user has viewed daily in the period 2016-05-01 to 2018-05-01. Note the logarithmic scale on the y-axis.

2.2 Explanatory variables

Due to the massive amount of possible explanatory variables available in all the various datasets of the database, the dataset constructed for this thesis only considered features relevant to the desired response variable. Most of the features are selected from a dataset comprising detailed user information, whereas a few features are additionally generated in order to achieve more information regarding match viewing observations, or to decrease the number of categories for some highly multinomial features.

2.2.1 Native features

The features selected from the unprocessed database are fully summarized in Table 2.1 and contains a mixture of static and dynamic features of various data types. The dynamic features can furthermore be partitioned as aggregating either the general app activities during the previous week, or the views generated on the different tab pages of the app during the previous week.

Static features

The only features that are completely static are `user_id`, which is a unique label for each user, and `country`, which is the only demographic feature available. Noteworthy is that the feature `country` is determined by the purchase country of the SIM card connected to the user. Due to this, the feature can not be considered as entirely accurate for every user.

Some features change so rarely they could almost be considered as static.

- `theme_changed` - number of times the user has changed the graphical design of the app.
- `num_teams_followed` - number of teams the user has selected to follow in the app.
- `push_notifications_enabled` - indicates whether the user has allowed the app to send push notifications when certain events occur or not.
- `user_type` - a segmentation of the users given factors such as how often the users is active, at which state the user visits the match pages (pre match, during match, or post match), how many matches the user views etc. The users are partitioned into six different clusters:
  - *The overlookers* - Low activity users active 6-7 days a month, and usually visit 3 unique matches per month with a slight favor towards post match viewings
  - *The addicts* - Characterized by their heavy usage of the app, being active two out of three days. The addicts visit on average 34 unique matches per month, with half of the match views being registered during live matches, whereas the remaining visits are divided equally between pre and post match views.
  - *The livefeeders* - Low activity users who visits the app approximately every fourth day. The livefeeders will typically visit 7-8 unique matches within a month, with 80% of these visits being during the matches.

Table 2.1: Summary of descriptive statistics of the features extracted from the native database. The data is aggregated over the period 2018-03-01 to 2018-05-01.

Feature name	Data type	Median	Mean	Std	Max
`user_id`	String	–	–	–	–
`date`	String	–	–	–	–
`country`	String	–	–	–	–
`theme_changed`	Integer	0	0.9626	±3.8661	1192
`num_teams_followed`	Integer	6	9.3687	±10.014	234
`push_notifications_enabled`	Boolean	–	–	–	–
`user_type`	Categorical	–	–	–	–
Activities previous week					
`days_active`	Integer	2	2.8385	±2.7588	8
`ad_click`	Integer	0	0.1769	±0.9524	581
`media_link_click`	Integer	0	1.1559	±3.8418	340
`vote`	Integer	0	0.4384	±4.0368	1697
`share`	Integer	0	0.0021	±0.0959	109
Views previous week					
`highlights`	Integer	0	1.7930	±6.4578	2098
`lineup`	Integer	0	3.5688	±13.645	4254
`match_events`	Integer	2	14.092	±48.097	7802
`match_statistics`	Integer	0	1.4846	±14.952	
`next_match_all`	Integer	0	0.1024	±3.0356	2303
`next_match_favorite`	Integer	0	0.0992	±1.6198	1022
`opinion`	Integer	0	0.0068	±0.2479	205
`planning_all`	Integer	0	0.0419	±1.5654	1504
`planning_favorite`	Integer	0	0.0616	±1.4657	2021
`recap_all`	Integer	0	0.0771	±1.8235	4453
`recap_favorite`	Integer	0	0.0754	±1.2149	875
`results_all`	Integer	0	0.8051	±16.050	7055
`results_favorite`	Integer	0	0.6817	±9.9207	5500
`team_fixtures`	Integer	0	0.5236	±4.7474	2077
`team_table_view`	Integer	0	0.1207	±1.7311	1947

– *The all scrollers* - Middle range activity users visiting the app about every second day, visiting 22-23 unique matches per month of which 60% are during the match and the rest divided equally between pre and post match visits, with a slight favor towards post.

– *The lazy* - The least active users, who visits the app approximately 6 days per month, rarely leaving the calender tab (Figure 1.1d) and never visiting the match events tab (Figure 1.1c).

– *The retro checkers* - Low activity users using the app approximately every fourth day, typically visiting 8 unique matches per month whereof 70%

are in a post match state. The remaining views are divided between during and pre match views, with a slight favor towards viewing during matches.

Activities previous week

Some of the features selected from the database are related to the total number of occurrences of certain general events in the app during the past week.

- `days_active` - Number of days the user has been active in the app during the past 7 days.
- `ad_click` - Number of ads the user has clicked on in the app flow during the past 7 days.
- `media_link_click` - Number of highlight videos the users has seen in the app flow during the past 7 days.
- `vote` - Number of times the user has voted in a question in the app during the past 7 days, e.g. player of the match, confidence for coaches, predicting winner etc.
- `share` - Number of times a user has shared a match page in the app to another app during the past 7 days.

Views previous week

Most of the selected features are related to which attributes of the app the user has been viewing, aggregated over the past week.

- `highlights` - Number of times the user has visited the tab gathering all highlights relevant to a match during the past 7 days.
- `lineup` - Number of times a user has visited the tab showing the lineups of each team of a match during the past 7 days.
- `match_events` - Number of times a user has visited the tab showing the occurred events in a match during the past 7 days.
- `match_statistics` - Number of times a user has visited the tab showing the statistics of a match during the past 7 days.
- `next_match` - Number of times a user has visited the match calendar to see which matches are to be played during the next 2 days. Aggregated over the past 7 days. One counter for all teams and a separate one for the user's favorite team are available.
- `planning` - Number of times a user has visited the match calendar to see which matches are to be played more than 2 days from the current day, aggregated over the past 7 days. One counter for all teams and a separate one for the user's favorite team are available.
- `recap` - Number of times a user has visited the match calendar to see the results of the matches played previous day, aggregated over the past 7 days. One counter for all teams and a separate one for the user's favorite team are available.
- `results` - Number of times a user has visited the match calendar to see the results of the matches played today, aggregated over the past 7 days. One

counter for all teams and a separate one for the user's favorite team are available.
- `opinion` - Number of times a user has visited a team page to see the current results on votable questions regarding the team during the past 7 days.
- `team_fixtures` - Number of times a user has viewed the fixtures on a team page during the past 7 days.
- `team_table_view` - Number of times a user has viewed the tournament tables on a team page during the past 7 days.

2.2.2 Feature engineering

The database contains a massive amount of features, both raw and aggregated. However a few additional features were generated and added to the dataset, partially as a result of preprocessing the data, but also to possibly increase the prediction accuracy by combining information from available features. The full summary of the generated features are given in Table 2.2.

Table 2.2: Summary of descriptive statistics of the features generated to improve prediction. The data is aggregated over the period 2018-03-01 to 2018-05-01.

Feature name	Data type	Median	Mean	Std	Max
`week`	Integer	–	–	–	–
`year`	Integer	–	–	–	–
`region`	Categorical	–	–	–	–
`league_of_favorite_team`	Categorical	–	–	–	–
`favorite_in_national_league`	Boolean	–	–	–	–
`nbr_followed_teams_playing`	Integer	4	6.5634	±7.9135	241
`followed_engagement`	Float	0.0591	0.1338	±0.1746	1.0
`favorite_team_playing`	Boolean	–	–	–	–
`favorite_engagement`	Float	0.0097	0.1550	±0.2383	1.196
`follow_favorite_loyalty`	Float	0.5547	0.5331	± 0.3153	1.0
`followed_tournaments_during_season`	Integer	14	12.847	±9.7365	214

Processed features

From the `date` feature, given in Table 2.1, the corresponding `week` and `year` is extracted as tools for simplifying seasonality comparisons. Since `country` is a categorical feature containing several hundred categories, one for each unique country registered among the users, a large majority of the feature space will be represented by these categories, causing sparsity. A new feature `region` is therefore introduced, merging countries together with respect to their geographical location. The new feature is categorical with the levels `Africa`, `Asia`, `Australia`, `Central America`, `Europe`, `Middle East`, `North America`, and `South America`.

The users have the option to follow the teams they are interested in, and the most followed team is labeled as a favorite team. Noteworthy is that the favorite team of each user is not selected manually by the user, but rather by the team receiving most activity for each user. The team labeled as favorite is therefore not

necessary among the teams selected by the user. An intuitive assumption is that the probability of a user viewing a match is highly correlated to which teams are attending, and whether any of the teams are labeled as followed or favorite. Among the users being considered in this thesis, 90% have selected at least one team to follow, as seen in Figure 2.4. As a user might change the number of teams to follow during the scope of the analysis, the number of teams followed is averaged for each user.

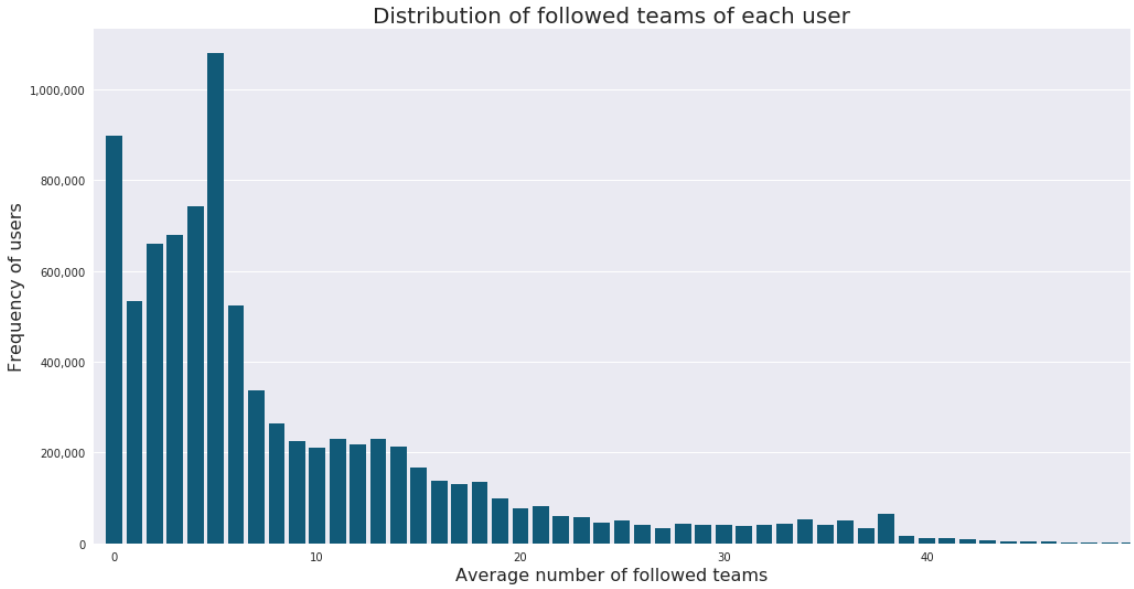


Figure 2.4: Distribution of average number of followed teams by each unique user.

Similarly to `country`, the database contains more than 15,000 teams, causing a sparse high dimensional feature space if categorized. 2 new features are therefore introduced to reduce the number of class levels significantly. The first feature is `league_of_favorite_team`, which groups the favorite team by their corresponding national league affiliation. However as the amount of available leagues in the database is still too large, this feature is constrained to only consider the five highest ranked national leagues; `Premier League`, `Serie A`, `La Liga`, `Bundesliga`, `Ligue 1`. An additional category named `Others` is included to represent the teams of the remaining leagues. The second added feature, `favorite_in_national_league`, indicates whether the league of the favorite team of each user is from the same country as the user or not. With these features, some information regarding the user's favorite team is maintained while the number of categories are significantly decreased.

Combined features

The metadata of each match is available in advance of the match date, and given the knowledge of which teams each user is more likely to follow, the accuracy of predicting match views is assumed to increase when including information regarding which teams will be playing during the time period of interest. The features `nbr_followed_teams_playing` and `favorite_team_playing` is therefore introduced in

order to indicate whether each user's teams of interest will be playing during the prediction period. However, as some supporters follow their teams more intensely than others, merely the information of the team's schedule might not be sufficient to predict their future activity in the app. Two new features, `favorite_engagement` and `followed_engagement`, are introduced to track how often the user is viewing matches involving their favorite and followed teams respectively. The engagement is defined as the ratio of matches played by the concerned team that has been viewed by the user. Conversely, loyalty is defined as the ratio of matches viewed by the user that includes either a followed team or the favorite team, i.e. does the user view football matches in general or strictly follow certain teams. The loyalty ratio is introduced through the feature `follow_favorite_loyalty`.

During a day where favorite or followed teams are playing, the total number of matches viewed by a user can be expanded using the values of these external feature values as follows

$$\texttt{\#viewed\_matches} = \frac{\texttt{foll\_eng} \cdot \texttt{foll\_playing} + \texttt{fav\_eng} \cdot \texttt{fav\_playing}}{\texttt{foll\_fav\_loyalty}}. \quad (2.1)$$

As seen in (2.1), for a user with low loyalty, the number of viewed matches might be high even though few favorite or followed team is playing. Reversely, when the engagements are low, they have small impact on the number of viewed matches.

For a user with high loyalty, the number of viewed matches is determined by how many matches involving a followed or favorite team are played, and how engaged the user is to those teams.

2.3 Constraints

On 2018-01-01 there were 17,805,220 users registered in the database, with new users downloading the app every day. However only a small partition of the users are actively using the app on a weekly basis (see Figure 2.2) causing the dataset containing all users to be highly imbalanced in favor of the non-active users. To reduce the imbalance between the classes, a constraint was implemented such that for each weekly dataset only users showing activity during the last 90 days were included. An exception to this constraint was users that had downloaded the app during the past month, as these users are too new to be considered as churned at such an early stage. Using these constraints a sufficient amount of active users were included, as illustrated in Figure 2.5, whilst reducing the amount of inactive users significantly, resulting in a reasonable size of the dataset.

Since all datasets are updated on a daily basis, new features have been introduced and updated during the lifetime of the app. To have consistency in the feature space for all considered datasets, the data stream analyzed in this thesis was limited to the time period 2016-05-01 to 2018-05-01, yielding 8,761,252 unique users being investigated.

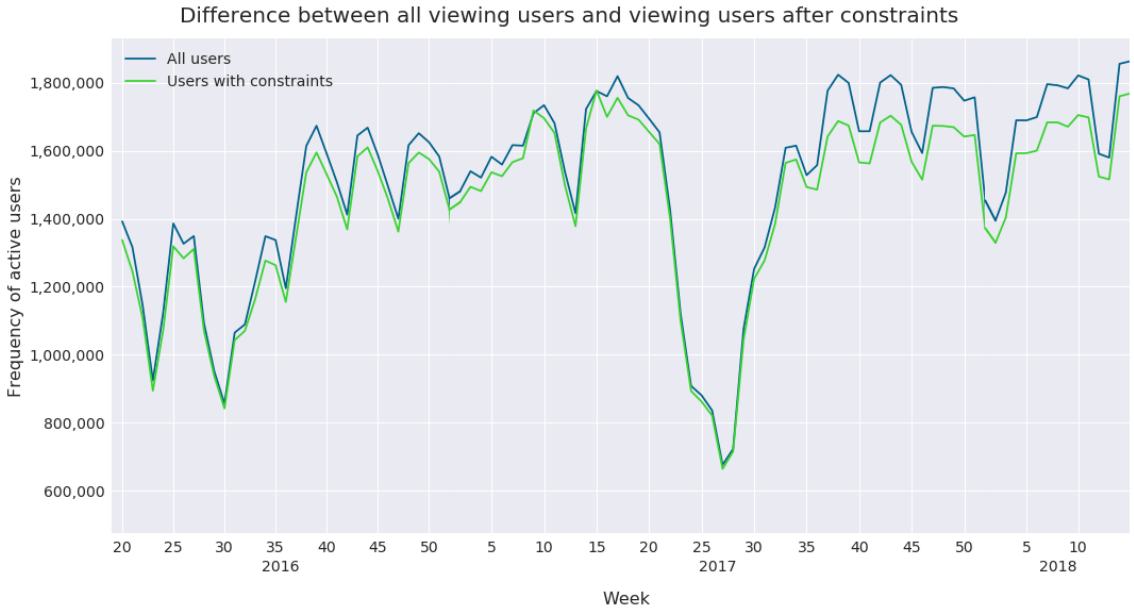


Figure 2.5: Difference between weekly amount of unique users viewing at least one match, with and without the constraints presented in Section 2.3.

2.4 Characteristics

The amount of the 8,761,252 unique users viewing at least one match for each day during the period 2016-05-01 to 2018-05-01 is illustrated in Figure 2.1, where the very drastically varying user frequency is obvious. However some seasonality patterns are visible, with the activity being consistently altering between high and low on a daily basis, with a few exceptions in certain periods of the year. These expectations become more visible when aggregating the number of unique users viewing at least one match during a week. Figure 2.2 displays the weekly amount of unique users viewing at least one match during the considered time period. It is noteworthy that the number of viewing users has increased constantly since the start of the period, and that the fluctuations are recurrent in an almost identical fashion, disregarding minor deviations. The only major inconsistency between the seasons is for the weeks 23-30, where the amount of viewing users was high in 2016 but very low in 2017. This is however not unexpected, as the popular UEFA European Championship was played during these weeks in 2016, whereas during 2017 only less popular tournaments were active during this period. If one would observe the activity during multiple season, it is likely that the even years would yield many active users during the beginning of the summer, due to the FIFA World Cups and UEFA European Championships, whereas decreased activity would be anticipated for odd years.

Due to massive amount of data obtained when considering weekly activities of 8,761,252 unique users during 2 years, a subset is considered to enable faster visualizations. The remaining analyzes presented in this section are therefore only considering observations during the time period 2018-03-01 to 2018-05-01, yielding 26,383,266 observations in total.

2.4.1 Correlations

A broad overview of the correlations between the features of the chosen dataset are illustrated in Figure 2.6. As seen in the figure, the correlations between most of the features are close to zero. However some blocks of features appear to have some correlations within each block.

Figure 2.6: Correlation matrix of all features included in the dataset. Most correlations are zero, with some exceptions appearing in blocks.

The features corresponding to activities in the app as well as the features regarding match page views show positive correlations between each other, as further illustrated in Figure A.1 in Appendix A. The positive correlations among these features are not surprising, and show that active users mostly view the tabs of the match page. Furthermore are positive correlations noticeable between push notifications, number of teams followed, and number of followed teams playing. Intuitively, a higher number of followed teams increases the possibility of a followed team playing. Similarly, if push notifications are enabled for followed teams the number of received notifications will increase. It is therefore reasonable that the correlations between these features are positive, as shown in Figure A.2a in Appendix A.

The generated features regarding engagement and loyalty towards followed and favorite teams also show strong correlations, as illustrated in Appendix A, Figure A.2b. Given the relationship between these features (2.1), the positive correlations between the engagement related features are expected, as well as the negative correlations with the loyalty feature.

Finally the lower right quadrant of the matrix illustrates some negative correlations between the categorical features, as shown in Figure A.3 in Appendix A. As the categorical features have been converted to dummy variables the correlation is expected to be zero as they are mutually exclusive events. However the negative correlation can appear when the proportions of each class population is imbalanced.

2.4.2 Distribution of numerical features

As seen seen in Table 2.1, all numerical features have some extremely high valued outliers, whereas the medians are 0 for most features, causing the distributions to be skewed and very long-tailed.

For illustration purposes, only the distribution of non-extreme observations is considered among the numerical features. The lower limit is restricted to be larger than 0 and the upper limit is restricted to a suitable percentile of the observations. The distributions containing all observations are illustrated in Appendix A.

In Figure 2.7, the 99th percentile distributions of the activity related features are illustrated. As seen in the figure, the distribution of days active during the last week is spread out with a slight decrease in users with respect to days active, whereas the user frequency for remaining features decreases more rapidly. The figure also illustrates that the number of media link clicks is more frequent over a larger interval than ad clicks and votes. Similarly to the other activity related features, the frequency of changed themes shows an exponential decay, with the decay constantly being larger compared to previous mentioned activity features. The share feature was excluded from the figure due to the 99th percentile having value 0, indicating that sharing from the app is very rare.

In Figure 2.8, the 95th percentile distribution of the match view related features are illustrated. Compared to Figure 2.7, the exponential decay of the frequency is slower, and thus the distribution more scattered, even though the percentile is lower. Match views are the essentials of the app so it is not surprising that the tail of the distributions will get wider and longer. The figure shows that match events is the most visited tab of the match view page by a large margin, followed by the lineup

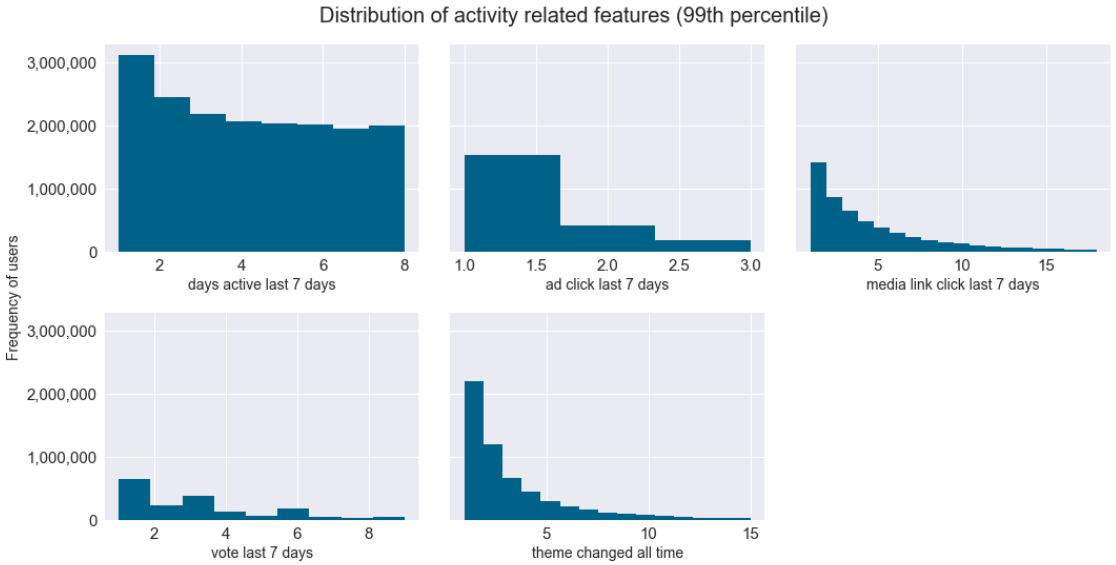


Figure 2.7: Distribution of the 99th percentile for activity related features. The observations of value 0 are excluded for illustration purposes.

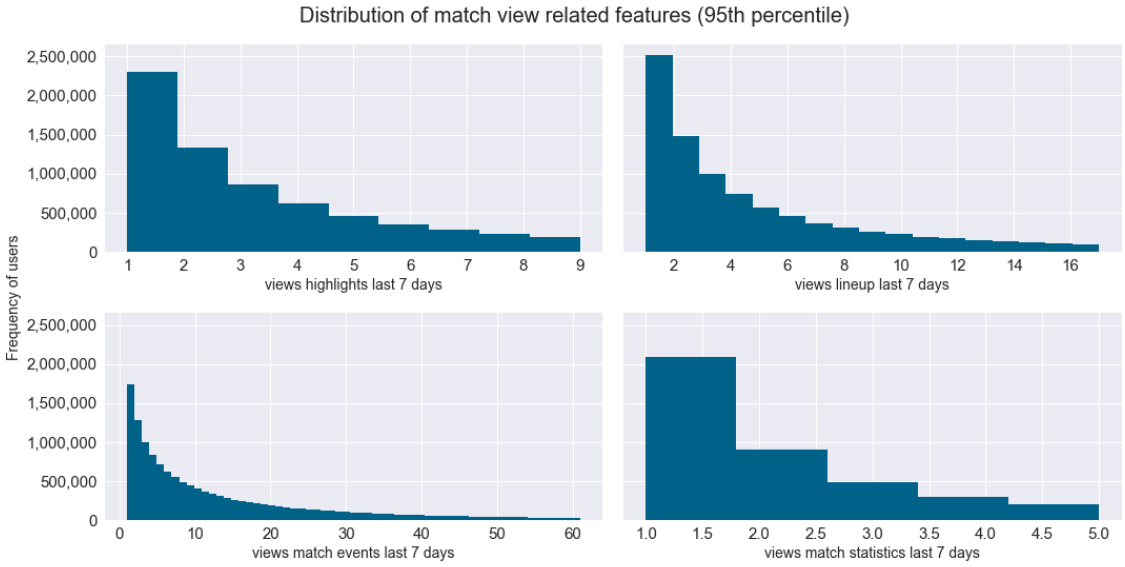


Figure 2.8: Distribution of the 95th percentile for match view related features. The observations of value 0 are excluded for illustration purposes.

tab and the highlights tab.

Figure 2.9 illustrates the 99.5th percentile distribution of the calender related features. Once again the exponential decay is obvious. It is furthermore noticeable that the distribution between features concerning all teams and the favorite team is similar for each respective feature type, with the tail being both longer and wider for favorite related features. This indicates that users more often visit the calender tab with the purpose of finding the schedule of their favorite team than teams in general, which aligns with the previous assumption of favorite team being a key feature in user behaviour.

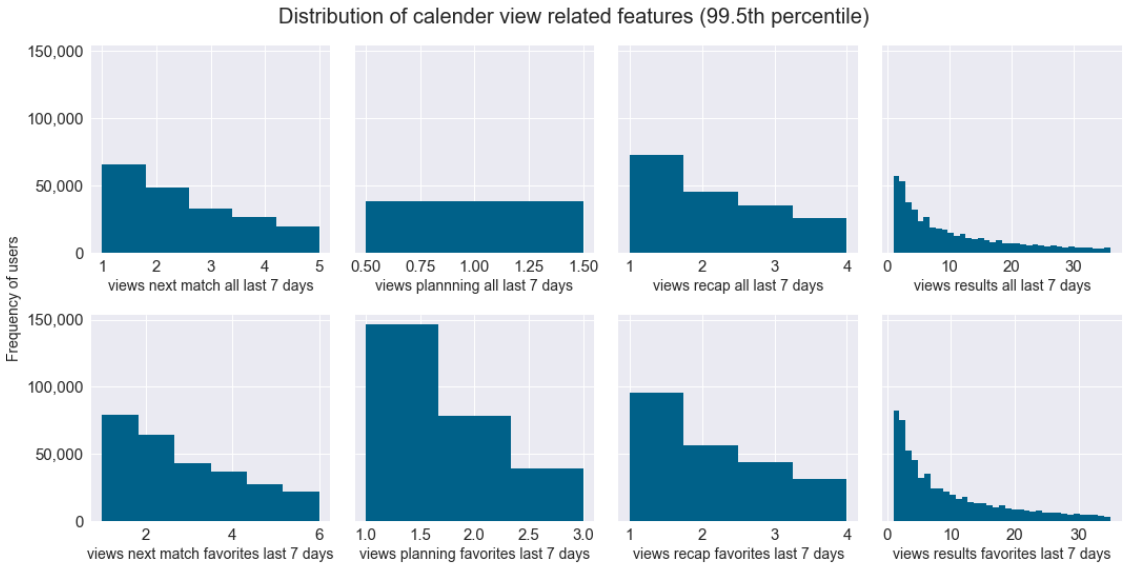


Figure 2.9: Distribution of the 99.5th percentile for calender related features. The observations of value 0 are excluded for illustration purposes.

The 99.5th percentile distribution of the team page related features is illustrated in Figure 2.10. Similar to the share feature in Figure 2.7, the opinion feature was excluded from Figure 2.10 due to the 99.5th percentile having value 0, indicating that viewing the team opinions is rare. The remaining features show an exponential decay, with the team fixture views having a higher frequency and wider tail than the team table view.

Lastly, the distribution of the engagement and loyalty generated features is illustrated in Figure 2.11. For this distribution, only the engagement users following or favoring a team are considered, i.e. missing values are ignored. As seen in the figure, the distribution of following and favorite engagement is exponentially decaying, whereas the loyalty is exponentially increasing. Given (2.1), this reversed behavior is expected. It is also noteworthy that the favorite engagement distribution is decaying less rapidly than the followed engagement distribution, indicating that users in general appears to be more engaged in viewing their one favorite team play than viewing the matches of all their followed teams.

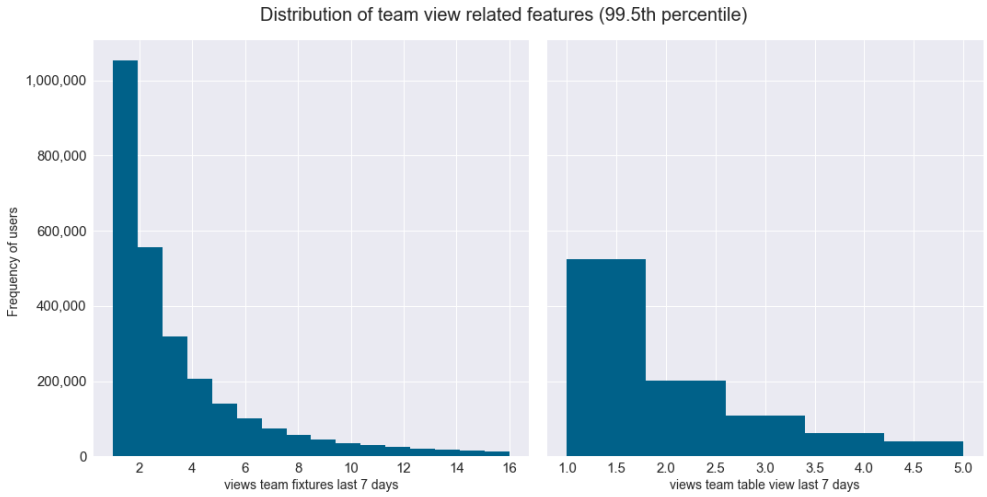


Figure 2.10: Distribution of the 99.5th percentile for team page related features. The observations of value 0 are excluded for illustration purposes.

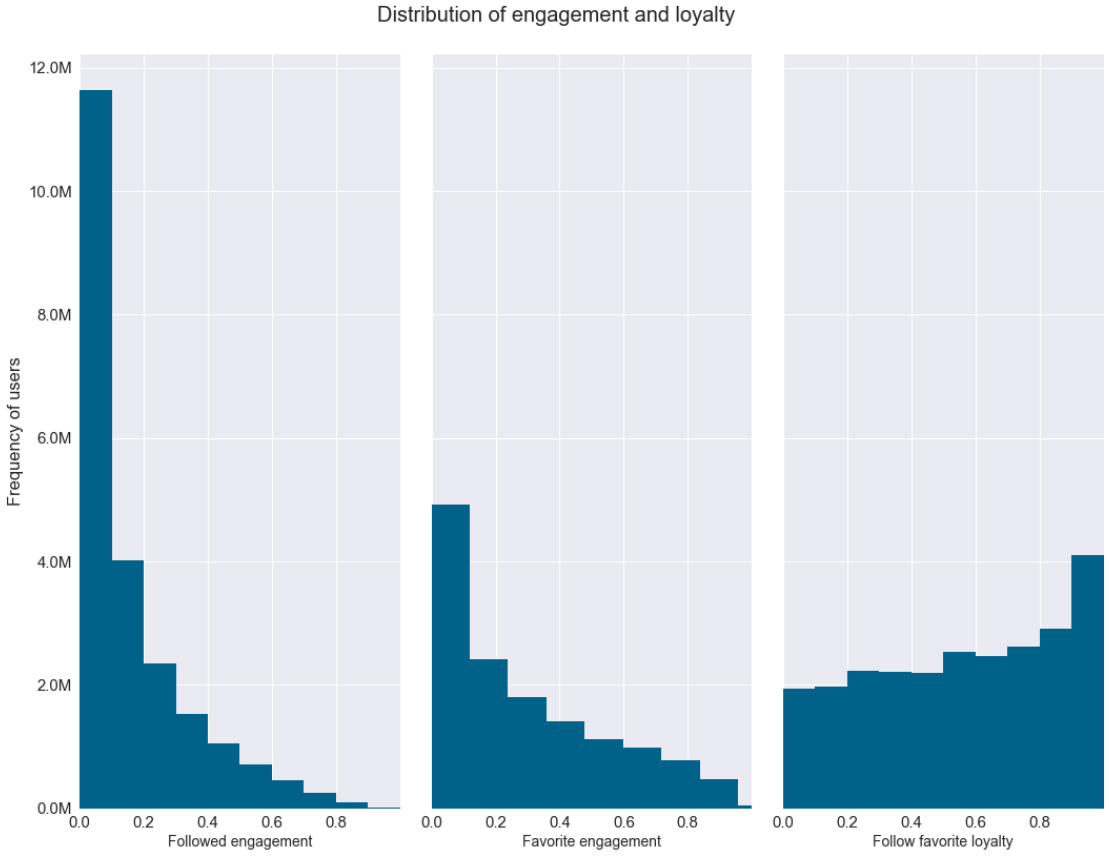


Figure 2.11: Distribution of the generated engagement and loyalty features. Only users following or favoring teams are considered.

2.4.3 Distribution of boolean features

The distribution of the boolean features of the dataset is illustrated with respect to the response variable in Figure 2.12. As seen in the figure, the majority of the users have enabled push notifications, and among these users, it is more frequent that the user will view matches in the following week. For the users with disabled notifications, the frequency of users viewing matches or not is equal, indicating that having push notifications enabled affect the response variable positively. Furthermore, Figure 2.12 shows that users with a favorite team in the national league have a tendency of viewing matches, whereas the frequency is even among users without the favorite team in the national league (including both users with favorite in other leagues and no favorite at all). Figure 2.12 also illustrates that the frequency of users viewing matches when their favorite team is playing is significantly larger than those not viewing when their favorite team is playing, and reversely the frequency of viewing users is lower than not viewing users when their favorite team is not playing. This observation supports the assumption of users being more active when favored teams are playing.

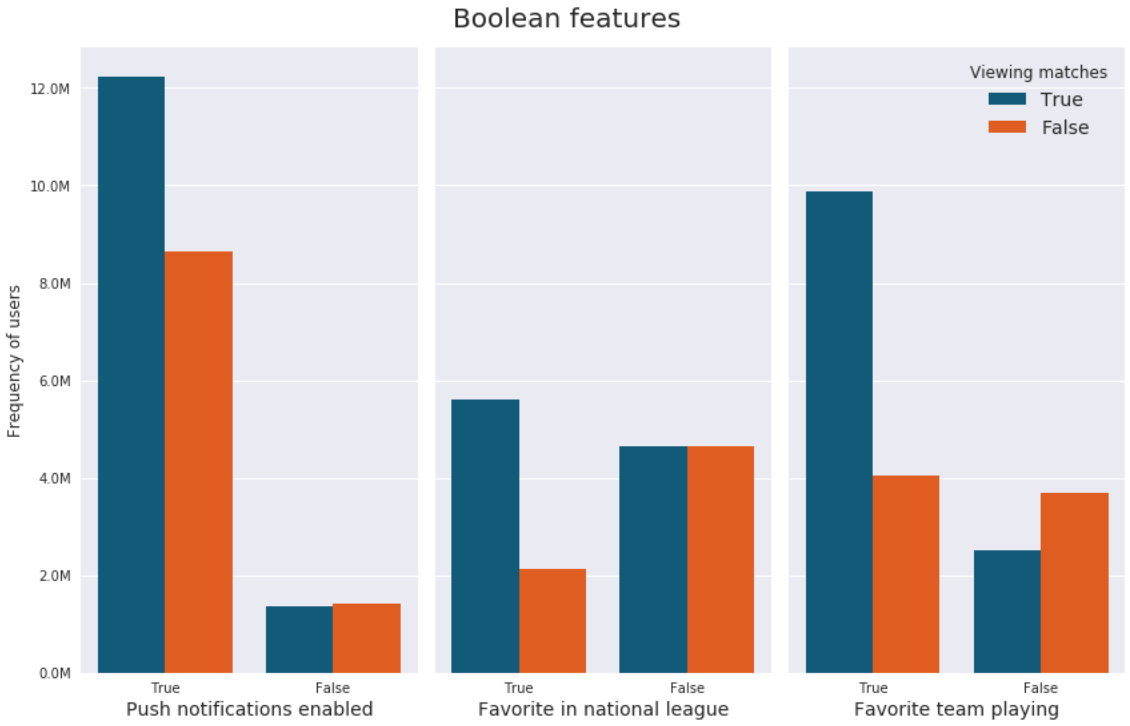


Figure 2.12: Frequency of the boolean features and their corresponding distributions of the response variable.

2.4.4 Distribution of categorical features

Three categorical features are included in the dataset, as described in Table 2.1 and 2.2. The distribution of each category within these features are illustrated with respect to the response variable in Figures 2.13, 2.14, and 2.15. Figure 2.13

shows the distribution of each user type and their respective frequency of viewing matches. As seen in the figure, the addicts is the largest group, and also the most active one. The livefeeders, all scrollers, and retro checkers have a slight favor in viewing matches than not viewing, whereas the lazy users rarely view any matches. The overlookers are also in favor of not viewing matches, but are also a very small cluster of users. The users without a type segmentation barely view any games, which is expected given that the user clustering is based on the activity patterns within the app, and the rarely active users have thus not been assigned to a cluster yet.

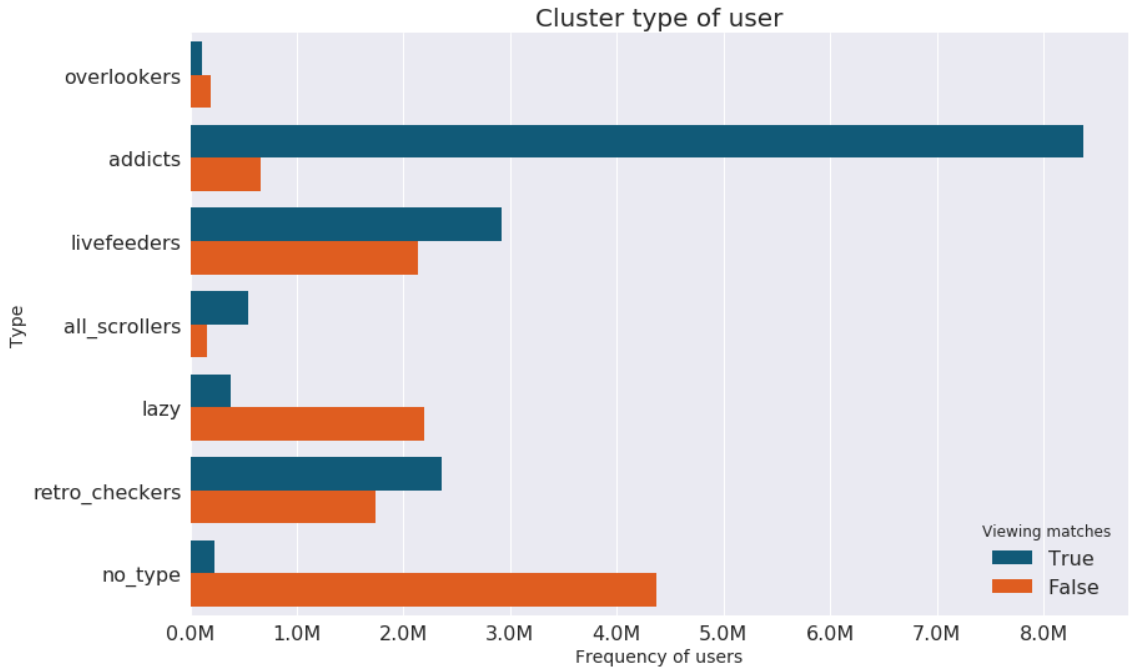


Figure 2.13: Frequency of the user types and their corresponding distributions of the response variable.

The distribution of the region feature is illustrated in Figure 2.14. The figure primarily shows that the vast majority of the users are Europeans. Secondly, the frequency within each region appears to be evenly distributed among viewing and not viewing users, with the exceptions of Europe, where the majority of users are viewing matches, and possibly Africa, where the frequency of non-viewing users is visibly larger.

The final categorical feature is the league affiliation of the user's favorite team, for which the distribution is illustrated in Figure 2.15. The figure shows explicitly that a user with a favorite team, regardless of league affiliation, is significantly more likely to view matches than a user without a favorite team. Furthermore it is noticeable that the British Premier League, Italian Serie A, and Spanish La Liga are the leagues containing the most favored teams.

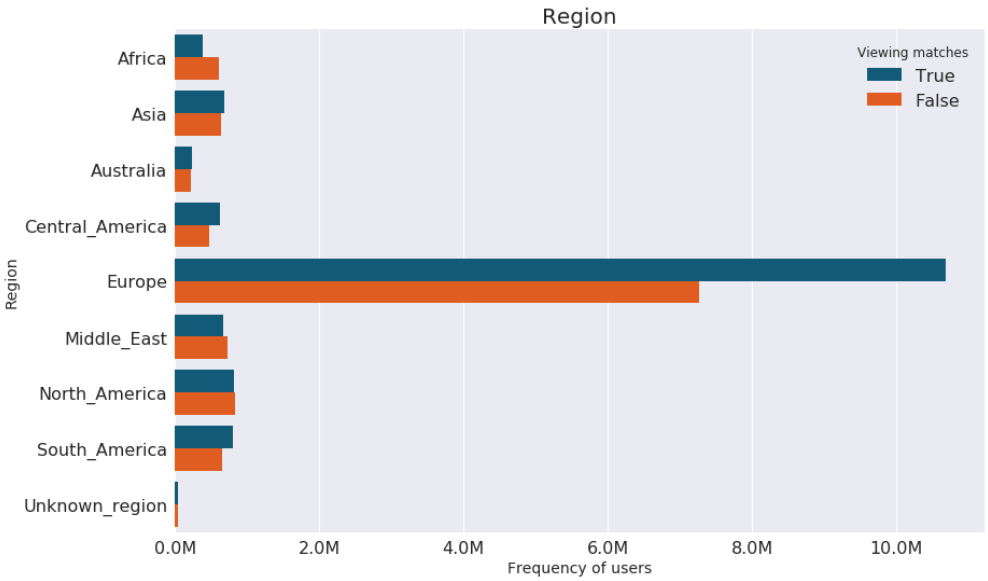


Figure 2.14: Frequency of the regions and their corresponding distributions of the response variable.

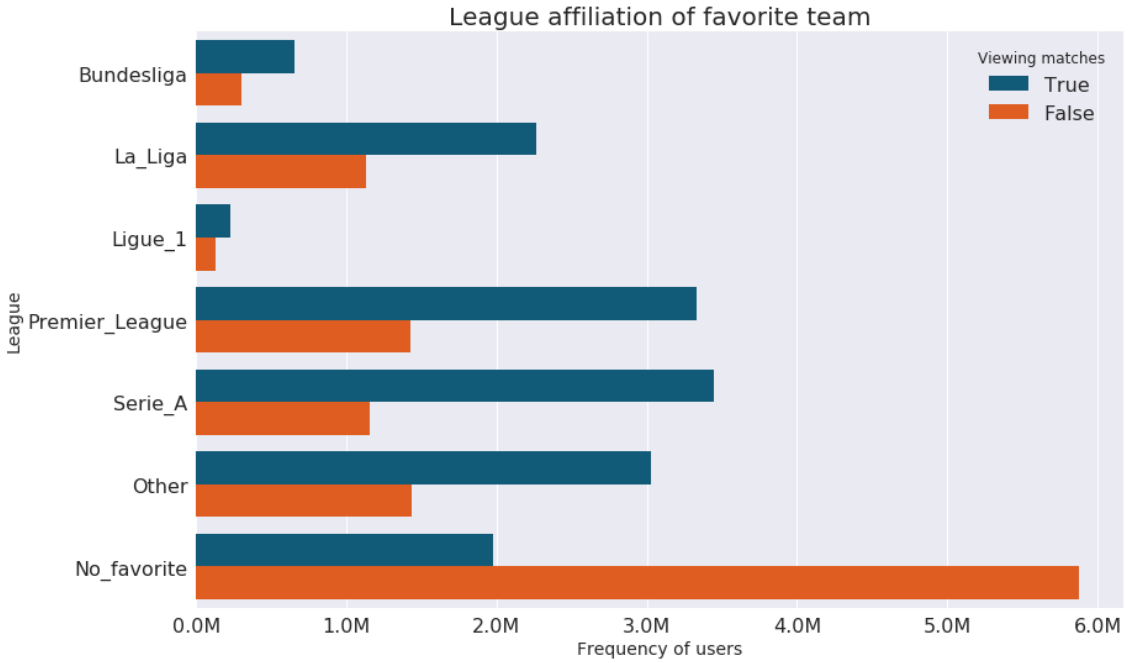


Figure 2.15: Frequency of the league affiliations and their corresponding distributions of the response variable.

3
Theory

Supervised learning is the task of learning a function f to map a set of input features $\mathbf{x}$ into an output measurement y such that $f(\mathbf{x}_i) = y_i$ for each observation $i = 1, \ldots, N$, where N is the number of observations available in the dataset. The outcome y is usually either quantitative (e.g. stock price), and then referred to as a *regression* problem, or categorical (e.g. heart attack/no heart attack), which is a *classification* problem. The function f, also called the prediction model, is trained using previously seen pairs of input measurements and output responses, and the objective is to enable the function to predict the outcome for new unseen observations. The set of observed pairs is called a *training set*, and the problem is considered to be "supervised" since the learning process is guided by the presence of previously observed outcomes.

In this chapter, supervised learning methods for classification used during the work of the thesis are briefly presented. For an extensive reading of the fundamental theory of machine learning and prediction models in general, see [14].

3.1 CART

Decision trees are tree-based methods which partition the feature space into a set of rectangles, and fit a simple model in each one of these regions. A popular decision tree method are Classification And Regression Trees (CART), which will be considered during this thesis

The feature space is initially split into two regions, with the split-point and variable to split on being chosen to achieve the best fit, defined by minimizing a cost function, and the response Y being modelled by the majority vote of Y in each region. The best candidate, i.e. the one reducing the cost function the most, is then split into two new regions. The partition process is recursively iterated until a stopping rule is applied. The stopping rule can for example be reaching a maximum number of splits, number of observations comprised in a region being above a fixed value, or the error rate improvement of adding a new branch being below a certain threshold. A toy illustration of the splitting process and the corresponding CART is provided in Figure 3.1. The result of the process will be a partition of M regions $R_1, \ldots, R_M$, and the corresponding classification model predicts Y using a constant c_m in each region R_m, such that

$$f(\mathbf{x}) = \sum_{m=1}^{M} c_m \mathbf{1}\left\{x \in R_m\right\},$$

where $\mathbf{1}\{x \in R_m\}$ is the indicator function. For a classification problem with K possible outcomes, the constant c_m can be estimated using the class proportion within each region as

$$\hat{c}_m = \arg\max_k \hat{p}_{mk}$$

where the proportion of class k in region R_m is given by

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbf{1}\{y_i = k\}.$$

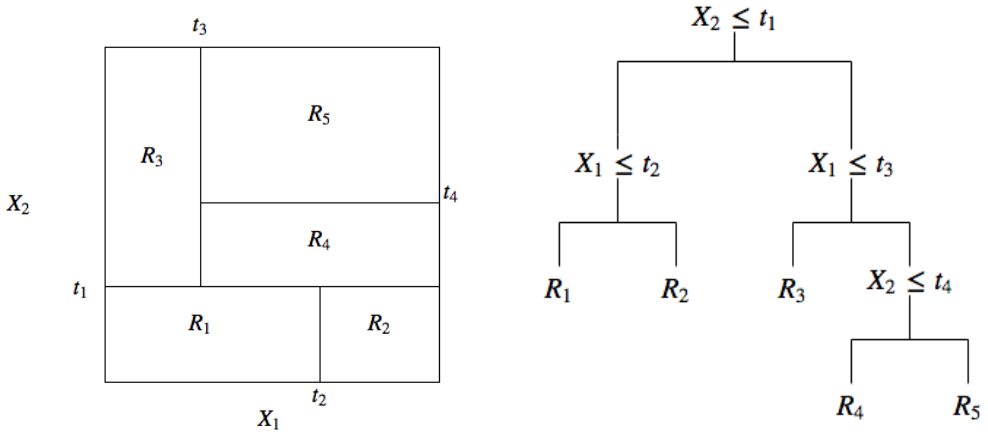


Figure 3.1: Toy example of a recursive binary partitioning of a two-dimensional feature space (left) and the corresponding CART (right).

The objective when splitting the data into two parts is to make each node as pure as possible in terms of class labels. Common impurity criteria are the misclassification error rate, given as

$$\epsilon_m = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbf{1}\{y_i \neq \hat{c}_m\} = 1 - \hat{p}_{m\hat{c}_m},$$

and the Gini index, given by

$$GI_m = \sum_{k \neq k'} \hat{p}_{mk}\hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}).$$

The Gini index is adjusted to minimize the variance within each region, and the more mixed a region R_m is, the larger the Gini index. In general, using the Gini index as splitting criterion is a very aggressive approach to form pure, single class regions quickly in the process.

One advantage of CART compared to many other supervised learning methods is that it is a nonlinear classifier, i.e. not based on a linear combination of the features, and therefore more adaptable to complex dependencies in the underlying data. They are also able to handle both numerical and categorical data at once, and are, among

the most known learning methods, the one being closest to qualify as "off-the-shelf" [14], i.e. not requiring time consuming tuning of the learning procedure.

The major disadvantage of CART however is that they are prone to overfitting, i.e. creating over-complex trees with low bias but high variance that are too specific with respect to the training data and not general enough to represent a greater population. They are therefore notoriously unstable, with small variations in the data substantially changing the appearance of the tree. For further reading regarding CART, see [14].

3.1.1 Random Forest

A popular technique for reducing the high variance of CART is to use an ensemble learning method known as Random Forest [15]. By applying bootstrap aggregation, abbreviated bagging, the many noisy but approximately unbiased CART are averaged into a more robust ensemble of learners. The procedure of bagging is utilized by repeatedly selecting a random sample with replacement from the training set and, for each sample, fit a tree f_b, where $b = 1, \ldots, B$ is the predefined number of trees to be grown. The resampling of data creates partially overlapping subsamples, yielding correlations between the individual learners f_b. When the training of the sample trees is completed, predictions of unseen samples $\mathbf{x}$ is made by averaging the predictions from all the separate regression trees, i.e.

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(\mathbf{x}),$$

or taking the majority vote of classification trees. Since each of the B trees are identically distributed (i.d.), the expectation of $\hat{f}$ is the same as for each individual f_b, and the bias of bagged trees will not be affected.

The average of B independent identically distributed (i.i.d) random variables, each with variance σ^2, will have variance $\frac{1}{B}\sigma^2$. However if the variables are simply i.d, i.e. not necessarily independent, with positive pairwise correlation ρ, then the variance of the average will be given as

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2. \tag{3.1}$$

As the number of trees B included in the forest increases, the second term will tend to zero, whereas the first remains and will limit the benefit of averaging by the size of the correlation of pairs of bagged trees.

The difference between the original bagging algorithm for trees and Random Forest is that before each split in the learning process, $m \leq p$ of the p given features are randomly selected as candidates for splitting. Typical values for m is $\sqrt{p}$ for classification and $p/3$ for regression (see [14]). This process, referred to as *feature bagging*, reduces the correlation between any pair of trees in the ensemble, and furthermore by (3.1) reduces the variance of $\hat{f}$.

One additional feature of Random Forests is the *out-of-bag* (OOB) error which estimates the mean prediction error on each training sample $\mathbf{x}_i$ using only the trees that did not include $\mathbf{x}_i$ in their bootstrap sample. The OOB error yields an almost

identical estimate as the one obtained with N-fold cross-validation, and Random Forests can hence, unlike many other nonlinear estimators, be fit in a single sequence, with cross-validation being performed along the way. For further details on Random Forest, the reader is referred to [15]

3.1.2 Gradient boosting

An alternative ensemble learning method to stabilize the variance of CART is to utilize a boosting approach. Similarly to bagging, boosting selects a random sample with replacement and fits a tree for each sample. In contrast to bagging however, boosting is based on various weak learners, i.e. classifiers that is only slightly better than random guessing, which are combined into a strong learner. For bagging each observation therefore has the same probability to appear in a new dataset, whereas for boosting the observations of input data are weighted and will be included in the new sets more often. As a consequence the boosting algorithms build the learner in sequential manner, in difference to the parallel training of bagging, to take account for the previous classifiers misclassifications. The weights for each observation are redistributed in order to emphasize errors and force the subsequent learners to focus more on them. Boosting also allocates a set of weights for each classifier, creating a weighted average of their estimates.

Considering a given dataset with N observations and p features $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$, $(|\mathcal{D}| = N, \mathbf{x}_i \in \mathbb{R}^p, y_i \in \mathbb{R})$, the prediction of a general ensemble of K additive functions is given by

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F}, \tag{3.2}$$

where $\mathcal{F}$ is a family of weak learners. Any family, including linear regressors and neural networks, can be used but in the scope of this thesis only classification trees are of interest. Each f_k will thus correspond to an independent tree structure q with leaf weights w such that the space of classification trees is given as $\mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\}(q : \mathbb{R}^p \to T, w \in \mathbb{R}^T)$ where T is the number of leaves in the tree.

The final prediction will be calculated by summing up the score in the leaves, classified by the decision rules given in the trees. The set of functions in the model will be learned by minimizing the regularized objective function

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k), \tag{3.3}$$

where l is a differentiable convex loss function that measures the error of the prediction $\hat{y}_i$ in (3.2), and

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|_2^2$$

penalizes the complexity of the model, with λ being a L_2 regularization coefficient penalizing the average weights and $\gamma \in [0, \infty)$ a penalization on the tree size.

As the tree ensemble model in (3.3) includes functions as parameters, traditional optimization methods in Euclidean space cannot be applied, and the model is trained

additively. Let $\hat{y}_i^{(t)}$ be the prediction of instance i at iteration t, and add f_t to minimize the objective function

$$\mathcal{L}^{(t)} = \sum_{i=1}^{N} l\Big(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)\Big) + \Omega(f_t).$$

This is a gready way to add the tree f_t that improves (3.3) the most while maintaining the complexity. Since the loss function l is differentiable, the second-order approximation is given by Taylor expansion of $l\Big(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)\Big)$ around $f_t(\mathbf{x}_i)$, such that

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{N} \Big[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \Big] + \Omega(f_t),$$

where

$$g_i = \frac{\partial l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}},$$
$$h_i = \frac{\partial^2 l(y_i, \hat{y}^{(t-1)})}{\partial \left(\hat{y}^{(t-1)}\right)^2},$$

are the first and second order gradient statistics on the loss function l. Since $l(y_i, \hat{y}_i^{(t-1)})$ is constant in the objective, it is redundant for optimization and hence removed from the objective. By further defining the instance set of leaf j as $I_j = \{i \mid q(\mathbf{x}_i) = j\}$ and expanding the penalization term Ω, the final regularized objective function for the boosted model is given as

$$\begin{aligned} \tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{N} \Big[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \Big] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} \left[\Big(\sum_{i \in I_j} g_i \Big) w_j + \frac{1}{2} \Big(\sum_{i \in I_j} h_i + \lambda \Big) w_j^2 \right] + \gamma T. \end{aligned} \tag{3.4}$$

For a fixed tree structure $q(\mathbf{x})$, the optimal weight w_j^* of leaf j will hence be given by

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda},$$

and the corresponding optimal value of (3.4) is

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{t=1}^{T} \frac{\left(\sum_{i \in I_j} g_i \right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \tag{3.5}$$

The quality of a tree structure q can be measured using (3.5) as a scoring function, similar to the impurity score for CART. Since it is impossible to enumerate all possible tree structures, a gready approach starting from a single leaf and iteratively

adding branches is often implemented. Let I_L and I_R be the instance sets of the left and right nodes created by a split, and define $I = I_L \cup I_R$. The loss reduction after the split will then be given as

$$\mathcal{L}_{\text{split}} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} + \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma. \tag{3.6}$$

Finding the best fit of (3.6) is a key issue of tree learning, and to do so the *exact greedy boosting algorithm* (XGBoost) was implemented in [16]. XGBoost is a state-of-the-art algorithm for scalable end-to-end gradient tree boosting, which enumerates a split finding algorithm over all possible splits on all features. As this is computationally demanding for continuous features, XGBoost sorts the observations according to feature value and visit the observations in sorted order to accumulate the gradient statistics for the structure score in (3.6).

3.2 Scaling for large data

The rapid digital evolution during the last decades has caused an explosion in data storage capabilities [17], [18], and with improved tracking techniques and phenomena such as internet of things, many things can be measured and analyzed digitally. As a consequence, the digital revolution has also provided possibilities to store and organize large scale data in an accessible way.

The most concerning issues emerging when the sample sizes become large are storage limitations and lack of computational efficiency. These limitations can be overlooked by introducing high-performing supercomputers or allocating the data storage and executions to remote shell protocols, but such resources might not always be available. For this thesis however, a simple MacBook Pro running macOS High Sierra with a 2.3 GHz Intel Core i5 CPU with 8 GB main memory is used.

The main approaches when analyzing massive samples locally include subsampling, which provides estimates for each subsample and aggregates by means, divide and conquer, which analytically determines how parameter estimates from each split are combined to obtain a sufficient estimate, and Online updating, which divides the data and treats the subsamples as sequential arrivals.

3.2.1 Subsampling

Bag of Little Bootstraps (BLB) (see [19]) is a fairly robust and scalable state-of-the-art subsampling technique, which assesses the quality statistics of estimators. BLB is an extension of the m out of n bootstrap (see [20]), which generally lacks robustness in hyperparameter specification, as well as often requiring prior information of the data. The procedure of BLB is to randomly select a subset of size m from a dataset of size n. Bootstrap is performed on each subset by creating weighted resamples with the size n. The estimator is then calculated on the resamples in the same way as bootstrap. Similarly to m out of n bootstrap, the computational time is reduced compared to ordinary bootstrap due to the smaller subsample sizes. However m out of n bootstrap usually require a rescaling of the output due to the difference

between sample size and resample size. Since evaluating the precision of an estimator method usually requires a known convergence rate of the estimator, this approach is less user-friendly. The rescaling issue is however negligible for BLB as the resamples are of the same size as the data. Furthermore, the resamples contain only a small amount of distinct points from the subset, despite having the same size as the full dataset. This feature reduces the computational cost of calculating a statistical function of the resamples. The estimates of precision obtained from a few subset can thus be averaged to achieve the BLB estimate of the precision.

A slightly different interpretation of subsampling is to use one reduced sample as proxy, instead of multiple subsamples. In [21], Ma and Sun presented a state-of-the-art approach for utilizing this subsampling approach by using the emerging family of statistical methods called leveraging, which captures how far an observation is from the center of data. Recalling regression theory, the predicted response vector of an predictor matrix $\mathbf{x}$ is given by

$$\hat{y} = \mathbf{x}(\mathbf{x}^\top \mathbf{x})^{-1}\mathbf{x}^\top y = Hy,$$

where $H = \mathbf{x}(\mathbf{x}^\top \mathbf{x})^{-1}\mathbf{x}^\top$ is referred to as the hat matrix, and the i'th diagonal element of the hat matrix, h_{ii} is defined as the leverage score of observation i. For a more thorough recap on regression theory the reader is referred to [14].

An unbiased estimate of the true coefficients can be obtained with a smaller variance by randomly sampling $r \ll n$ observations with replacement. The sampling probabilities of each sample is constructed as $\pi_i = h_{ii}/\sum_j h_{jj}$, and the subsample is solved either as a weighted least squares problem (WLS) or an ordinary least squares problem (OLS). As WLS can be sensitive to smaller values of π_i and thus increasing the variance, the authors of [21] propose a shrinkage leveraging method which regularizes the sampling probabilities such that

$$\pi_i = \lambda \frac{h_{ii}}{\sum_j h_{jj}} + (1-\lambda)\frac{1}{n}$$

for a tuning parameter $\lambda \in [0, 1]$.

3.2.2 Divide and conquer

The design paradigm of divide and conquer is based on recursively decomposing a problem into multiple minor problems of a similar character and subsequently combining the solution of each subproblem to solve the original problem.

In [22], Lin and Xi proposed a computation and storage efficient strategy for estimating the parameters of statistical models in massive datasets. Considering a general estimation problem, the maximum log-likelihood estimate is defined as $\ell(\theta\,; x) = \ln \mathcal{L}(\theta\,; x)$ for a likelihood function $\mathcal{L}(\theta\,; x)$. The objective is then to solve a score equation

$$\sum_i \Psi(y_i, \theta) = \frac{\partial \log f_\theta(y_i)}{\partial \theta} = 0. \tag{3.7}$$

For a massive dataset, the dataset is partitioned into K chunks, and for each chunk the solution to the score equation (3.7) is given as

$$M_k(\theta) = \sum_{i \in k} \Psi(y_i, \theta) = 0.$$

The estimate based on the k'th subset is denoted as $\hat{\theta}_{n,k}$ and the aggregation of the estimate is defined as

$$A_k = -\sum_{i \in k} \frac{\partial \Psi(y_i, \hat{\theta}_{n,k})}{\partial \theta}, \tag{3.8}$$

for which the linearization of the scoring equation is approximated as

$$M_k(\theta) \simeq A_k(\theta - \hat{\theta}_{n,k}).$$

The approximate solution to the global scoring equation will thus be given as

$$\sum_k M_k(\theta) = \sum_k A_k(\theta - \hat{\theta}_{n,k}) = 0$$

which is solved by

$$\hat{\theta} = \left(\sum_k A_k \right)^{-1} \left(\sum_k A_k \hat{\theta}_{n,k} \right). \tag{3.9}$$

The expected value of A_k, given in (3.8), is known as the *information matrix*, and its inverse is the asymptotic variance of the maximum likelihood estimate. The solution given in (3.9) is hence a kind of weighted average of estimates, with the weights being inversely proportional to the estimation variance.

3.2.3 Online learning

When new observations arrives sequentially to the dataset in form of data streams, the traditional batch methods designed for static datasets will not be sufficient. Due to fast growing amounts of data and limited computational resources, they are not capable of analyzing efficiently enough. The prediction model must also consider concept drift, i.e. distributional changes in the data stream over time. For a traditional batch model, the performance might drastically deteriorate for unexpected events, but also seasonal events expected over a period of time. For some applications the arrival of the data might be so rapid that labelling of items might be delayed and consequently inaccurate. To adapt the speed, size and evolving nature of data streams, several classification algorithms, such as specialized sliding windows, sampling methods, drift detectors, and adaptive ensembles, have been implemented.

Ensemble learning methods are certainly attractive for classifying data streams, as they facilitate adaption to the alterations in a stream without discarding the historical knowledge. Especially for a stream with distinct seasonal patterns, the classifiers that perform poorly in the current period might be significant again when the cycle restarts. An ensemble learner can hence be adapted to a data stream by

replacing some of the outdated classifiers with classifiers trained on the most recent data, in order to both preserve the past and pursue the current. Several supervised ensemble learning approaches for data stream classification have been studied (see [23]), and a suitable method is based on the characteristics of the data stream.

Observations in a data stream can be provided either *online*, i.e. instance by instance in consecutive moments in time, or as *data chunks*, where larger sets of data are analyzed block by block. The model of the stream can either be *stationary*, where the observations are drawn from a fixed probability distribution, or *non-stationary*, where classes and attribute distributions can change over time, i.e. affected by concept drift.

These drifts can appear in various forms (see [24]), such as incremental (a sequence of small non-critical changes), gradual (transition phase where sampling slowly surpasses from one probability distribution to another), sudden (abrupt change of source distribution between time steps t and $t+1$), and recurring (concepts reoccur in a cyclic fashion), and therefore demand different approaches of detection and adaption. The model can generally tackle concept drift in an *active* or *passive* way, i.e. detecting concept drift and trigger adaptations in the classifiers or simply ignore them.

One of the earliest approaches of using ensemble methods in an online environment is the *Streaming Ensemble Algorithm* (SEA) [25], which creates a new classifier for each new instance of data. The quality of the new classifier is evaluated with respect to the next incoming instance, and subsequently replaces a learner in the ensemble that performed worse quality on that training chunk. The approach is able to avoid overfitting and maintain diversity, due to favoring classifiers which can correctly classify nearly undecidable examples, and is shown to recover faster to concept drift than an individual learner. A potential concern with SEA however, is the risk of adapting slowly to new concepts, due to new classifiers being potentially outweighed by the older ones.

Another online ensemble method dealing with concept drift is the *Dynamic Weighted Majority* (DWM) [26] which maintains a weighted set of m base learners E that can be added and removed based on the performance of the global algorithm. The weights $w_1, \ldots, w_m$ of a learner are reduced with a multiplicative factor β when performing poorly, and eventually removed from the ensemble if the weights reach a sufficiently low weight θ. New learners are added whenever the global ensemble performs poorly, with the learners being evaluated every p iteration. As the algorithm is based on ensembles, any online learning algorithm, as well as different combinations, can be implemented as base learner.

A third framework worth mentioning when dealing with concept drift is the *Batch Weighted Ensemble* (BWE) [27], which incorporates a drift detector to an evolving ensemble of classifiers with the intention of adapting to both sudden and gradual drift. The drift detector accumulates the classification accuracy for each learner and builds a regression model based on the previous accumulated values, which estimates the tendency in the data. The detector is subsequently embedded into the BWE where, if drift is signaled by the detector, the ensemble will add a new classifier built on the batch, and establish weights for each classifier. All classifiers with weight 0 are deleted, and if the entire ensemble is deleted, a new classifier is created based

on the last received batch.

To handle reappearing concepts in an effective way, the *Evolutionary-Adapted Ensemble* (EAE) [28] collects all elementary classifiers trained on each instance of the stream in an unlimited pool. The classifiers in the pool are never removed, such that knowledge of past contexts is preserved for future use, in contrast to most ensemble solutions which discard poor performing learners. The pool is only updated when concept drift is detected in the stream, and the number of learners contributing to the final decision is fixed and limited, in order to ensure high performance. The elements from the pool joining the decision-making ensemble are selected by an evolutionary-based optimization algorithm aiming to minimize misclassification rate.

So far, this thesis has solely considered CART as a decision tree learner. However multiple other methods have been developed, and in terms of data streaming Hoeffding trees [29] is of certain interest. Whereas CART assumes all training examples can be stored simultaneously in main memory, and therefore are limited in number of examples they can learn from, Hoeffding trees are designed to read each example at most once and thereby ignoring the limit of CART. This design claims it is sufficient to only consider a small subset of training examples when finding the best attribute to test at a given node. For a stream of examples, the initial examples will thus be used to choose the root attribute, whereas the succeeding examples are recursively passed down to the corresponding leaves and used to determine the appropriate attributes there. Noteworthy is that the examples are assumed to be generated by a stationary stochastic process.

The exact number of necessary examples at each node is obtained by using the statistical property *Hoeffding bound*. Considering a real-valued random variable r with range R, and supposing n independent observations are given of this variables, the Hoeffding bound states that with probability $1-\delta$, the true mean of the variable is at least $\bar{r} - \epsilon$, where

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta}{2n}}.$$

The attractive property of the Hoeffding bound is that it is independent of the probability distribution which generates the observations. Due to this, the bound is more conservative than distribution-dependent bounds. Considering $G(X_i)$ be the heuristic measure of choosing test attributes (e.g. information gain or Gini index), the objective is to ensure that the attribute chosen using n examples is the same as would be chosen after infinite examples. Assuming that G is to be maximized and letting X_a be the feature with highest observed $\bar{G}$, followed by X_b as the second best feature, the difference between the heuristic values is defined as $\Delta\bar{G} = \bar{G}(X_a) - \bar{G}(X_b)$. The Hoeffding bound then guarantees that, given a desired δ, the true $\Delta G \geq \Delta\bar{G} - \epsilon > 0$ with probability $1 - \delta$ and that X_a therefore is indeed the best attribute at the given node with probability $1 - \delta$. The given node can therefore be split using the current best attribute, and succeeding examples are passed to new learners.

Based on the Hoeffding tree architecture, a high-performance data mining system named Very Fast Decision Trees (VFDT) was implemented. VFDT provides several refinements to the Hoeffding tree algorithm such as reduced cost for recomputing

G, decreasing memory cost by replacing the least promising active leaves with inactive leaves that dominates the active ones, dropping poor attributes in order to free storage, and initialization to allow for higher accuracies for smaller number of examples etc. For further details of VDTF and Hoeffding trees, see [29].

3.3 Model evaluation

A binary classifier maps the set of predictors $\mathbf{x}$ to a binary response variable $\hat{y} \in \{0, 1\}$, with the true response y being known and the predicted response being labeled as

- True Positive (TP) when both $\hat{y}$ and y are positive,
- True Negative (TN) when both $\hat{y}$ and y are negative,
- False Positive (FP) when $\hat{y}$ is positive and y is negative, and
- False Negative (FN) when $\hat{y}$ is negative and y is positive.

The most intuitive model evaluation of classifiers is to investigate the accuracy of correct predictions, i.e.

$$\frac{\sum TP + \sum TN}{N},$$

where N is the total number of observations. However accuracy might not always be reliable, as False Positive prediction in many cases are far more harmful than False Negative ones. Additionally, for a dataset with large imbalance in the response classes, a model predicting the the majority class as response for every observation will yield a high classification accuracy despite being a useless model. This is referred to as the *Accuracy Paradox*, and to establish a trustworthy evaluation of the model, additional evaluation metrics need to be analyzed.

3.3.1 Area Under ROC-curve

The True Positive rate (TPR) of a prediction model, also known as the sensitivity or recall, measures the proportion of correctly identified positive class values, i.e.

$$TPR = \frac{\sum TP}{\sum TP + \sum FN},$$

and can be interpreted as a measure of the classifiers benefits. Furthermore, the False Positive rate (FPR), known as the false alarm ratio, measures the proportion of wrongfully classified positives, i.e.

$$FPR = \frac{\sum FP}{\sum FP + \sum TN}.$$

FPR resembles the Type I error of the decision rule, and can be interpreted as a measure of the classifiers costs.

The predicted class of each instance is based on a prediction score, often given as a continuous random variable X, and the trade-off between these TPR and FPR is illustrated by plotting these two characteristics against each other for various

discrimination thresholds $T \in [0, 1]$, such that the instance is classified as positive for $X > T$, and negative otherwise.

This graphical relation yields a *receiver operating characteristic curve* (ROC curve) (see [30]), and illustrates the diagnostic ability of the classifier. A random classification would yield $\sum TPR = \sum FPR$, and the closer the curve tends to the $[0, 1]$ coordinate of the space, the better performance by the classifier. As an extension, the *area under the ROC curve* (AUC) gives the probability that the classifier will rank a uniformly drawn random positive instance higher than a uniformly drawn negative one, i.e.

$$AUC = \int_{\infty}^{-\infty} TPR(t)FPR(t)\,\mathrm{d}t,$$

with a higher AUC score indicating a better classifier. AUC is therefore indifferent to any imbalance of the response classes, and thus a more stable metric than accuracy.

3.3.2 Variable Importance

Another evaluation method of interest when analyzing trees is the importance of each feature in the dataset. As the feature space in many data mining applications can be large, but often only a few of the model predictors having substantial impact on the response, a majority of the features might be irrelevant and could be discarded. Investigating the contribution of each variable in the prediction can therefore give a deeper understanding of the model.

For a CART T, Breiman et. al proposed in [31] a measure of relevance of each predictor $\mathbf{x}_\ell$, given as

$$\mathcal{I}_\ell^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 \mathbf{1}\left\{v(t) = \ell\right\}, \tag{3.10}$$

where $\hat{i}_t^2$ is the maximal estimated improvement. For each of the $J - 1$ internal nodes of the tree, one of the input variables $\mathbf{x}_{v(t)}$ is used to partition the region associated with the current node t into sub-regions. The chosen variable is the one which gives the maximal estimated improvement $\hat{I}_t^2$ in squared error risk over that for a constant fit over the entire region. Summing such squared improvements over all internal nodes for which $\mathbf{x}_\ell$ was chosen as the splitting variable will yield the squared relative importance for the feature. For additive tree expansions, such as gradient-boosted tree models, this importance measure is generalized by averaging over the trees, i.e.

$$\mathcal{I}_\ell^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_\ell^2(T_m).$$

Once again, averaging will stabilize and yield a more reliable measure than for a single CART in (3.10). For K-class classification, K separate models $f_k(\mathbf{x})$, $k = 1, 2, \ldots, K$ are induced, where each model consists of a sum of trees

$$f_k(\mathbf{x}) = \sum_{m=1}^{M} T_{km}(\mathbf{x}),$$

and the importance measure generalizes to

$$\mathcal{I}_{\ell k}^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_{\ell}^2(T_{km}),$$

where $\mathcal{I}_{\ell k}$ is the relevance of X_ℓ in separating the observations of class k from the other classes. The overall relevance of the feature $\mathbf{x}_\ell$ is obtained by averaging over all classes such that

$$\mathcal{I}_{\ell}^2 = \frac{1}{K} \sum_{k=1}^{K} \mathcal{I}_{\ell k}^2.$$

Variable importance for Random Forest is constructed in the same fashion as for gradient-boosted models. The importance measure attributed to the splitting variable is the improvement in the split-criterion at each split in each tree, which are accumulated over all tress in the forest separately for each feature. Where boosting might ignore some features completely, the candidate split-variable selection in Random Forest increases the chance that any single variable will be included. A different interpretation of variable importance for Random Forest, constructed on the OOB samples, is further discussed in [14].

4
Models

One objective of this thesis is to expand the ensemble decision tree learning models with an online updating phase, in order to adapt to the sequential arrival of the data stream. The offline alternatives to the stream data would either be to use the same trained model as a static model to predict new instances, i.e. new weekly observations, or to retrain a new model with the most recent data. The obstacle arising when applying the same model to predict instances in the arriving stream is to train a model with sufficient information to cover all possible arrivals. Given the yearly seasonality of football, and consequently the user activity in the app, a model trained in a shorter period of time during the year will perform poorly during a different period of the year. For example, a model based on the activities of a month of many football matches around the world will assume most users have churned when arriving to an off-season period such as July, and vice-versa for a model trained during the off-season.

For a stream with short term seasonalities and sufficient amount of available data, the static approach might be efficient. However since the data covered in this thesis has a distinct quadrennial seasonality it would require large resources of data storage, memory, computational power, and time to train a sufficient model. Resources that are not available during the progress of the thesis.

A long term issue of the static prediction model is that the general user behavior might change over time, and the model will become irrelevant to the data after some time. Reversely, the drawbacks of training a new model for each arriving instance are the loss of history, as well as the computational time increasing. When only considering the most current data of the stream, the information of the previous instances will be neglected and the predictive power derived from the seasonalities will be lost. If the consecutive instances differ significantly in user behavior and activity, the model based on the last weeks data might produce poor predictions on the current weeks data. Recalling Figure 2.2, the amount of WAU can alter with more than 200,000 users over the span of only a few weeks, and during summer, the difference will be much more severe. In order to include a short term historical span and avoid the poor performances caused by weekly amplitudes, a rolling window can be introduced for which a pre selected amount of the most current consecutive instances m are included in the training. For a rolling window the dataset containing T stream instances is partitioned into $N = T - m + 1$ subsets, where the initial window contains the data from instance 1 to m, the second window from instance $m + 1$ to $2m$ and so on. However due to the large amount of WAU, the number of observations for each window will quickly grow to be massive, and the computational training time for each window will increase rapidly as the windows grow larger.

Furthermore, each instance will be included in training m times, which appears to be an inefficient approach when the size of each instance is so large.

As the Online training methods discussed in Section 3.2.3 are fairly novel, open-source modules for general implementation are not established, the algorithms implemented in this thesis are inspired by the methodology of some of these methods.

4.1 Online adaptation of Random Forest

The initial algorithm implemented in this thesis for online improvements of Random Forest is a simplified adaptation of the *Dynamic Weighted Majority algorithm* (DWM), for which the online trained ensemble learners are weighted and removed based on their performance, and new learners are added based on their global performance. Instead of weighting the learners, the trees are ranked according to their individual prediction performance on the newest instance in the stream. A pre selected ratio of the worst performing trees are discarded and replaced with a Random Forest trained only on the newest instance. The number of trees in the new Random Forest will be determined by the number of trees that have been removed from the previous model, keeping the size of the forest constant. The pseudo code for the online adaptation of Random Forest developed in this thesis is presented in Algorithm 1.

Algorithm 1: Online Random Forest updater.

Input : $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$, stream instance
Input : $\hat{f}$, Random Forest model
Input : n, number of trees
Input : ω, replant weight

$y_{\text{pred}} \leftarrow \hat{f}.\texttt{predict}(\mathbf{x}_i)$
prediction score $\leftarrow \texttt{score}(y_i, y_{\text{pred}})$

foreach *tree* $\in \hat{f}$ **do**
  $tree\_scores \leftarrow \texttt{score}(y_i, tree.\texttt{predict}(\mathbf{x}_i))$
end
$\hat{f}.\texttt{remove}(\texttt{sorted}(tree\_scores)[: (n \cdot \omega)])$
$\hat{f}_{\text{new}} \leftarrow \texttt{RandomForest}(\mathbf{x}_i, y_i, n\_estimators = (n \cdot \omega))$
$\hat{f}.\texttt{extend}(\hat{f}_{\text{new}})$

Output: $\hat{f}$
Output: prediction score

By utilizing this online approach, the newest instances of data can be embedded into the model without retraining previously seen data, and the data can thus be discarded when used. Compared to the aforementioned static approach, the online approach is more relevant to the current data instance, and hence more likely to obtain accurate predictions. Furthermore, in contrast to a rolling window approach, the online approach will require less computational time, as only the most current

instance is considered for retraining, as well as being less restricted to memory and storage capacities, since the information from previous instances is already nested into the model.

The drawback of this approach is arguably the loss of information from previously discarded learners. As discussed in Chapter 2, the data shows several seasonality tendencies, with the significant yearly decrease during summer being the most distinct one. Inspired by the *Batch Weighted Ensemble* (BWE) and *Evolutionary-Adapted Ensemble* (EAE) architectures, an extension to Algorithm 1 is introduced, with the pseudo code for the extension being provided in Algorithm 2.

Algorithm 2: Online adaptation of Random Forest with drift detection and an unlimited storage pool for discarded learners.

Input : $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$, stream instance
Input : $\hat{f}$, Random Forest model
Input : $\mathcal{F}$, unlimited pool of trees
Input : n, number of trees
Input : ω, replant weight

$y_{\text{pred}} \leftarrow \hat{f}.\texttt{predict}(\mathbf{x}_i)$
prediction score $\leftarrow \texttt{score}(y_i, y_{\text{pred}})$
if *drift_detected* **then** `// See Algorithm 3`
  $\hat{f}_{\text{new}} \leftarrow \texttt{pool\_searcher}(\hat{f}, \mathcal{F}, \mathcal{D})$; `// See Algorithm 4`
  $y_{\text{pred}} \leftarrow \hat{f}.\texttt{predict}(\mathbf{x}_i)$
  prediction score $\leftarrow \texttt{score}(y_i, y_{\text{pred}})$
  if *drift_detected* **then**
    $\hat{f}_{\text{final}} \leftarrow \texttt{OnlineRandomForestUpdater}(\mathcal{D}, \hat{f}, n, \omega)$
    $\mathcal{F}.\texttt{append}(\hat{f}_i \notin \mathcal{F})$
  else
    $\hat{f}_{\text{final}} \leftarrow \hat{f}_{\text{new}}$
  end
else
  $\hat{f}_{\text{final}} \leftarrow \texttt{OnlineRandomForestUpdater}(\mathcal{D}, \hat{f}_{\text{new}}, n, \omega)$
  $\mathcal{F}.\texttt{append}(\hat{f}_i \notin \mathcal{F})$
end

Output: $\hat{f}_{\text{final}}$
Output: prediction score

Similarly to Algorithm 1, the new instances are initially evaluated using the current model. Different approaches to detect drift can be utilized, with the most common being to compare the obtained score to previous scores, and signal drift warning if the difference between the obtained score and the mean of previous scores is larger than a prior selected amount of standard deviations. To detect sudden concept drifts, an additional condition is implemented to signal drift when the score has dropped with a certain margin compared to the most previous score obtained by the

model. The pseudo code for a drift detecting implementation is given in Algorithm 3, in which the coefficients for number of standard deviations and accepted score drop can be altered given how strict the drift detection is desired to be.

Algorithm 3: Detecting possible current drift.

Input : *scores*, list of previous AUC scores
Input : ϵ, obtained AUC score of current instance
Input : σ, threshold for acceptable standard deviation of AUC score
Input : δ, threshold for acceptable drop ratio of AUC score

```
if ε < mean(scores) − σ · std(scores) then
    ret ← True
else if ε < δ · score[last] then
    ret ← True
else
    ret ← False
end
```

Output: ret

If no drift is detected, the schema will follow as in Algorithm 1, with the ensemble being updated according to each learner's individual prediction performance on the new instance. If a drift signal is detected however, the model investigates how the learners of the pool performs on the new instance, and the learners yielding improved scores replace the worst performing learners of the current model. If the model obtained by the best performing learners of both the current ensemble and the pool is able to overthrow the drift signal, the new ensemble will be considered as the new model. Otherwise a refreshing of the learners, given by Algorithm 1, is executed in order to gain new learners more suitable to the given instance.

Due to the pool being unlimited, the amount of learners will ultimately demand large storage and slow down computational time when iteratively finding the best adapted learners for the drifting instance. In order to maintain a sufficiently small pool size, the pool is only updated with new learners when being necessary, i.e. when the most suitable ensemble of learners still cannot prevent drift. However as the instances are arriving as a stream, the pool will be gradually extended with a few of the best performing new learners in order to maintain relevant to the changes in data. The pseudo code for an algorithm searching through the pool for more suitable learners is implemented in Algorithm 4.

4.2 Online adaptation of XGBoost

Due to the sequential manner in which boosting algorithms are learned, the structure of XGBoost appears to be well suited for a data stream without any major additions to the algorithm. For the weekly arrival of new data, the model learned from the previous weeks will be utilized as an initial model, for which the weights corresponding to the classifiers in the ensemble are redistributed with respect to the errors they made in predicting the most recent data. An extension of the model

Algorithm 4: Searching through pool for more suitable learners.

Input : $\hat{f}$, Random Forest model
Input : $\mathcal{F}$, pool of learners
Input : $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$, stream instance

foreach $l \in \mathcal{F}$ **do**
  $pool\_scores \leftarrow \texttt{score}(y_i, l.\texttt{predict}(\mathbf{x}_i))$
end

foreach $tree \in \hat{f}$ **do**
  $tree\_scores \leftarrow \texttt{score}(y_i, tree.\texttt{predict}(\mathbf{x}_i))$
end

$sorted\_pool \leftarrow \texttt{sort}(pool\_scores, descending)$
$sorted\_tree \leftarrow \texttt{sort}(tree\_scores, ascending)$

while $sorted\_pool[0] > sorted\_tree[0]$ **do**
  if $\texttt{tree\_id}(sorted_pool[0]) \notin \hat{f}$ **then**
    $\hat{f}.\texttt{remove}(sorted\_tree[0])$
    $\hat{f}.\texttt{extend}(sorted\_pool[0])$
    $sorted\_tree \leftarrow \texttt{sort}(sorted\_tree, ascending)$
    $sorted\_pool \leftarrow \texttt{sort}(sorted\_pool, descending)$
end

Output: $\hat{f}$

based on the most recent instance will be implemented and saved as the initial model of future instances. The pseudo code of the online adaptation of XGBoost for this thesis is presented in Algorithm 5.

Algorithm 5: Online adaptation of XGBoost.

Input : $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$, stream instance
Input : ϕ, XGBoost model
Input : n, number of expansion trees

1 $y_{\text{pred}} \leftarrow \phi.\texttt{predict}(\mathbf{x}_i)$
2 prediction score $\leftarrow \texttt{score}(y_i, y_{\text{pred}})$
3 $\phi_{\text{extended}} \leftarrow \texttt{train\_XGB.}(\mathbf{x}_i, y_i, n, \phi)$
Output: ϕ_{extended}
Output: prediction score

As for the online adaptation of Random Forest, no previous data is saved, reducing the storage and memory limitations compared to a rolling window approach, while incorporating both historical and current behaviors in the model. Compared to the online adaptation of Random Forest the online adaptation of XGBoost might be able to maintain more information from the previously seen instances, since no trees are being replaced in the model. As a consequence of this, the online adaptation of XGBoost might become too large and therefore prone to overfitting, as well as becoming burdensome on the storage capacities when being implemented for a long term.

5
Results

The models implemented for this thesis are considering instances aggregating the activity behavior within the app during the past week, evaluated every Saturday during the period 2016-05-01 to 2018-05-01. As the data is considered as a stream, each instance will at arrival be initially considered as a test set and thereafter as a training set extending the model. The sizes of the instances vary in order to maintain a relevant user base, and a summary of the sizes is provided in Table 5.1. Since the size of the full dataset is extremely large, a slightly smaller stream will be considered initially for the purpose of tuning the model parameters. This initial stream is reduced to include the trend of the two most current months, i.e. from 2018-03-01 to 2018-05-01. The corresponding size summaries of instances in the minor stream is also provided in Table 5.1.

Table 5.1: Summation of instance sizes for streams being evaluated, including number of instances, average amount of observations, standard deviation, smallest and largest instance, median instance, and total number of observations in each stream.

Interval	2016-05-01 to 2018-05-01	2018-03-01 to 2018-05-01
Instances	104	9
Mean	2997558	2931474
Std	± 184575	± 42003
Min	2306270	2888659
Median	2992750	2914231
Max	3467159	3009626
Total	311746113	26383266

5.1 Parameter tuning

The models considered are based on online adaptations of Random Forest and XGBoost (as given in Algorithms 1, 2, and 5) with the common hyperparameters to tune being number of trees ($n_{\text{trees}} = 10, 50, 100$) and percentage of trees being replaced (Random Forest) or added (XGBoost) for the updated model ($\omega = 0.0, 0.1, \ldots, 1.0$). The total number of models to be compared will hence be $2 \times 3 \times 11 = 66$, and the performance of each model is evaluated based on their accuracy and AUC score. The resulting performances of each model during the term 2018-03-01 to 2018-05-01 are illustrated as boxplots in Figure 5.1.

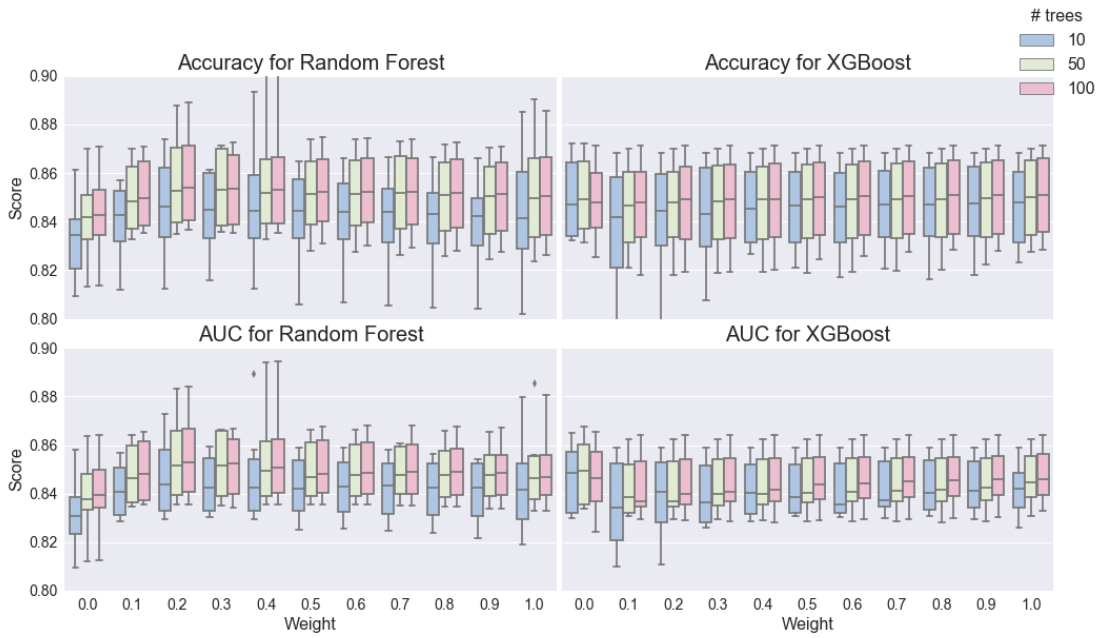


Figure 5.1: Boxplots of the accuracy (upper row) and AUC (lower row) for predicting weekly activity. Random Forest (left column) and XGBoost (right column) models with various values for the number of trees and replant weight parameters are considered between 2016-05-01 to 2018-05-01. As illustrated, the Random Forest models are more affected by the choice of parameters compared to the XGBoost models.

As seen in the figure, the Random Forest models (left column) perform better when being updated with new trees for each instance. Compared to when the replant weight is 0, i.e. the initial model is consistent through all instances in the stream, both evaluation scores indicates that updating the model with new learners is slightly more beneficial. However replanting a larger portion of the learners appear to have no additional improvement compared to a replant weight of 20%, as the evaluation scores stabilizes. It furthermore is noticeable that a replant weight of 1, i.e. when the model is solely based on the previous instance, appears to be more suitable than having no updates at all. For this stream it hence appears that an online updating approach considering both historical and current data is beneficial for model performance.

Regarding the influence of forest size it is noticeable that for low replant weights, where most or all trees remain in the forest through all instances, and large replant weights, where most or all trees are replaced in the model for each new instance, the performance increases as the size increases. For the remaining Random Forest models, the performance differences are marginal between 50 and 100 trees, whereas for $n_{\text{trees}} = 10$ the model is consistently performing worse (but still surprisingly well considering the small ensemble size).

For the XGBoost models (right column) the contributions from the replant weights appear to be rather or negligible, with the models performing very simi-

larly for all considered weights. The number of trees however appears to have some influence on model performance, as the evaluation scores are consistently larger for more trees, however not significantly. The smallest models ($n_{\text{trees}} = 10$) have a larger variance for low weights, i.e. when the initial model contains few trees and the expansion ratio is slow, and are outperformed for larger weights. For the models containing more trees, it is noticeable that the largest model ($n_{\text{trees}} = 100$) constantly achieves a slightly better score.

For the XGBoost models it is important to recall that the new trees trained on the new instance are not replacing the trees in current model, but rather extending it. However the weight of 1 is, for comparison reasons, utilized as discarding the current model altogether and train a new model on the latest instance. For all models, it is obvious that the AUC and accuracy yields rather proportional results. This indicates that the models do not only provide accurate predictions, but are also stable and unbias in terms of prediction imbalance.

5.2 Computational time

The computational time elapsed for receiving a new instance and updating the current version is displayed in Figure 5.2 for each model. As distinctly illustrated in the figure, the XGBoost based models require significantly less computational time compared to Random Forest, especially considering the logarithmic scale. The obtained results are expected considering XGBoost is a more recent publication with one of the key objectives being to be more scalable than current methods, an objective reached by exploiting out-of-core computation as well as parallelization. As the figure shows, the speed up of XGBoost is 2 to 50 times faster compared to Random Forest. Comparing the Random Forest models, the computational time grows exponentially with respect to the weight. The number of trees also have an impact, with the computational time being approximately 10 times the number of trees. Going from weight 0, i.e. no renewal of the model, to 0.1 a radical escalation in computational time can be noticed, indicating that the process of selecting which learner to remove is disproportionately expensive to training a new model.

For the XGBoost models, the computational time is near constant to the non-updating model for all the expanding weights. However when discarding and retraining the entire model ($\omega = 1$), the computational time increases, and the dependence of the model size is more evident.

An additional observation from Figure 5.2 is for weight 0, where the same model is used for all instances. The computational time required appears to be independent of the number of trees whereas the computational time increases as the number of trees grow in Random Forest. This illustrates the difference between the bagging and boosting approaches, where bagging summarizes the predictions of all its base learners, whereas boosting has combined the learners into one on which the prediction is executed. Hence the boosting models are not affected by the number of trees growing in the same fashion as the bagging models are.

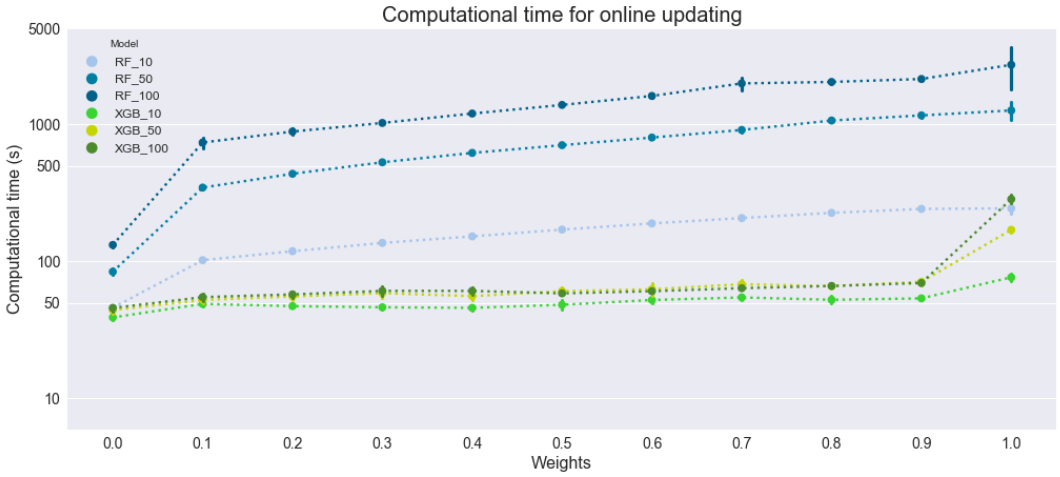


Figure 5.2: Computational time for online updates for various models, where `RF` abbreviates Random Forest, `XGB` abbreviates XGBoost, and the number of the model is the amount of trees in the ensemble. Note the logarithmic scale on the axis for computational time.

5.3 Performance on entire data stream

The accuracy and AUC scores of each instance in the entire considered data stream from 2016-05-01 to 2018-05-01 are illustrated in Figure 5.3. As the computational time of predicting each instance of the full data stream is several hours for each model, only a few models were selected is considered in this section. From Figure 5.1, the Random Forest model with $n_{\text{trees}} = 50$ and $\omega = 0.2$ showed promising performance scores and qualified as a interesting model for prediction on the full dataset. A model with larger replant weight was also of interest, and the Random Forest model with $n_{\text{trees}} = 50$ and $\omega = 0.2$ was considered for the full dataset. As the XGBoost models were negligible of parameter choice, the model with $\omega = 0.9$ was chosen as it is the model gaining most information from new instances. The XGBoost model with $\omega = 1$ was chosen to compare with the performance of renewing the model by each instance. Finally, the Random Forest model remaining unchanged during the stream was included as baseline. All models considered had a tree size $n_{\text{trees}} = 50$.

Figure 5.3 shows that the constant Random Forest model with weight 0 is consistently outperformed by the other models, indicating that further information can be gathered from newer observations. The other model's ability to adapt to current observations is most obvious in the concept drifts of the data, for which the updated model recovers from the drop in accuracy faster than the constant one. Furthermore it is noticeable that the fluctuations in Figure 5.3 show a clear similarity to the fluctuations in Figure 2.5, illustrating that a correlation between the considered score metrics and the amount of WAU is present.

One concern with sequentially updating the model is that the information from older models not reflecting the current observations will eventually be neglected

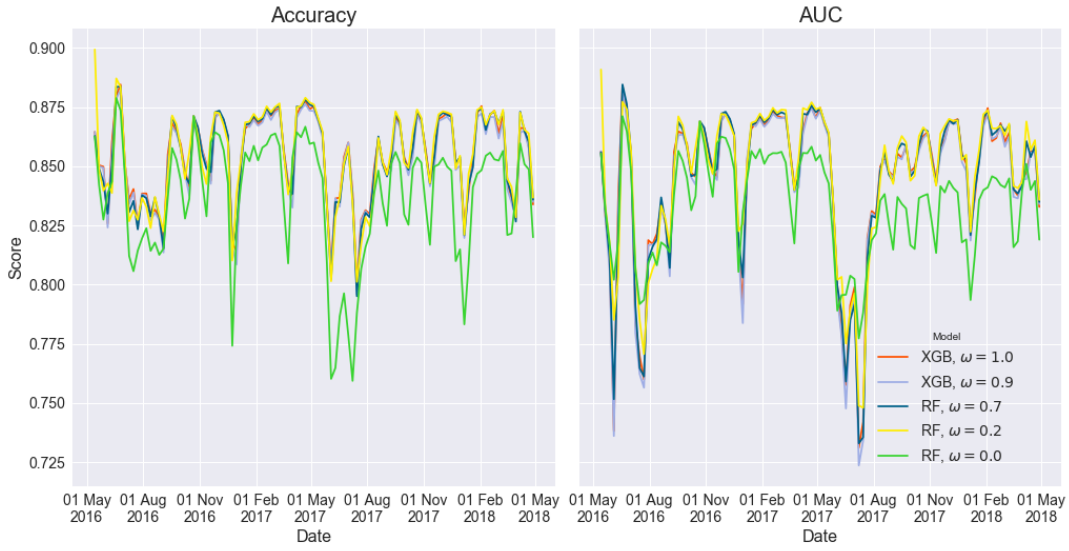


Figure 5.3: Sequential accuracy (left) and AUC (right) scores for selected Random Forest and XGBoost models being updated live for the stream of weekly activity between 2016-05-01 to 2018-05-01.

from the model. While this in general is a desired quality of the online approach, it causes poor performances when concept drifts occur. The AUC graph (right) in Figure 5.3 illustrates clearly how a higher value of ω, i.e. a higher adaptation rate ($\omega = 0.7$) results in worse performance during concept drifts than the models that maintain information of older instances for a longer time ($\omega = 0.2$). Noteworthy from the figure is also how the accuracy recovers quickly from concept drift whereas the AUC drops significantly. This indicates that accurate predictions are made, but that they are biased towards one prediction class, most likely the negative one as the drops occur during the significantly less active summer periods.

Given Figure 5.3 it might appear that the concept drifts are sudden. However recalling from Figure 2.2 the drop in WAU shows an annual seasonality pattern of the concept drifts. The concept drifts might therefore be detected in an earlier stage, and a more appropriate model might be replicated by earlier discarded learners. Following the schema of Algorithm 2, an extension of previous models including a drift detector and an unlimited pool storing all previously used base learners is implemented. The obtained prediction evaluations are illustrated in Figure 5.4. The extension of the Random Forest model with $\omega = 0.2$ is compared to the original model and the baseline of Figure 5.3. As shown in Figure 5.4, the extended model achieves a slightly better AUC score than the previous models during drifts. However the decrease is still significant during less active periods.

In order to investigate the contribution of the extended model, Figure 5.5 illustrates the number of replaced base learners. Comparing the findings of Figure 5.5 with the variations in evaluation performance of Figure 5.4, it is clearly visible that the model is able to detect drift. No learners are replaced during the periods of which evolution scores are high, whereas the number of replaced learners are close to

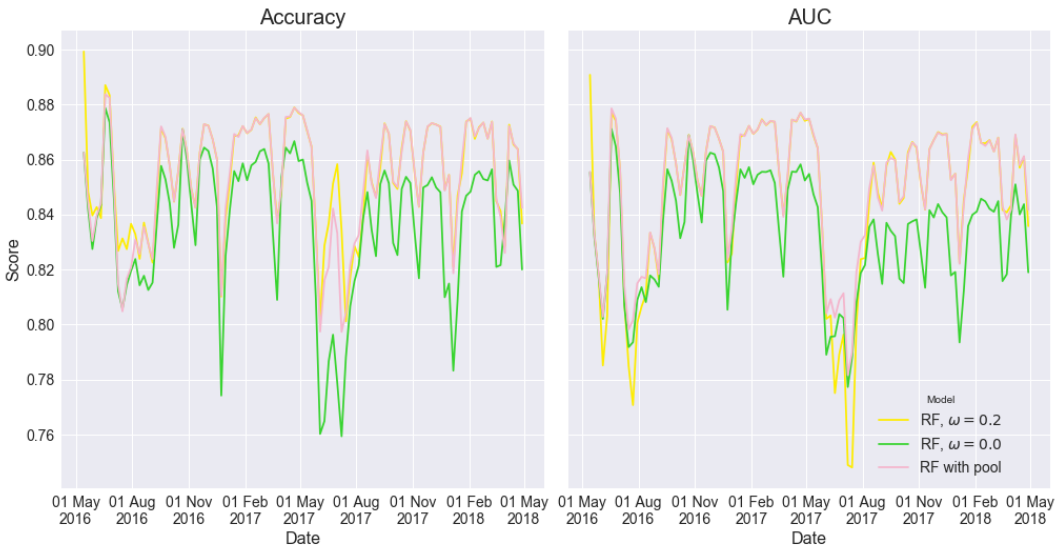


Figure 5.4: Sequential accuracy (left) and AUC (right) scores for the extended online Random Forest model with drift detection and unlimited storage pool, as well as previously observed models being updated live for the stream of weekly activity between 2016-05-01 to 2018-05-01. Utilizing the extension of a pool increases the performance slightly.

negatively proportional to the drop of performance scores during the corresponding dates.

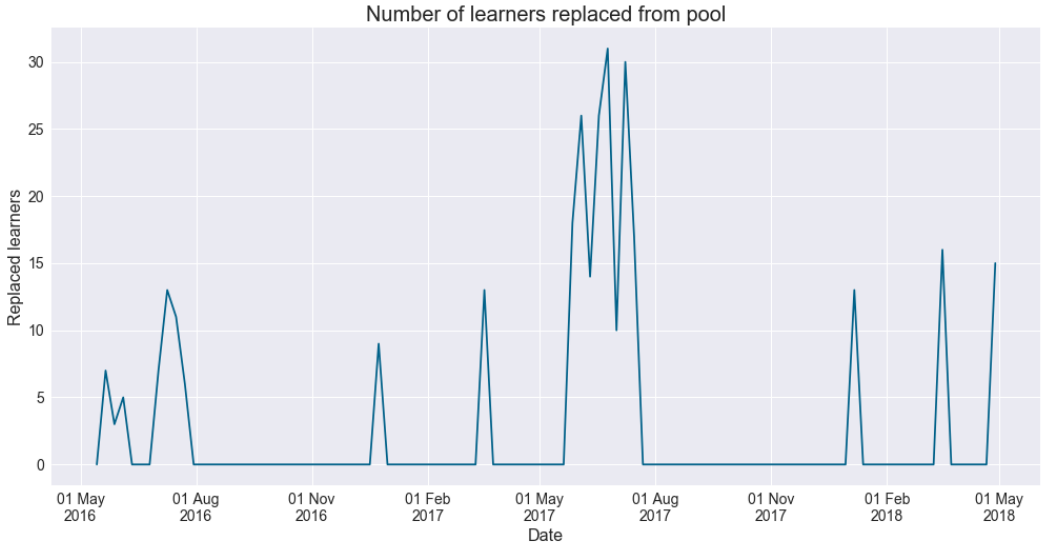


Figure 5.5: Number of learners in current model being replaced with learners stored in the pool. Note that significantly more learners are being replaced during the summer.

5.4 Feature Importance

The summation of the Feature Importance measures for all submodels of a Random Forest online model with replant weight $\omega = 0.2$ is given in Table 5.2. The table illustrates that the generated features regarding engagement and loyalty are consistently among the most important features for model performance, showing that generating these variables have an impact on the model. Furthermore the features generated with respect to the followed teams or tournaments are in general ranked higher than the features generated with respect to favorite team. The reason for this might be that the followed teams have been actively selected by the users whereas the favorite team is given by the user activity, such that the behaviors in viewing the favorite are more diverse. Is is also worth to recall that a user can follow multiple teams and tournaments but only one team is considered as the favorite at a time, such that a followed team will be playing more regularly than a favorite team.

The most important feature appears to be the aggregation of active days during the past week. This conclusion is not very surprising given that this feature more or less is the vague base for the other features based on the past week. Among the other aggregated features it appears that the views of the match event, lineups and highlights are important for the model, which is also expected given that these are the tabs most informative for the match updates. Furthermore a slight importance can be seen for viewing the calender tab, especially when viewing the current and past results for the favorite team.

The categorical features appear to have generally low importance. Recalling the observations of Section 2.4.4, the distribution between viewing and not viewing users was proportional for most classes in each categorical feature, and it is hence reasonable that the model in general do not gain any significant information from the users class affiliations. However a few exceptions to this generalization are found, where observations of classes such that addicts, lazy users, and users without a given type have a more significant importance. Also the feature of having no favorite team shows a higher importance than other league affiliations, which is persistent to the illustrations of Section 2.4.4

The obtained importance for the, on average, highest ranked features is illustrated in Figure 5.6 for each submodel of the Random Forest Online model with weight $\omega = 0.2$ during the considered data stream. As the figure illustrates, the importance, as well as the rank, of each features varies significantly between instances, showing once again the benefits of including the most current observations to the model. Considering the two most important features, an interesting inversely proportionality can be observed. During the considered period, the `followed_engagement` often appears to be decreasing in importance as the `days_active_last_7_days` feature increases, and vice versa. This observation is most obvious during the less active summer periods, where the importance of `followed_engagement` increases significantly and outperforms the other features with a large margin, whereas the usually most important `days_active_last_7_days` feature decreases drastically during the same period. Furthermore an interesting observation is the long term decrease in importance of `follow_engagement`, opposite to the continuous increase of importance for `favorite_engagement`.

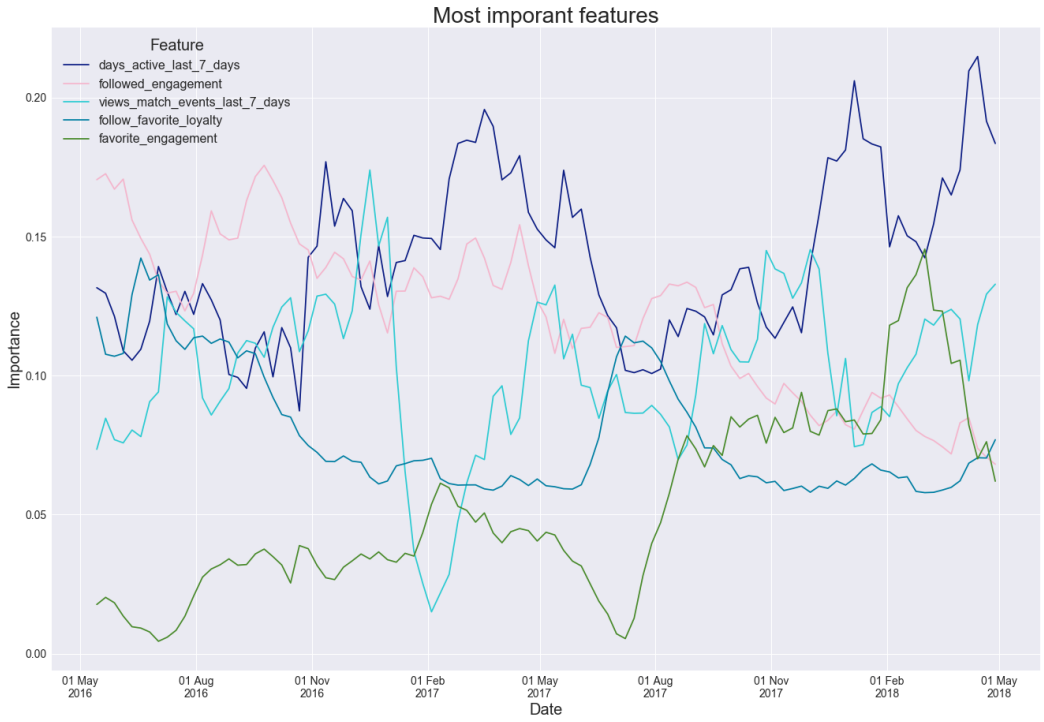


Figure 5.6: Importance progress of the features with highest average importance (see Table 5.2). As seen, the importance, as well as rank, vary for the features with respect to time of year.

Table 5.2: Summary of importance for all features considered in a Random Forest Online model with replant weight $\omega = 0.2$.

	Mean	Std	Min	Median	Max
days_active_last_7_days	0.142268	±0.029113	0.087298	0.139761	0.214810
followed_engagement	0.121890	±0.028079	0.068119	0.127717	0.175648
views_match_events_last_7_days	0.101692	±0.028685	0.015027	0.104957	0.173967
follow_favorite_loyalty	0.079113	±0.022678	0.057891	0.068832	0.142300
favorite_engagement	0.052664	±0.032747	0.004432	0.043379	0.145520
views_lineup_last_7_days	0.050744	±0.012738	0.025346	0.050312	0.079640
num_teams_followed	0.041112	±0.007823	0.032227	0.038642	0.060263
views_results_favorites_last_7_days	0.040909	±0.017722	0.002827	0.046048	0.072518
followed_tournaments_during_season	0.039824	±0.006023	0.026385	0.039231	0.053827
No_type	0.039288	±0.012412	0.011109	0.038517	0.068841
nbr_followed_teams_playing	0.034670	±0.005091	0.022954	0.034003	0.048844
views_highlights_last_7_days	0.023106	±0.007600	0.010953	0.022082	0.047992
favorite_team_playing	0.019999	±0.018702	0.001122	0.013047	0.072380
addicts	0.019852	±0.013423	0.004123	0.016668	0.069650
views_results_all_last_7_days	0.018529	±0.007358	0.001783	0.020411	0.034972
views_next_match_favorites_last_7_days	0.015536	±0.008005	0.000799	0.017187	0.033201
lazy	0.013269	±0.007766	0.004988	0.011395	0.042881
No_favorite	0.012892	±0.009158	0.000000	0.013980	0.033477
theme_changed_all_time	0.010887	±0.002522	0.006081	0.010112	0.018893
media_link_click_last_7_days	0.010433	±0.003349	0.004428	0.010069	0.017971
views_recap_favorites_last_7_days	0.009246	±0.004601	0.000744	0.009169	0.019904
push_notifications_enabled	0.008609	±0.006695	0.005406	0.006168	0.037837
views_match_statistics_last_7_days	0.007869	±0.002764	0.003413	0.007442	0.016870
views_next_match_all_last_7_days	0.005956	±0.003422	0.000551	0.005858	0.017127
views_planning_favorites_last_7_days	0.005915	±0.002802	0.000753	0.005375	0.013425
all_scrollers	0.005869	±0.002325	0.001205	0.005777	0.014364
views_recap_all_last_7_days	0.005302	±0.002702	0.000573	0.005628	0.011227
livefeeders	0.004772	±0.000788	0.003677	0.004655	0.008177
Europe	0.004632	±0.000375	0.004072	0.004500	0.005883
retro_checkers	0.004420	±0.000709	0.003409	0.004218	0.006610
vote_last_7_days	0.004318	±0.001217	0.002782	0.003893	0.007883
favorite_in_national_league	0.004131	±0.002347	0.000000	0.004870	0.007616
Premier_League	0.003909	±0.000934	0.001305	0.003820	0.007257
views_team_fixtures_last_7_days	0.003873	±0.000964	0.002435	0.003720	0.005951
Other	0.003722	±0.000618	0.003180	0.003552	0.006206
Serie_A	0.003691	±0.000848	0.001702	0.003462	0.005237
overlookers	0.003146	±0.001401	0.001018	0.002913	0.009256
North_America	0.002981	±0.000411	0.002385	0.002894	0.004154
La_Liga	0.002916	±0.000796	0.000900	0.002997	0.004858
ad_click_last_7_days	0.002428	±0.000742	0.001284	0.002238	0.004897
South_America	0.002407	±0.000163	0.002192	0.002373	0.002987
views_plannning_all_last_7_days	0.002033	±0.001025	0.000280	0.001849	0.004364
Central_America	0.001997	±0.000149	0.001736	0.001988	0.002348
Middle_East	0.001956	±0.000154	0.001557	0.001906	0.002306
Asia	0.001864	±0.000144	0.001520	0.001850	0.002181
Bundesliga	0.001705	±0.000364	0.000489	0.001818	0.002014
views_team_table_view_last_7_days	0.001363	±0.000302	0.001015	0.001274	0.002281
Australia	0.001119	±0.000110	0.000897	0.001101	0.001442
Ligue_1	0.001060	±0.000199	0.000446	0.001037	0.001524
Africa	0.001049	±0.000213	0.000779	0.000957	0.001712
Unknown_region	0.000376	±0.000051	0.000322	0.000353	0.000519
views_opinion_last_7_days	0.000286	±0.000110	0.000059	0.000258	0.000531
share_last_7_days	0.000200	±0.000052	0.000115	0.000197	0.000322

6
Conclusion

The objective of this thesis was to predict retention among users of the app Forza Football and exploit these predictions for preventing possible churning users. Continuous instances of user activity were mined and processed to obtain a two year data stream, including both native and generated features. The fundamental theory of various tree based ensemble classification methods was presented, along with theoretical discussions regarding managing large scale data in an efficient manner. Based on the discussion of various large scale adaptations, online learning was considered the most applicable technique given that new data instances arrive to the stream sequentially. The main objective with applying online learning was to adapt the predictive models to the evolving data streams and still remain scalable and time efficient. Utilizing ensembles of learners appeared appropriate, given that their facilitation of multiple classifiers enable a mixture of preserved and recent learners, as well as simplifying the renewal process. Online ensemble learning models based on XGBoost and Random Forest were implemented for various model sizes and replacement ratios. As most methods in the field of online learning are still state-of-the-art, open-source implementations suitable for general purposes are still rare, and simpler adaptations were implemented from scratch. An extension of the online adaptation of Random Forest with concept drift detection and an unlimited pool for storage of previously discarded learners was also implemented, as the data shows obvious activity seasonalities and therefore benefit from recalling learners suitable for the drift periods.

The XGBoost models and Random Forest models obtain excellent prediction scores, even for a static model using the initial ensemble without replacements. The prediction performance increases however when adapting online learning, and the ratio of learners being replaced for each update varies slightly and can be tuned based on preferences of historical or present data for the model. The extended Random Forest implementation managed to detect drift in a satisfactory way, and culminated in a slightly improved model in terms of stability and unbiased predictions.

Considering computational efficiency, XGBoost yielded more scalable models compared to Random Forest and demanded significantly less computational time for predicting and updating each instance of the data stream. Furthermore, whereas the computational time for each update of the Online Random Forest adaptation showed proportionality with respect to ensemble size, XGBoost appeared to be almost indifferent in computational time with respect to both ensemble size and ratio of replaced learners.

The importance of the considered features was also investigated, where the features generated during preprocessing had a major impact. Additionally, the aggre-

gated amount of visits to the most general pages of the app, as well as the disseminated usage of the app during the previous week also showed great importance for prediction. The categorical features showed in general much lower importance for the predictive model, with the exception of certain user clusters.

6.1 Future work

Firstly, with accurate prediction of upcoming user activity being accessible, prevention of user churn can be further developed. Users predicted to become inactive or obtaining a false positive misclassification can be targeted for receiving a suitable push notification, or be highlighted for further tracking. As no absolute definition of churning users is given for the non subscribing users of the Forza Football application, appropriate restrictions on when and how to advance the user are vague and might be handled differently with respect to user cluster type.

Secondly, as the models in this thesis were implemented from scratch, further development of the algorithms might result in computational improvements, but might also benefit in regard of prediction performance. Especially the processes of replacing learners of the ensembles might benefit from optimization analysis in order to decrease execution time. For the extended Online Random Forest model with an unlimited pool, a computational speed up is likely possible by implementing a more efficient search algorithm when finding the best performing learners of the pool.

Recalling the obtained results of Chapter 5, the prediction performance of the more active periods of the season are outstanding, whereas for the less active off seasons the performance drops drastically alongside the number of WAU. Even though the results can be considered as good in terms of prediction scores, further investigation of suitable modeling for the off season might be of interest.

The models implemented in this thesis are all restricted to replacing a fixed number of learners for each iteration, and a further development of the algorithms might be to introduce more dynamic replacement weights, e.g. cross-validating the proposed new learners based on the newest instance and only add the ones outperforming the current learners in the ensemble.

As mentioned in Section 2.1, the response variable of this thesis could be further expanded to a count of weekly viewed matches. In order to reduce the overdispersion, a mixture of Poisson regression models can be investigated. The mixture might, as an example, be distinguished by any of the available categorical features, such as region or, probably most suitably, user type.

Bibliography

[1] Art Weinstein. Customer retention: A usage segmentation and customer value approach. *Journal of Targeting, Measurement and Analysis for Marketing*, 10(3):259–268, Feb 2002.

[2] Lawrence Ang and Francis Buttle. Customer retention management processes: A quantitative study. *European Journal of Marketing*, 40(1/2):83–99, 2006.

[3] Fred Reichheld and Christine Detrick. Loyalty: a prescription for cutting costs. *Marketing Management*, 12(5):24, 2003.

[4] Jaishankar Ganesh, Mark J. Arnold, and Kristy E. Reynolds. Understanding the customer base of service providers: An examination of the differences between switchers and stayers. *Journal of Marketing*, 64(3):65–87, 07 2000. Copyright - Copyright American Marketing Association Jul 2000; Last updated - 2011-10-24; CODEN - JMKTAK; SubjectsTermNotLitGenreText - United States; US.

[5] Hyunseok Hwang, Taesoo Jung, and Euiho Suh. An ltv model and customer segmentation based on customer value: a case study on the wireless telecommunication industry. *Expert Systems with Applications*, 26(2):181 – 188, 2004.

[6] C. B. Bhattacharya. When customers are members: Customer retention in paid membership contexts. *Journal of the Academy of Marketing Science*, 26(1):31, Dec 1998.

[7] Mark R. Colgate and Peter J. Danaher. Implementing a customer relationship strategy: The asymmetric impact of poor versus excellent execution. *Journal of the Academy of Marketing Science*, 28(3):375, Jun 2000.

[8] Barbara R. Lewis and Graham H. Bingham. The youth market for financial services. *International Journal of Bank Marketing*, 9(2):3–11, 1991.

[9] Susan M. Lloyd. The kids market: Myths and realities (1st edition). *Journal of Consumer Marketing*, 17(7):627–637, 12 2000.

[10] FootballAddicts AB. Forza Football app. `https://www.forzafootball.com/apps/`, 2012. [Online; accessed 2018-06-07].

[11] Fédération Internationale de Football Association (FIFA). FIFA World Cup. `https://www.fifa.com/worldcup/`, 2018. [Online; accessed 2018-08-14].

[12] Union of European Football Associations (UEFA). UEFA European Championship. https://www.uefa.com/uefaeuro/index.html, 2018. [Online; accessed 2018-08-14].

[13] South American Football Confederation (CONMEBOL). Copa América. http://www.conmebol.com/en/copa-america-2015-en/history, 2018. [Online; accessed 2018-08-14].

[14] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, second edition, 2009.

[15] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.

[16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[17] Martin Hilbert and Priscila López. The world's technological capacity to store, communicate, and compute information. *Science*, 332(6025):60–65, 2011.

[18] Apek Mulay. Sustaining moore's law: Uncertainty leading to a certainty of iot revolution. *Synthesis Lectures on Emerging Engineering Technologies*, 1(1):1–109, 2015.

[19] Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael I. Jordan. A scalable bootstrap for massive data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(4):795 – 816, 2014.

[20] P. J. Bickel, F. Götze, and W. R. van Zwet. *Resampling Fewer Than n Observations: Gains, Losses, and Remedies for Losses*, pages 267–297. Springer New York, New York, NY, 2012.

[21] Ping Ma and Xiaoxiao Sun. Leveraging for big data regression. *WIREs: Computational Statistics*, 7(1):70 – 76, 2015.

[22] Nan Lin and Ruibin Xi. Aggregated estimating equation estimation. *Statistics and Its Interface*, 1:73 – 83, 01 2011.

[23] Bartosz Krawczyk, Leandro L. Minku, João Gama, Jerzy Stefanowski, and Michał Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132 – 156, 2017.

[24] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2), 2004.

[25] W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 377–382, New York, NY, USA, 2001. ACM.

[26] J. Zico Kolter and Marcus A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8:2755–2790, December 2007.

[27] Magdalena Deckert. Batch weighted ensemble for mining data streams with concept drift. In Marzena Kryszkiewicz, Henryk Rybinski, Andrzej Skowron, and Zbigniew W. Raś, editors, *Foundations of Intelligent Systems*, pages 290–299, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[28] Konrad Jackowski. Fixed-size ensemble classifier system evolutionarily adapted to a recurring context with an unlimited pool of classifiers. *Pattern Analysis and Applications*, 17(4):709–724, Nov 2014.

[29] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM.

[30] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861 – 874, 2006. ROC Analysis in Pattern Recognition.

[31] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984.

A

Figures

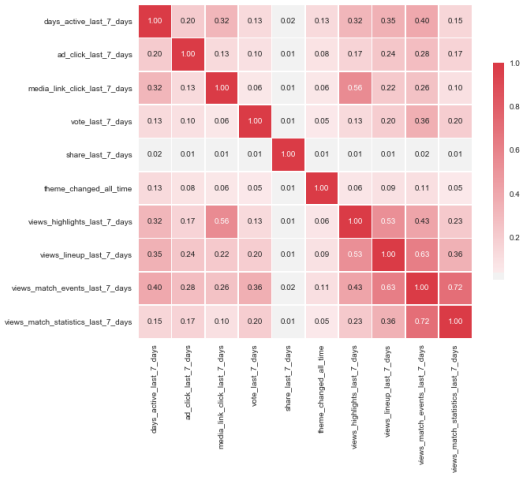
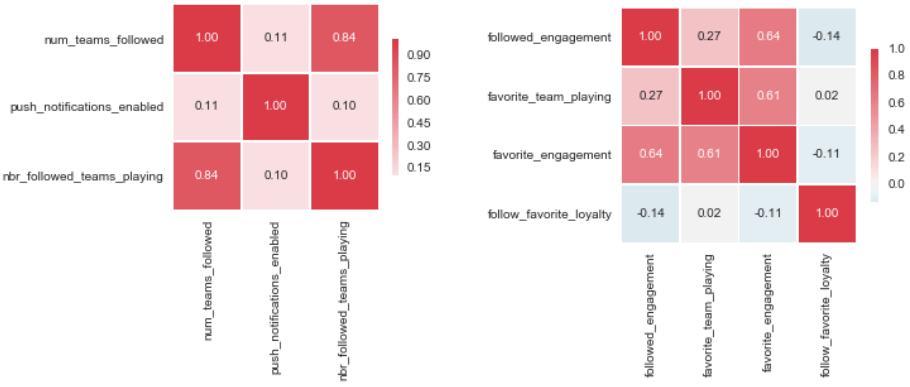


Figure A.1: Correlation matrix of match page view related features.



(a) Team following and notification related features.

(b) Generated features regarding engagement and loyalty.

Figure A.2: Correlation matrices of selected blocks of data.

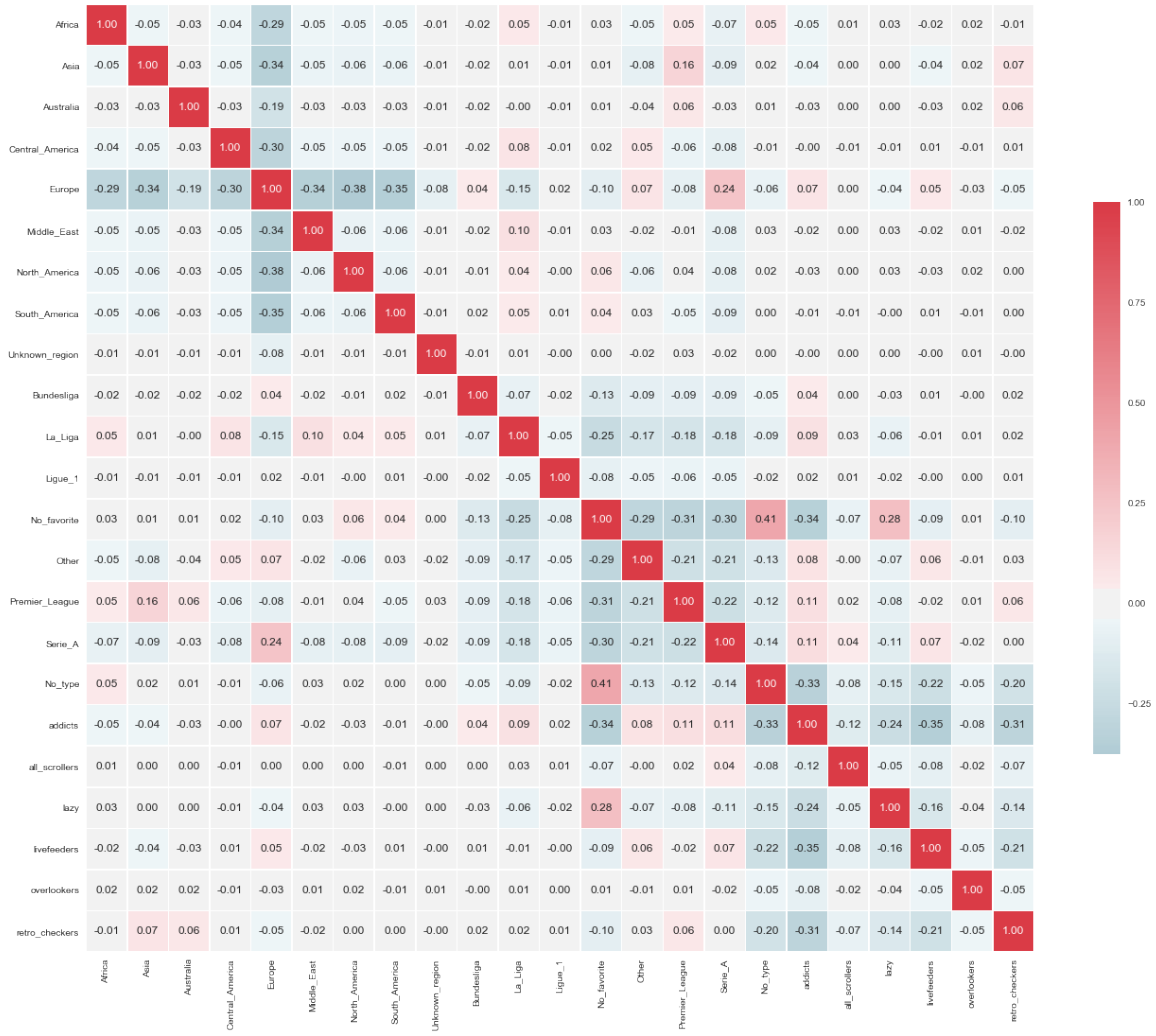


Figure A.3: Correlation matrix of the dummy variables given by the categorical features.

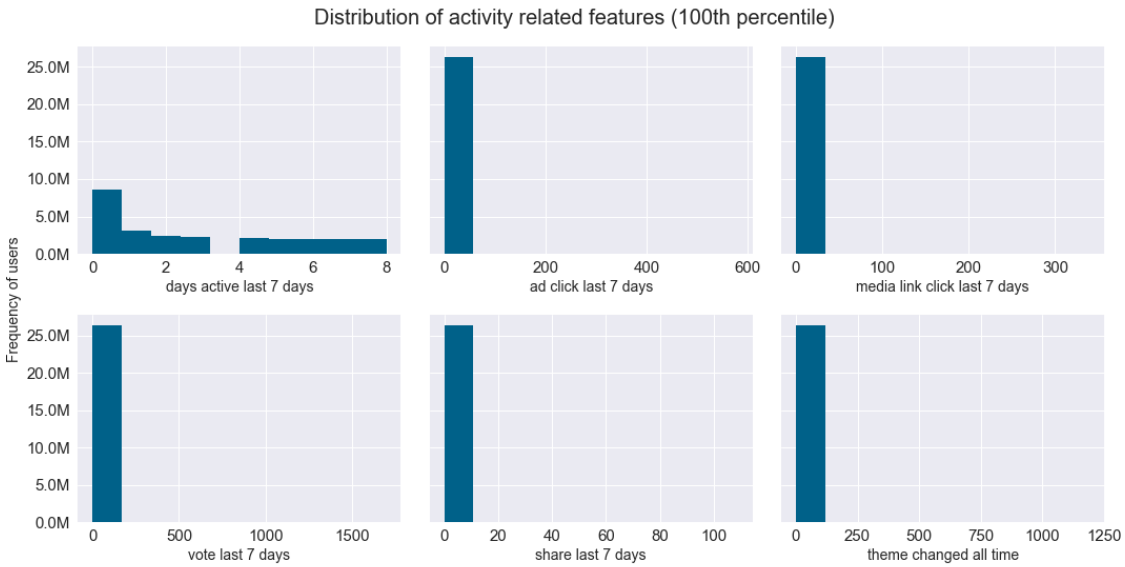


Figure A.4: Distribution of all observations of activity related features.

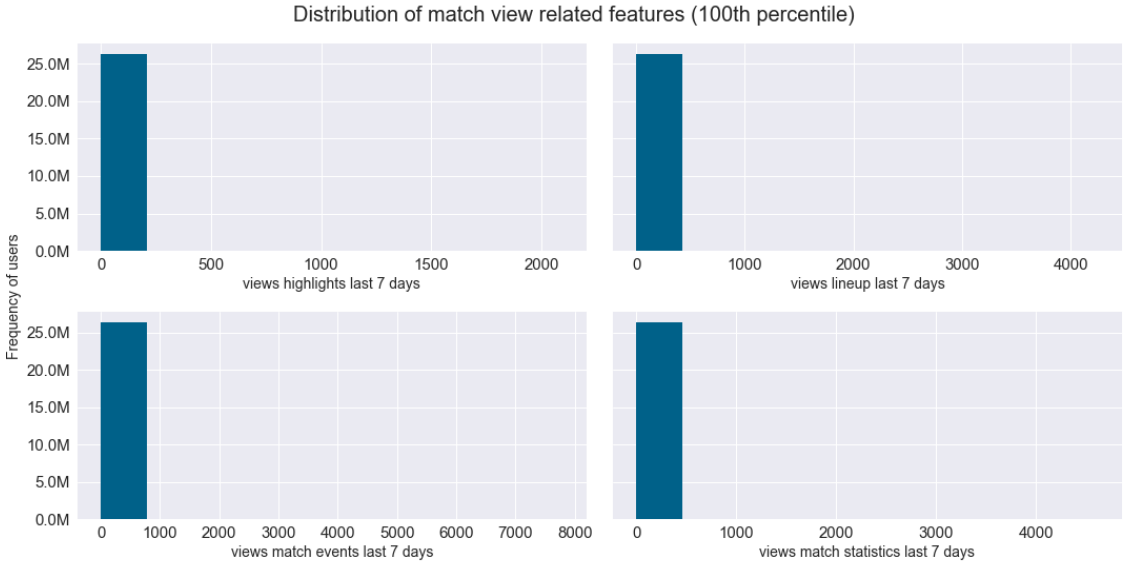


Figure A.5: Distribution of all observations of match view related features.

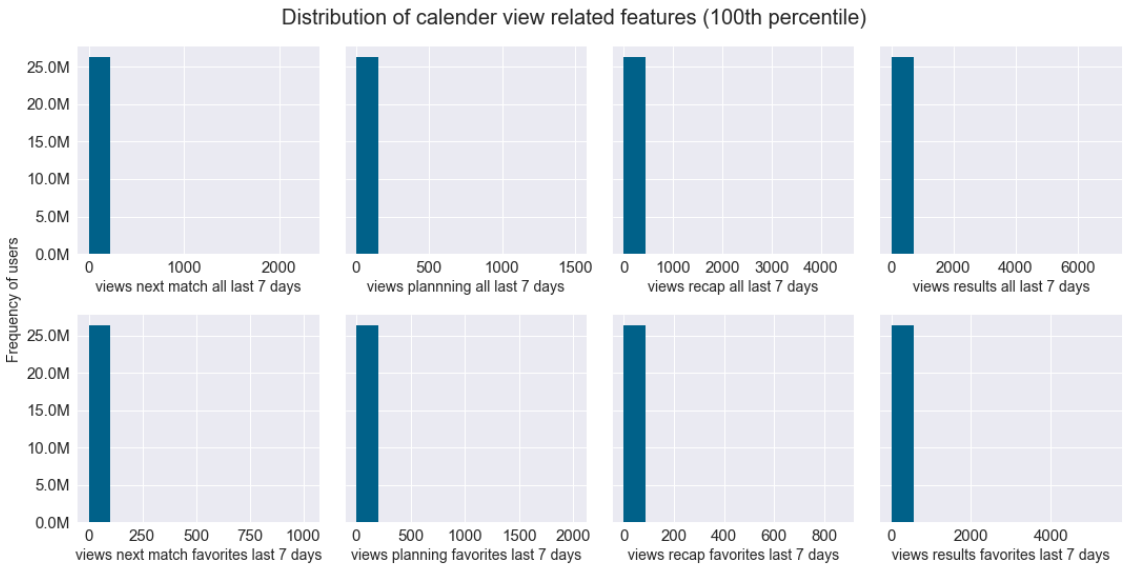


Figure A.6: Distribution of all observations of calender related features.

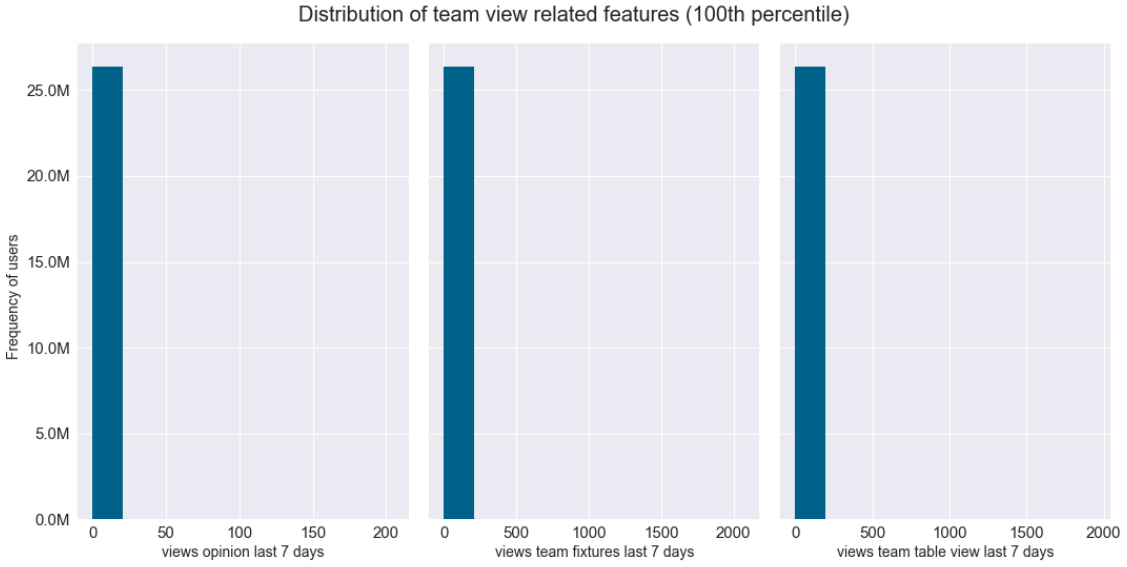


Figure A.7: Distribution of all observations of team page related features.