TUHH

Master Thesis

Exploration of Static and Temporal Machine Learning Approaches to Non-Contractual Churn Prediction

Author:

Natalya Furmanova

Supervisor:

Prof. Dr. Ralf Moeller

A thesis submitted in fulfilment of the requirements

for the degree of Master of Science

in the

Department or Software, Technology and Systems

October 2013

Declaration of Authorship

I, Natalya Furmanova, declare that this thesis titled, 'Exploration of Static and Temporal Machine Learning Approaches to Non-Contractual Churn Prediction' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Abstract

The Internet Gaming industry is booming. With thousands and millions of users, the data reflecting user gaming activity is growing exponentially. However, the nature of non-contractual online gaming is volatile, with multiple, perhaps latent, factors affecting customer churn. The dependencies and possible factors for churn can be discovered using Data Mining and Machine Learning methods, with varied success. Temporal character of non-contractual paid online gaming activity represents a challenge as well as a valuable source of potential business intelligence regarding churn. This Master Thesis explores one static and one temporal Machine Learning approach to analyze the multivariate time series reflecting online casual micro-gaming user activity. In focus of the thesis is the hypothesis that the temporal transitions are an important factor for churn prediction, and a method that takes them into account can be successfully used for this purpose. The thesis includes an account of methods selection, deep data exploration, theory behind the algorithms and their implementation as well as evaluation of the methods' performance and effectivity in demystifying the drivers of the non-contractual customers defection.

Acknowledgements

I would like to thank my family and close friends for being there for me during the challenging six months that I took to produce this body of work. Additionally, I would like to thank Professor Ralf Moeller for allowing me to write the thesis in his department and encouraging me to explore the topic that became of deep interest to me, and Rainer Marrone for advising me and patiently helping me arrive at the final draft. On the practical side of things, I would like to thank Martin Kavalar and Agnes Reissner for supporting my efforts and for the enormous experience I have achieved in working with the real-life ocean of data.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Data Mining and Machine Learning are currently widely used in many areas of human activity - medicine, business, finance, security and others. The scientific community has been actively developing approaches to improving knowledge discovery, including new classification methods, variants and data preprocessing techniques. The vast field for such research is mining, classifying and clustering of time-directed processes. Temporal aspect one of the most elusive yet inevitable characteristics of the todays important data sets. A list of examples employing sequential data includes gene classification, weather forecasting, financial markets analysis, speech recognition and many others. The specifics of each such dataset vary greatly, depending on the nature of the underlying process. The process can be a sequence of states, elements or symbols, as well as a numeric time series, defined as a series of measures taken with equal time intervals [1]. While the majority of research available today concentrates on the static methods, there has been a lot of scientific interest to creating and applying variants of temporal methods to model the time dependencies in the data. Markov- and hidden Markov Models, Dynamic Time Warping, Discrete Fourier Transform, statistical modeling techniques such as Autoregression have been approaching time-directed data in different domains and from different perspectives.

One of the areas where sequential data has become ubiquitous, is Customer Relationship Management. In the age of modern scalable online businesses, customer purchasing history and accompanying details are stored in databases in the form of timestamped records. Giga- and terabytes of data containing actions stretching years back in time, represent a potential raw material for statistical and Machine Learning approaches to finding specific patterns symptomatic to previously unknown segments of customer base. Predicting customer lifetime value and classification of customer defection are just few of the examples. Information, hidden in the properties of transitions between the states or events or the changes that the measured data incurs, frequencies and lengths of events, is the dimension that is missing from any static characteristics of the customers or aggregated values, considered in isolation.

The work that I have undertaken in scope of this thesis represents an exploration of the static and a temporal approaches to the problem of classifying and predicting customer churn in non-contractual setting. At the core of my effort is the hypothesis that the customer activity data that is temporal by nature, can be with greater success modeled using temporal approaches. Having selected an initial static method, Random Forest Classification, in order to consider the effectivity of the method in predicting behavioral pattern of churning users, I further take a different approach - modeling time series with mixture Markov models and analyze the challenges in using the method. This work builds upon my initial excursion into the world of Data Mining in CRM [2], where I have taken a look at the static pattern recognition and prediction in order to identify the potential dependencies caused by new product introduction by an online gaming company. This research contains more in-depth view into the data that represents the company's customer base activity, such as purchases and gaming engagement of the users, in the form of variable length multivariate time series, as well as implementation of the ensemble of temporal and static methods.

In the thesis I have used recent advances in Machine Learning and stochastic modeling techniques with the nature of the business problem and specifics of data in mind - a variant of mixture Markov model that allows for various-length sequences, as well as discretization techniques in order to map a multivariate time series to a univariate state transition sequence via medoids clustering for large-scale datasets. A series of other Machine Learning and statistical methods have been used in order to prepare and explore the data. The reader is taken through the end-to-end process, starting with existing research, via data exploration down to the details of implementation with statistical language `R`, returning to the meaning for the business goals.

This paper is structured as following: Chapter 2 gives a thorough overview of the latest available methods used for temporal data analysis and classification. In Chapter 3 I give the detailed description of the business problem (churn prediction in non-contractual setting), touch upon the origin of the dataset and perform exploration of the customer base data in order to understand its nature and peculiarities. Chapter 4 talks about the first churn prediction approach. In this Chapter I attempt to predict short-term churn risk by using a static classification method - Random Forest Classification - with the use of multiple static features which represent statistics of time dimension. Chapter 5 gives an in-detail account of the application of the temporal approach which involves discretization by medoids clustering, followed by mixture Markov modeling and concludes with a naive Bayes classification. In this Chapter, I address overcoming the complexities of the dataset - such as multivariate character and varying length of the sequences. At the end of the algorithm, the sequences are represented via probabilistic models and can be thus classified. Finally, Chapter 6 evaluates the performance of the methods described in Chapters 4 and 5, as well as looks at the challenges of each of the methods, followed by concluding final thoughts.

Chapter 2

Review of Current Research in Time Series Knowledge Mining

2.1 Introduction

Multiple approaches have been developed in the recent years for mining knowledge in temporal data constructs, including time series. The approaches differ by goal (such as clustering, sequential pattern mining, classification or forecasting), the nature of the data (words in the document, audio streams, protein sequences, temperature measurements), and by method of sequence representation. While classical approaches are maturing with appearance of the new variants, the novel approaches arise. The approaches to mining sequential data take their roots in methods originally developed for static data (such as Association Rule Mining [3] or k-means[4]). It is, however, clear that the directed nature of such data represents multiple layers of complexity not present in the static data, most prominent of which are the problems of multidimensionality and massive size of data, finding measure to compare sequences or calculate distance between them and representing the time series (or more precisely, their shapes) [5]. Thus, the initial methods developed for static data might be partially or completely unfit for mining sequential data, forcing development of the new variants and method ensembles which help overcome the challenges mentioned, among others.

Time series distinguish themselves among other kinds of sequences in that the ordering represents physical time axis and the points are numeric measurements taken with equal intervals in time [6]. Time series are often used to represent underlying physical processes or fluctuations of the financial markets, medical measurements, sociological observations and other aspects related to natural and human processes. The researchers generally consider time series in the context of the specific problem which requires either clustering or classifying the patterns reflected by the time series, determining missing points based on the existing points in the time series or predicting (forecasting) future values. Depending on the task, one or many time series might be considered.

2.2 Prediction and Classification

One of the most common and mature tasks related to time series is prediction or forecasting, surveyed in [7]. Prediction methods try to find future values of the time series based on the existing values by fitting a linear model based on minimizing objective error function. Such methods try to represent the time series via linear (or nonlinear) formula using specific coefficients. This task will not be described extensively due to the abundance of the subtopics.

Classification of the time series is another popular task in the area of Time Series Mining. While Clustering represents an unsupervised learning activity the goal of which is to find groupings, the groupings in Classification task are known a-priori, while the goal is to train a system that can distinguish between various classes based on their distinction features, and assign class labels to the new instances possessing the same set of features. Since Classification is one of the classical tasks in Data Mining and Machine Learning, there exist multiple methods and algorithms (From Naive Bayes Classifier and Decision Trees to Neural Networks and Support Vector Machines). However, the dimensionality curse of the time series data poses a question of representation of time series and their features in order to preprocess the time series for clustering, especially in cases of varying length time series. One of the most popular methods has been a 1NN Classifier with DTW [8]. In order to represent the time series, some researchers have used segmented representation (using information piecewise from the time series [9]), while others have used discriminative probabilistic models to represent the time series - such as Hidden Markov Models ([10]). Some of the regressive models mentioned in the previous sections have been used in the Classification task as well. It is clear that the appropriate representation of time series has been at the focus of the task of classification of time series: if a time series can indeed be represented as a combination of discriminative features, coefficients or model parameters, such collection can be used as an input to any of the existing proven classifiers. The approaches to finding such features have been proven to suffer from various problems - from impediments in robustness to overtraining [5].

Thus, classifying the time series can be represented as an ensemble (or a sequential implementation) of a group of methods.

2.3 Clustering and Other Methods

Clustering is finding latent groupings within the data. In the context of data being time series, the clustering task is complicated by the search for an appropriate similarity measure (or the somewhat opposite measure, distance between objects). Finding a similarity or even density clustering between time series cannot be defined as simply since each point in the time series needs to be compared with the corresponding point in another time series. Thus, in order to apply even the basic distance-based methods (such as k-means algorithm) to cluster time series, the distance/similarity measures need to be modified.

Univariate time series have been frequently approached with Dynamic Time Warping as a distance measure for clustering [11]. The measure reflects the similarity of time series and can determine sequences that are similar but sometimes "stretched" versions of another. The measure was first used for Image Processing/ Pattern Recognition. Multiple literature articles point out at the advantage of this distance measure used for clustering is that it works with the varied length time series. However, a recent article by Keogh et al. [12] seems to refute this statement. Despite of the method's popularity, it makes a series of assumptions on the kind of similarity of the time series in question. Specifically, if the frequency domain is of importance (i.e. the frequencies of the time series are an important parameter that distinguishes the classes of time series), DTW will not work well as a similarity measure, thus leaving the need for other distance measures. Once the distance is computed, a variety of methods can work with it similar to the static methods (such as SVM [13] or Self-Organizing Maps [14]).

Other ways to determine the similarity of time series are coefficients based on transformations from time to frequency domains - such as Discrete Fourier Transform [15] and Discrete Wavelet Transform [16]. The coefficients are achieved with the usage of complex number representations. The subsequent representations can be later used in order to compare sequences using the above mentioned methods.

As opposed to univariate time series, adding variables to be tracked (multivariate time series) and varying length of distances represent additional difficulty barriers that need to be addressed. Methods that apply probabilistic modeling to the time series set, such as Markov mixtures and Hidden Markov Models or variants of such, step away from the traditional time domain view and represent the time series in terms of transition probability between the states ([10] and [17]). Such methods allow to express varying length of the time series by building a variance to express conditional probability for each component. However, there are drawbacks to applying such methods, such as scaling problems while executing EM-algorithm and an assumption of the presence of the model within the data, while in in reality it might be not there [5]. The probabilistic models comprise a series of the most recent advances in the time series clustering.

Other time series mining task areas include segmenting the time series (essentially a dimensionality reduction problem), anomaly detection (or finding an unusual subsequence) and motif discovery (or finding a regularly repeating subsequence). The two latter tasks are closely connected and are used in such areas as security and biology. The area of motif discovery is currently in a state where a lot of questions are still unanswered.

2.4 Summary

Given the importance of time series data, the scientists have been working to extend the static Data Mining and Machine Learning methods in order to accommodate the challenges and peculiarities that arise in performing such tasks on time series. While there have been significant discoveries leading to wide applications of the temporal/time series methods in the recent years, finding the optimal way to preprocess the time series for usage with such tasks stays among the developing areas of Machine Learning and Statistics. The following Chapters of the thesis will illustrate such challenges on the example of a specific dataset related to customer behavior in online purchasing.

Chapter 3

Business Problem and Data Understanding

3.1 Business Description and CRM Challenges

The company providing the dataset for the described research is a small online gaming business, to be called Company in this thesis. The business model includes both freemium (free-to-play and premium features) and pay-to-play revenue streams (real-money, or cash games). There are others, like merchandise sales; however, they are out of scope for this research. The freemium revenue stream can be considered a contractual relationship with the customer: memberships that include additional features - such as tracking gaming statistics and possibility to form teams - and can be purchased for 1 month. To the opposite, the pay-to-play component of the business is non-contractual: the games occur in "micro"-format, and last on average up to several minutes. In order to be able to play such a game, the gamer deposits the amount of money that exceeds the minimal limit for playing one game . In case of losing the game, a certain amount gets deduced from the user account - and to the opposite, in case of winning the round, a specific amount gets added. It is important to mention that any user who has deposited money into a gaming account, can withdraw his holdings at any moment. Thus, the user is not contractually bound and can quit gaming any point in time - as soon as after one transaction.

Initial findings have shown that the pay-to-play gaming is more volatile to customer defection in Company's case - both due to its non-contractual settings and its user base. Some of the differences and initial findings are described in [2]. Thus, the thesis concentrates on the pay-to-play (to be referred to interchangeably as "paid","cash" or "purchasing") activity and customers.

3.1.1 Defining Purchasing and Free User Activity

If an online business offers both free and paid services, distinguishing between activity in the sense of purchasing, and all other activity is the base assumption in order to define any other principles or rules. In case of the Company, there are similar versions of the game offered in free, subscription, and cash gaming format (pay-to-play). Thus, a user can be active on the website while not active in the sense of purchasing activity. For the purpose of this experiment, the subscription services have been excluded from the cash activity.

Let us define *cash activity* as all user activity encompassing gaming that involves purchasing:

- deposit money into user gaming account
- play a singular cash game
- play a cash tournament
- withdraw money from user gaming account

All the other activities are considered non-purchasing activities. Examples include playing free and leagues games, tournaments, leaving a comment on the blog post or adding another player as a friend on the website social network. The user thus does not contribute to the profit by undertaking such activities despite of being active in the Web- or Mobile- version of the gaming platform. However, free and paid activities do not occur strictly in isolation. Rather, they are intertwined (for instance, a user can play both free and cash games - possibly with correlation). Free activity might in some cases be a precursor of the first payment, whereas in other cases such conversion might never materialize - and to the opposite, a user can start directly with the paid version. Thus, a concept of *conversion* arises (a recent business term defined as the change of status from free to paying customer). These aspects will be further considered during the task of Exploratory Data Analysis.

3.1.2 Defining Churn and Lifetime Customer Value in Non-Contractual Setting

The term *churn*(defection), when applied to a business's customer base, is defined as "leaving the supplier of services" [18]. Managing churn is one of the important components of Customer Relationship Management, especially in the scenarios when customer base growth is slowing down due to reaching the point of saturation for existing product.

In context of Online Gaming this can happen for a variety of reasons - some as simple as loss of motivation to spend money on the game due to frustration with frequent losses and as complex as latent substitute product (for example, a Football Championship season taking place or fluctuations in discretionary income of the gamer). Some types of micro-gaming - such as traditional German card games, the product offered by the Company - have additional nuances: in this case playing for real money (making micro-bets) represents a realistic approximation of the traditional game as it occurs when the players participate in person instead of online. This adds complexity in understanding the motives of the players.

Customer Lifetime Value is the predicted profit that specific user can bring to a company over the user's lifetime [19]. Churn, or leaving for good, is one of the vital components for such prediction: the predictive lifetime value is in direct correlation with the customer's lifetime duration, alongside with other factors - such as frequency and amount of purchases. In the Company's case, the profit per user can not be calculated in a straightforward way due to the payment model which makes the cash flow from the loser's player account to the winner's player account (with a small percentage directed as a fee towards the Company). Let us consider two examples. First one is the users who continuously make small payments into their accounts but lose the games frequently due to more skilled/experienced opponents or their own playing habits (such as increasing the bets, for example). They arguably represent the critical, yet volatile segment of the customer base. Once their account depletes after a gaming session, they are faced with the choice of depositing more money into their account or stopping the cash gaming altogether. The second example includes a customer who continuously wins the cash games and, while not contributing to the profit directly, perpetuates the profits by supporting the overall game-playing activity. These two examples illustrate how various segments, representing opposite purchasing/behavioral patterns, can be equally vital to the Company's profits. The role of churn prediction becomes more visible in this context. Using various modeling techniques, it is potentially possible to discern the behavioral/purchasing patterns that ultimately resolve in leaving for good as opposed to those that represent more stable, or loyal, customer segment.

In the context of no-binding relationship where users are free to switch between free and pay-to-play version of the product at any time, an assumption needs to be made regarding the period of inactivity of the user which results in being marked as churned(defector). While some researchers have approached this analytically, based on the means of interpurchase periods and frequencies of purchases [20], the consensus between the majority of researches has been that a period of inactivity can be fixed across the customer base according to the business rule. This particularly applies to the Company's customer base, due to certain disparities in activity such as the following example:

- a user who has become active, played games for cash intensely for a few days, then was inactive for some months; after that , a user becomes active again;

Such an example can be easily misclassified as churned by using the analytical model with such parameters as mean and variance, although the user is still active. Thus, following the Occams Razor principle, I have selected two constant values that define churn and risk of churn, based on comparable practices in the industry.

Definition 3.1. A customer has churned if he/she has not been active in the purchasing sense for 90 days.

Definition 3.2. A customer is at risk of churning if he/she has not been active in the purchasing sense for 30 days (4 weeks).

Using these fundamental definitions together with definition of the purchasing activity, I now have created a basic framework that can be used for churn exploration, classification and prediction. Exploratory data analysis, together with basic clustering of the customer base, will further uncover the data properties that might play a role in the above classification.

3.2 Exploratory Data Analysis

Exploratory Data Analysis is traditionally a first step in any Data Mining/Machine Learning Tasks, given a problem is defined in business language. The goal of this step is an in-depth look at the data available in order to grasp the nature of the data in the sense of it's statistical features - such as the distribution, presence of visible correlations or groupings, extremes, means ,medians and quantiles. Data exploration guides the scientist from the problem defined in business terms to the further steps of modeling and evaluation, which involve mathematical, statistical and Machine Learning concepts rather than business terms. The subtasks of this step involve evaluation of the relevant tables and fields in the database, extracting a sample by querying the database, cleansing and slicing the data to see it from various angles. The importance of this step is difficult to underestimate - after all, the data and insights provided by this step are to be used in the modeling experiments and ultimately provide a ground for hypotheses.

In order to understand what the customer base of the company represents, what features and characteristics (static and otherwise) can be useful in churn prediction task and finding the most profitable customers, I have performed exploratory data analysis augmented with some deeper insights such as static clustering by various dimensions and used graphical tools available in `R` statistical language in order to visualize the distribution and other statistical features of various one- and multidimensional datasets of interest. Static clustering step, although not traditionally a part of data exploration step , but rather a modeling exercise on its own, has been necessary in this context, since it furthers the understanding of the data and its challenges.

3.2.1 Finding Relevant Data

As a first step towards the solution, I have identified the tables of the PostgreSQL database that contain the information, pertaining to gaming/purchasing activity of the users:

- users
- payments

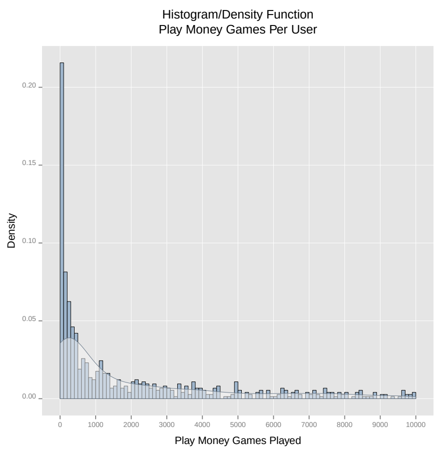

FIGURE 3.1: Histogram of Days Before Conversion

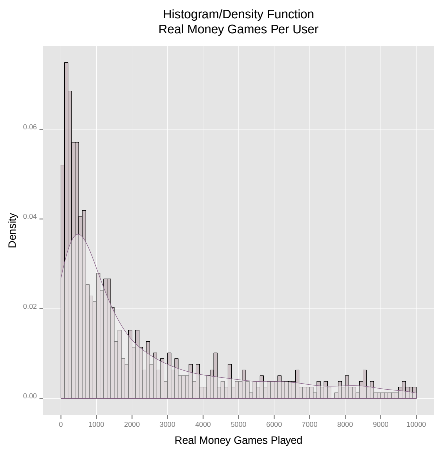

FIGURE 3.2: Histogram of Cash and Free Games Total per User

- daily_rankings, weekly_rankings, monthly_rankings, eternal_rankings
- games
- tournament_users, tournaments, tournament_games
- profiles
- aggregated view tables (game_stats, payments_stats)

While some of them contain static aggregated data (game_stats,payments_stats), others contain activity data with time information via timestamps (payments, daily_rankings, etc.). The above tables are the primary sources of data for the research described in this paper. The datasets have been extracted and cleansed using `PostgreSQL` and `R` languages.

3.2.2 Studying Distributions of Aggregate User Activity

3.2.2.1 Univariate Distribution Plots

As a first step towards understanding the data that I was presented with, I have explored the distributions of the aggregates per user for the paying customer base. The data has been extracted from the aggregated view tables, where each line represents totals corresponding to a *user ID*. The sample of 1000 users has been selected at random from over 14000 real-money gamers.

Figures 3.1, 3.2 show histograms and probability density functions of real-money and free total games played by each user. Histograms are scaled to 1 in order to address the distribution in relation to probability density function. As seen from the histograms, the data is highly concentrated near to zero and extremely dispersed towards greater values of number of games played. The data has distribution similar to negative binomial model. Figure 3.3 shows density function of actual data (real-money games) and density function of the fitted negative binomial distribution (with mean = 10264 and dispersion parameter 0.4).

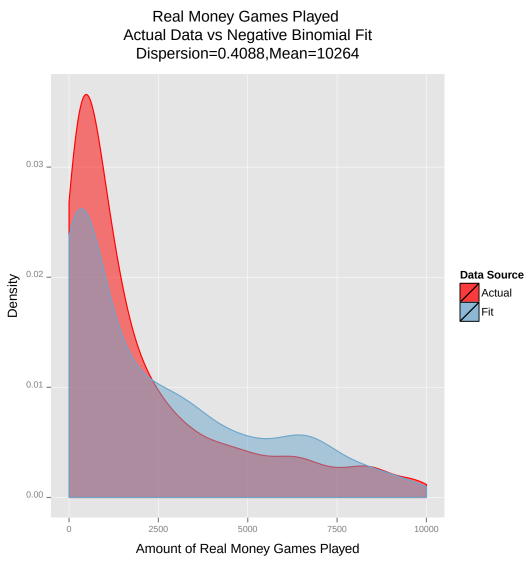

FIGURE 3.3: Real Money Games: Actual Data and Fitted Negative Binomial Distribution

Another important dimension is total *gain* over lifetime. It reflects how successful the user has been in winning or losing in real-money gaming. Figures 3.4 and 3.5 show distribution of gain across the total base and box plot of *gain* (the units are points and are kept to preserve magnitude of differences; actual numbers are hidden). There are several peaks in the histogram , which could signal a mixture of distributions. The box plot (another descriptive statistics tool) shows the first (25%) to third (0.75%) quartiles (Q1,Q3) and indicates dispersion degree inside the box, median and whiskers that extend to 1.5 IQR (where IQR=1.5(Q3-Q1)) . Points outside of this range are considered outliers.

Figures 3.4 ad 3.5 illustrate two properties of the *gain* dimension: the potential for containing groupings (multimodal frequency distribution) and presence of outliers (points that have very large positive or negative gain in comparison with the most users). For the Company, this means that there is a small number of exceptional

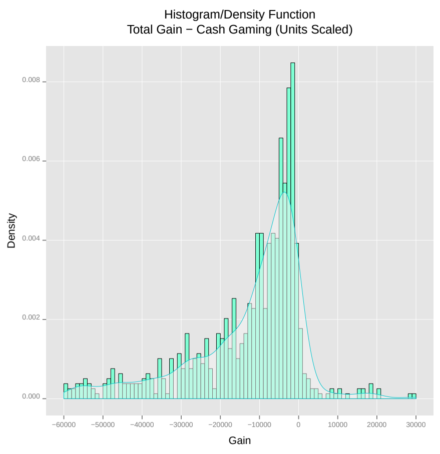

Figure 3.4: Distribution/Density Function - Gain (Real Money Gaming, Units Scaled)

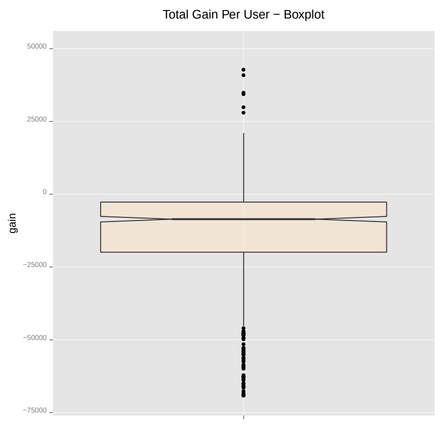

Figure 3.5: Box Plot - Gain (Real Money Gaming, Units Scaled)

winners or losers, whereas the whole group of users can potentially be segmented into separate groups characterized by typical gain values.

3.2.2.2 Pairwise Distribution and Correlation Analysis

In this subsection I explore the possibilities of visible pairwise dependencies between dimensions. The tools I use are scatterplots and Pearson correlation coefficient in order to assess the strength of the linear dependency. Exemplary pairwise scatter plots with corresponding correlation coefficients can be examined on Figures 3.6, 3.7. The dimensions reflect whether there is a visible linear dependency between how many games a user plays and gain on the former, and number of payments vs gain on the latter. As can be seen on the figures, correlation between real-money games played and the total gain of a user is rather insignificant - the points appear scattered in both positive and negative directions from the starting point (0,0). The negative correlation between the number of payments and total gain seems to be more significant (users with lower gain, have been also depositing money more frequently into their cash gaming accounts). Both pictures, however, support the statement that losing the game is a likely outcome for a player.

Additional scatterplots can be seen in Appendix A.

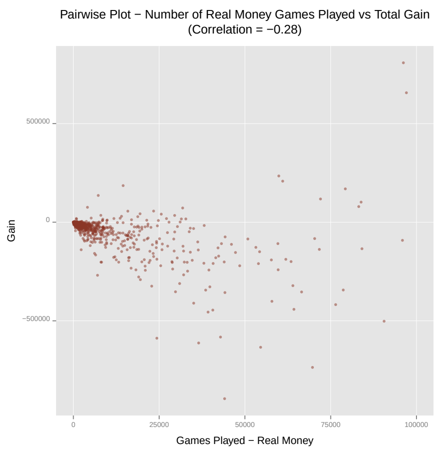

FIGURE 3.6: Pairwise Plot: Real Money Games Played vs Total Gain

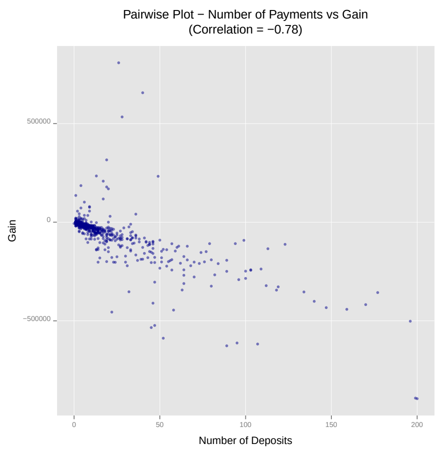

FIGURE 3.7: Pairwise Plot: Number of Payments vs Total Gain

3.2.3 Segmentation of the User Base by K-Medoids Clustering

While the analysis of distribution gives an initial picture of the user base according to spending/gaming frequencies and amounts, visualizations alone cannot spot the dependencies and data groupings which are not visible to human eye. This is especially true for multiple dimensions. Unsupervised Machine Learning techniques, such as Clustering (also referred to as Customer Segmentation in CRM context), can be helpful in detecting such hidden dependencies.

3.2.3.1 Preparation of Data for Clustering

As a preparation step for clustering I have trimmed the dataset down to a matrix of 6 dimensions representing aspects of purchasing activity by user. The resulting matrix contains 14562 rows and 6 columns. Each dimension of the matrix is discrete and numeric. However, since the dimensions vary greatly in magnitude (from single digits to 10^6), in order to avoid the bias towards dimensions with larger magnitude, I have centered the matrix by subtracting the column's mean from each element in the column and scaled it by dividing each element in the column by the column's standard deviation. A sample of the resulting matrix can be seen in Listing 3.1.

```
> head(clusterdata.scaled)
      real_money        gain  net_payments  num_payments  num_payouts  max_payment
[1,]  -0.2476961  -0.2291235     0.2651018     0.6001842    0.1553447   -0.2950053
```

```
[2,] -0.3907201  0.3443764   -0.3232097   -0.4632002  -0.3658761  -0.5158714
[3,] -0.3910392  0.3371998   -0.3232097   -0.4632002  -0.3658761  -0.5158714
[4,]  0.2745709 -0.5662532    0.2231206    2.7269530   4.3251110  -0.1477613
[5,] -0.3983001  0.3421823   -0.3501553   -0.4998687  -0.3658761  -0.9576036
[6,] -0.3091744  0.3241056   -0.3232097   -0.4632002  -0.3658761  -0.5158714
```

Listing 3.1: Scaled and Centered Clustering Data Matrix

As an extra step in attempt to reduce dimensions, I have applied the Principal Components Analysis technique to the scaled and centered data. I have used `princomp()` function in `R` which has returned the result displayed in Listing ??.

```
> summary(princomp(clusterdata.scaled))
Importance of components:
                        Comp.1 Comp.2 Comp.3  Comp.4  Comp.5    Comp.6
Standard deviation     1.8259 1.2511 0.7871 0.59493 0.35349 0.0454675
Proportion of Variance 0.5557 0.2609 0.1033 0.05899 0.02083 0.0003446
Cumulative Proportion  0.5557 0.8166 0.9198 0.97883 0.99966 1.0000000
```

Listing 3.2: Principal Component Analysis Results

As seen from Listing 3.2, the number of dimensions can indeed be reduced, since Component 1 through Component 4 represent a significant percentage of variance (98%). This supports the previous sections, in which some correlations have already been discovered. With such evidence, the clustering can now occur in 4 linearly uncorrelated dimensions.

3.2.3.2 Iterative K-Medoids Clustering

The matrix resulting from previous step can be taken as input into CLARA algorithm, first suggested by Kaufman and Rousseeuw in [21]. The algorithm acts similar to *K-Medoids* algorithm. The strength of the algorithm is its robustness: it has been designed specifically for clustering large volumes of data. Partitioning around Medoids (or K-Medoids) is a modification of K-means method where existing points are assumed as cluster centers (medoids) - the points that are located closest to all the points in the cluster - instead of calculating means of all coordinate dimensions. The advantage of the CLARA method over a regular K-Medoids clustering is that the majority of calculations are not performed on the whole dataset during each iteration, but rather on a specific sample. The steps of the iterative algorithm can be seen on Listing 3.3.

```
Select m samples of size n
Perform medoids clustering in each sample with k clusters.
Find the best result among all samples ( the one which provides the best
    partition according to the value of objective function)
Place the rest of the points into clusters by calculating distances between them
    and cluster centers from previous step.
```

Listing 3.3: Steps of CLARA Clustering Algorithm

In order to start clustering, an appropriate distance measure needs to be selected. Euclidian distance metric (square root of sum of squares) has proven to be less effective in multiple dimensions than alternative metrics, for example Manhattan metric (sum of differences of each dimension) [22]. Thus, I have chosen this metric to calculate the distance matrix for the algorithm.

I have performed clustering by using `clara()` function in R. The function allows to give parameters m, n, k as tuning parameters. After repeated experiments I have concluded that combination of m=20 and n=1500 allow for such representation of the initial matrix that further improvement in the quality of clustering is insignificant. In order to find appropriate k, number of clusters, I have performed the clustering iteratively and calculated average silhouette width (k=(3..20)),while keeping m,n fixed. Resulting silhouette widths can be seen on Figure 3.8. The clear downward trend can be seen as the number of clusters increases.

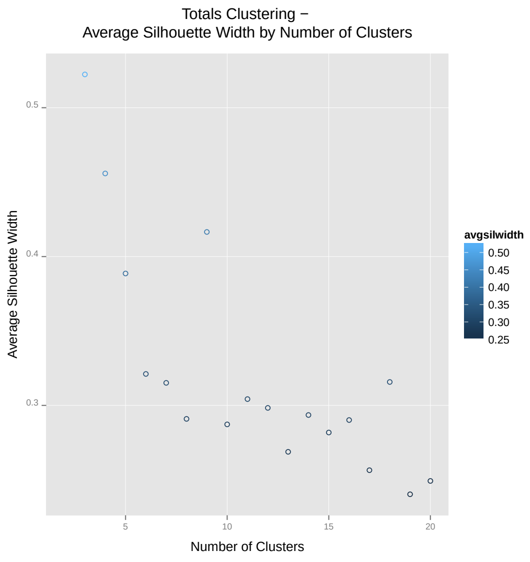

Figure 3.8: CLARA - Silhouette Widths by Number of Clusters

As a result of iterative clustering experiment, optimal number of clusters was found to be 3, with average silhouette width of 0.52. This might be signaling of the overall low clustering tendency of the data (or absence of natural data groupings with distinctive characteristics [23]). A look at the distributions of the dimensions suggests that the only available dimension with visible groupings is *gain*.

3.2.3.3 Totals Clustering Results - Visualizations

The `CLARA` clustering algorithm has identified three clusters of varying length; for each cluster, the row number of the medoid point has been identified. The results can be examined in the `R` output (Listing 3.4).

```
  >   static.clustering.res$clusinfo
      size max_diss   av_diss isolation
[1,]  3467 49.71398 3.4083732 16.010951
[2,] 10172 17.13344 0.6081742  5.518019
[3,]   953 71.10791 7.9336319  9.064758
```

Listing 3.4: Results of CLARA Clustering Algorithm

As is seen in Listing 3.4, the clusters vary greatly in size and in the maximal distance (dissimilarity) between their points.

With clustering results, it is possible to look at the data points in each cluster. Since there are 6 dimensions in the original dataset, direct approach to visualize the clusters of points is not necessary possible. However, it is possible to consider one or two dimensions at a time. Figures 3.9 - 3.14 illustrate the clustering results. Results show that there is clear division by gain (people who insure the biggest losses while playing vs medium vs very small losses, with occasionally users with positive gain). In the other dimensions, such as real money games played and number of payments, the division is less clear (each cluster has multiple outliers). The pairwise plots, in their turn, show that there is less correlation in the dimensions as the values acquire bigger magnitude (specifically, clusters 1 and 3 are more fanned-out and scattered across the coordinate plane, implying low correlation).

Figures 3.9 - 3.14 show that the customer base indeed can be split into three groups (least "losers" to biggest "losers"). It can be also seen that the separation is somewhat artificial in the sense that there are no visible distinct groupings that are well separated. However, it slices the customer base into three parts which

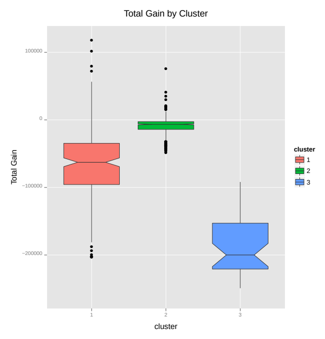

FIGURE 3.9: Box Plot - Gain by Cluster

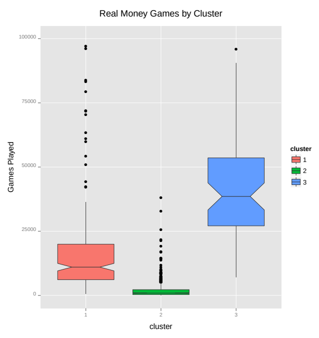

FIGURE 3.10: Box Plot - Real Money Games by Cluster

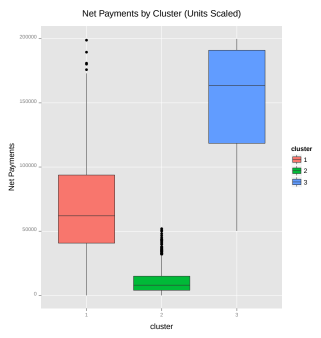

FIGURE 3.11: Box Plot - Net Payments by Cluster (Units Scaled)

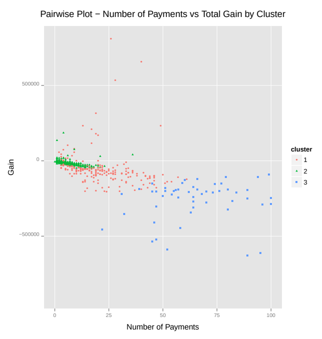

FIGURE 3.12: Number of Payments vs Gain by Cluster

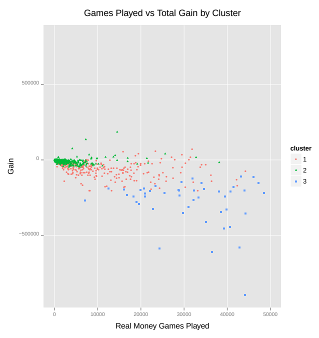

FIGURE 3.13: Number of Games vs Gain

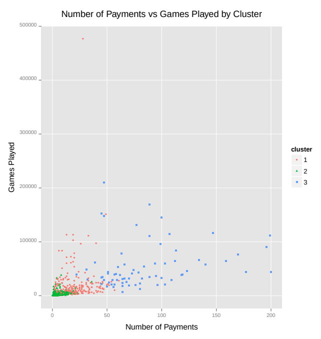

FIGURE 3.14: Number of Payments vs Games Played by Cluster

can possess certain characteristics (described above). What the clustering does not show at this point, is the peculiarities within each group, related to churn. In order to see which users have spent longer time playing (more loyal) and which ones have churned, we need to introduce new dimensions - the ones that have to do with the time axis.

3.2.4 Adding Temporal Variables

After review of the results of totals clustering, there are still unanswered questions regarding churn. As is seen from the previous section, there are three more or less distinct categories of users , split mostly by total gain over their lifetime as paying customers. There is a rather strong correlation with games played and negative gain (which implies that winning a game does not happen more often than losing) and same for net payments vs gain. However, is losing a lot over time a precursor of churn? In order to make a step towards understanding churn and its factors, some new variables need to be introduced,such as:

- duration of paying customer lifetime: difference (in days) between the first paid activity and the last paid activity
- duration of pre-conversion period: difference (in days) between the user activation date as a free gamer and start of payment activities
- class label - "Churned" and "Active".

These dimensions can be found using definition of churn by Definition 3.1, and the values of starting dates (activation, first payment) as well as dates of last purchasing activity. Resulting dimensions are discrete numeric values with time-units days. In order to examine the subset of paying customer s, I have reduced the total sample using the following business rules, which have been enhanced with the results of the previous exploratory steps:

1. a user is taken into sample if he/she made at least one deposit into gaming account
2. a user is taken into sample if he/she has been active during a period of 3 months as recently as 2012 (activity period) - this does not mean the user has been active for the whole duration of time
3. a user is taken into sample if he/she has converted to paying customer at least earlier than 6 months from the current date - this condition was introduced in order to be able to check the "churn" condition (3 months of inactivity in addition to previous 3 months of activity).

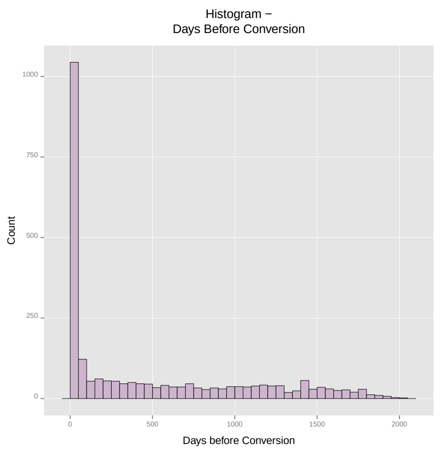

Figure 3.15: Histogram - Membership Days Before Conversion

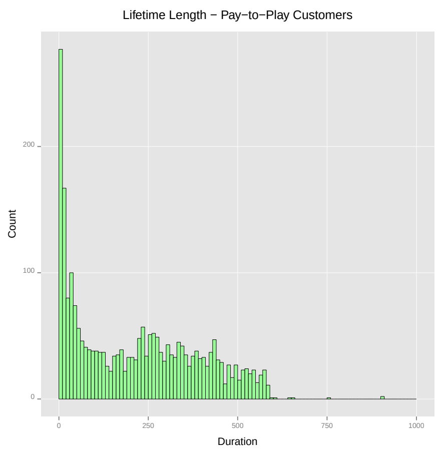

Figure 3.16: Membership Lengths

As seen on the Figure 3.15, most users convert almost immediately after becoming active on the website, reflecting the initial intention of using pay-to-play version of the game. There is a rapid slowdown in conversion following a certain period, let us call it ”settling in” period. This can also be read in terms of density or conditional probability (probability of converting given that a user has been already active on the website for a given number of days). Figure 3.16 exhibits two peaks in the histogram: there is a possibility of two rather distinct groups - which can be translated into the following statement: users have a tendency to either quit rather quickly, or stay and use paid services for an average of one year, with rarer exceptions of ”long-living” customers. Now, let us include the churn factor into the picture: Figures 3.17 and 3.18 show distinction by churn criteria. Here we see that still active (loyal) customers have indeed longer lifetimes than the churned customers, while length of period before conversion seems to not affect neither churn, nor length of paid activity period (we see almost uniform distribution). Thus, initial analysis shows that duration of before-conversion period, or ”age” of customer at the time of conversion itself does not seem to affect the further purchasing activity.

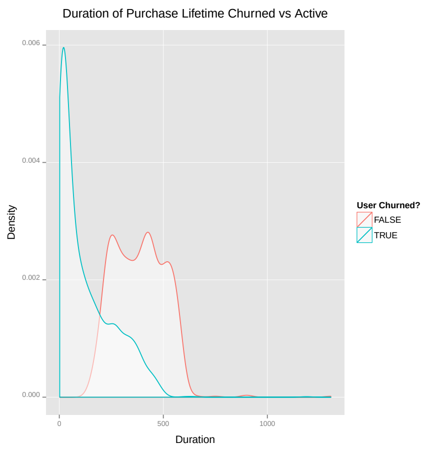

FIGURE 3.17: Density Plot - Paid Activity Period Duration (in days) by Class

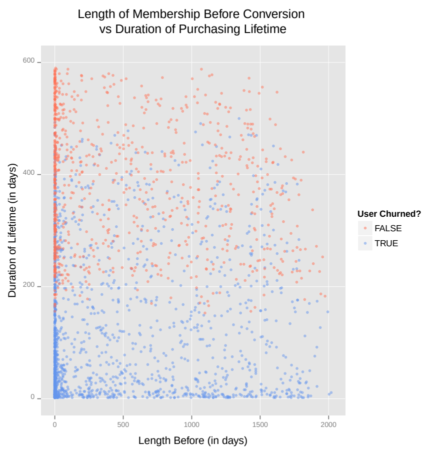

FIGURE 3.18: Length of Period Before Conversion vs Length of Paid Activity Period Duration

The two new dimensions introduced with the use of timestamp information from the transactional database tables, provide a new view on the customer activity. Augmented with the class labels (”Churned”, ”Active”), we can see the first correlations (length of paid activity is distinctly distributed across two classes). There are also some rejected hypotheses: pre-conversion period length seems to not affect the overall tendency to churn. This evaluation takes us closer to the introduction of time axis and heterogeneity of user activity across it.

Armed with the statistical features gathered in the previous sections, I have performed a visualization exercise in order to illustrate activity for individual users or user groupings over time dimension. The challenge of such visualization is the innate multidimensional character of the data, as well as choice of slicing and dicing the data and selecting a sample. I have chosen the dimensions of real money games and points games (total number played per user). Firstly, I have looked at Figure 3.19, which shows that there is a large spectrum of gaming attitudes (from avid gaming users to those whose paid activities carry rather accidental character). In order to get more insight, I have segmented the base and visualized cash versus free gaming (monthly) in each segment in the form of heat plot, where cash and free dimensions are juxtaposed in order to see increases and decreases in the weekly gaming activity. An example of such chart can be examined in Appendix A, where the reader can observe the variety of activity patterns within one cluster.

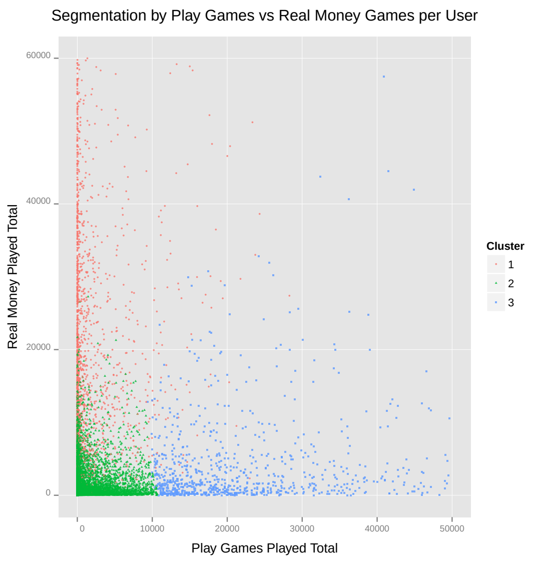

FIGURE 3.19: Segmentation of Paying Customers by Games Played

3.3 Conclusions and Outlooks for Churn Classification

Exploration of various aspects of purchasing behavior and their correlation has provided an initial overview on the general patterns and statistical properties of the data. However, it has still shed minimal light on the motivators and actions that lead to users' churn. Exploration has uncovered that within the total satistics data there are three visible clusters that combine heterogenous behaviors. The majority of considered dimensions (totals) have no strong correlation with each other. Moreover, the nature of the data is noisy which prevents from clean (isolated) clustering by standard methods (as seen on Figure 3.19). Most dimensions have negative binomial distribution pattern, reflecting the real-life pattern: most of the paying users are short-term customers, and are prone to leaving or settling for free version - correspondingly, they represent the most volatile segment. The visualization of gaming activity over time underlines the existence of additional information that cannot be described by static statistical descriptions, such as mean, variance of totals. The two methods applied and described in the next chapters represent an attempt to uncover this information by approaching the temporal character of the data in two different ways.

Chapter 4

Churn Prediction Static - Random Forest Classification

4.1 Theoretical Framework

This Chapter describes one of the approaches I have taken to predicting risk of churn, according to Definition 3.2. This approach follows a similar strategy as the Random Forest Classification approach that I have taken in my Project Work [2], with emphasis on additional features that interpret temporal information. This experiment is intended to test the assertion that static classification algorithm is capable of predicting with enough sufficiency, when encompassing observations regarding as many aspects about user's history and current activities as possible. In this chapter I also illustrate how the choice of modeling and response variables affects the outcome of Classification, and concentrate on the assumptions that are necessary to build the model.

Random Forest Classification is an Ensemble Classification Method which includes both voting strategy (the method selects a mode of the separate decision trees results) and bagging strategy (sampling with replacement) for predictive features at each tree split. More theoretical information on this method can be found in [2]. I have selected this method due to proven efficiency in many areas, as well as possibility to compare predictive weight (or importance) of individual variables. Additionally, this method avoids building parametric models, deals well with the cases of conditionally dependent predictive variables ([24]) and performs the classification ”inside the box” - a more-or less automatic process.

The selection of features for the classifier was partially inspired by the framework built in [25] and is using a heuristic that extracts static values such as mean and variance and positive or negative change in amounts over time. It works with abstract representations of the time series and subsequences, where subsequence is defined as 4.1.

Definition 4.1. *Subsequence is a portion of time series of lesser or equal size, which preserves the ordering of the sequences states.*

While the mentioned article describes early classification of the time series, I have modified it in order to predict the the user's propensity to churn - in other words, to predict the near future activity - by using the most recent activity in combination with total activity, with addition to some static characteristics of the users. Thus, the method will consider the very end of the sequences representing user activity history rather than the beginning. The user lifetime time series per se do not appear in any of the modeling steps. Rather, they are represented by global and local properties, which in turn try to preserve the main statistical characteristics, such as, for example, positive or negative trends, variance, mean.

It is worth mentioning that since this is used to show initial results with basis for improvements and is not the focal point of this thesis, I have left local optimizations of the data modification techniques out of scope. The emphasis of this chapter is to explore a general approach and draw conclusions of its applicability to the problem at hand.

4.2 Data Selection and Preparation

The task of data selection and preparation consists of defining the potential predictive features (dimensions) to be used in churn classification and the sample.The challenge of the task consisted of, among other factors, defining the static predictor variables characteristic of certain time periods by modifying temporal variables based on their statistical features. This is done in attempt to preserve maximum amount of temporal information.

For the experiment, I have selected a sample of 2400 user_IDs who were active before the start of test period (to be described in next section). The ID's themselves were not used in the exercise in order to preserve anonymity. In order to build the

Field	**Type**	**Extraction Method**
total average gain	numeric	direct
membership length	numeric	direct
last week - tournaments played	boolean	direct
currently positive gain real money	numeric	derived
currently positive gain play money	numeric	derived
currently average games per week (real money)	numeric	derived
currently average games per week (play money)	numeric	derived
gender	categorical	direct
large intervals between activities	boolean	derived
assertive gaming style (knockings)	numeric	derived
difference between leading and lead moves	numeric	derived
total average games per week	numeric	derived
total variance in games per week	numeric	derived
number of friends	numeric	direct
final score at the end of current period	numeric	direct
average gain during current period	numeric	derived
sporadic gaming pattern during current period	numeric	derived

TABLE 4.1: Predictive Dimensions

predictor values based on the temporal knowledge, I have split the features into global characteristics (totals across the whole lifetime), current (most recent four weeks of activity), very current (most recent week of activity) and future (predicted activity or class labels) periods. After as many features as possible (a total of 17 dimensions) were extracted from the data storage based on the experiential knowledge. As seen in the Table 4.1, the values have been achieved in two ways: directly extracted or achieved via modification (mostly a linear combination or statistical feature of a group of timestamped data). Dimensions include some features described in Chapter2,as well as variables which have been identified as having potential to affect user's churn (an example of such is *tournaments played*, a variable that I have studied in [2]). Lastly, I have tapped into the gaming style of users, collecting statistics of their assertiveness and risk-taking in the game. For timestamped data I have taken weekly frequency of measurements. Predictive features represent variety of data types (boolean, categorical and numeric).

In order to derive the statistics of user activity, I have relied on selecting the characteristics that could potentially be relevant to the business case. Some of the dimensions have been discretized by splitting the axis into intervals (an example is *large intervals between activities* which is essentially the opposite to the frequency parameter; in this case, large intervals represent an interval that is larger than 30 days). An example of usage of business expertise is the use of trend information for the *current gain* instead of using the actual value: thus a spectrum of values representing gain has been reduced to a boolean variable that reflects whether user has been mostly losing or mostly winning during the current period.

Experiment #	Hypothesis	Class Label
1	user has not played real money during test period	Yes (1)/ No (0)
2	user has not played real money during first week of test period	Yes (1)/ No (0)
3	user not played real money some or all of the weeks during test period	Yes (1)/ No (0)
4	user has played some of the weeks of test period, but not all	Yes (1)/ No (0)
5	user has stopped playing real money in the second half of the month	Yes (1)/ No (0)
6	user has switched to free gaming	Yes (1)/ No (0)

Table 4.2: Predictions/ Class Labels

The resulting sample has been split into training and test subsample by taking random samples one half each of the total sample. Code for dimensions derivation can be examined in Appendix B.

4.3 Defining Response Variables

The second task of building a Random Forest Classifier is to assign class labels to each row of the prediction matrix. For this exercise, I have selected six scenarios for the test period, each with the boolean outcome (1 or 0). For the purpose of the first iteration of the experiment, I have selected a period of one month (according to Definition 3.2). The scenarios test various outcomes for the user in the first month following predictive period. Table 4.2 illustrates the scenarios for classification, where the class label is essentially a prediction (Yes or No) of the question in the left column, or hypothesis. I have gone from strict hypotheses (such as "a user has churned during the test period") to semantically relaxing the conditions ("a user has been inactive some OR all of the weeks"). The hypotheses in question pertain mostly to the gaming activity and represent classes of unequal size.

The classification experiments are an attempt to understand short-term predictive power of the approach as well as efficacy of the dimensions selected.

4.4 Modeling Process and Results

After completing the groundwork described in previous sections, I have performed iterative Random Forest Classification exercise - by changing the tuning parameters such as number of trees in the forest (I have settled at setting the n=20000, which significantly exceeds the number of rows in the sample, and setting number of features being sampled at each split at m=4). Since sampling is done with replacement, each tree gets a chance to concentrate on certain features for classification. I omit the details of implementation in this paper due to having already described them in precision in my previous work ([2]).

In order to train the classifiers, I have used the `randomForest()` function, the details of implementation of which can be studied in [26]

Iterative modeling with large forests (20000 trees) has yielded positive results in some of the experiments, while the others displayed a significant degree of bias, rejecting the most hypotheses defined in Table 4.2. Listing 4.1 illustrates the `R` output of class error for each class label prediction and each experiment. It is seen that most of the experiments have displayed a great bias (or degree of misclassification) .

The two experiments that have displayed the lowest average misclassification rate are experiments 3 and 4 (4.2), with experiment 4 being the one with the most evenly distributed misclassification rate. Due to relative success of these experiments which implies greater validity of selected dimensions, it is worth to look at the relative importance of the predictive variables according to the two information gain criteria (MDA and Gini) [26]. Figure 4.1 shows such criteria for a list of 17 parameters, sorted from highest to lowest. It can be seen that such criteria as games played per week and final score at the end of the predictor period have higher predictive power, whereas number of friends and positive/negative trends do not contribute significant information to the classification.

The results of the experiments have shown initial insights into short term prediction, or prediction of customers at risk of churning. The partial effectivity of the approach and variable selection provides a tentative benchmark for comparison with application of other methods. Chapter 6 will provide a more complete evaluation of the results of applying this method alongside examining the results of using the temporal method, described in Chapter 5.

```
> experiment1$confusion
     0  1 class.error
0 1001 30  0.02909796
1  100 55  0.64516129
> experiment2$confusion
    0   1 class.error
0 880  58  0.06183369
1 126 122  0.50806452
> experiment3$confusion
    0   1 class.error
0 275 178    0.392936
1 117 616    0.159618
> experiment4$confusion
    0   1 class.error
0 401 207   0.3404605
1 195 383   0.3373702
> experiment5$confusion
    0   1 class.error
0 646 113   0.1488801
1 309 118   0.7236534
> experiment6$confusion
    0   1 class.error
0 647 112   0.1475626
1 311 116   0.7283372
```

LISTING 4.1: Classification Results - Random Forest

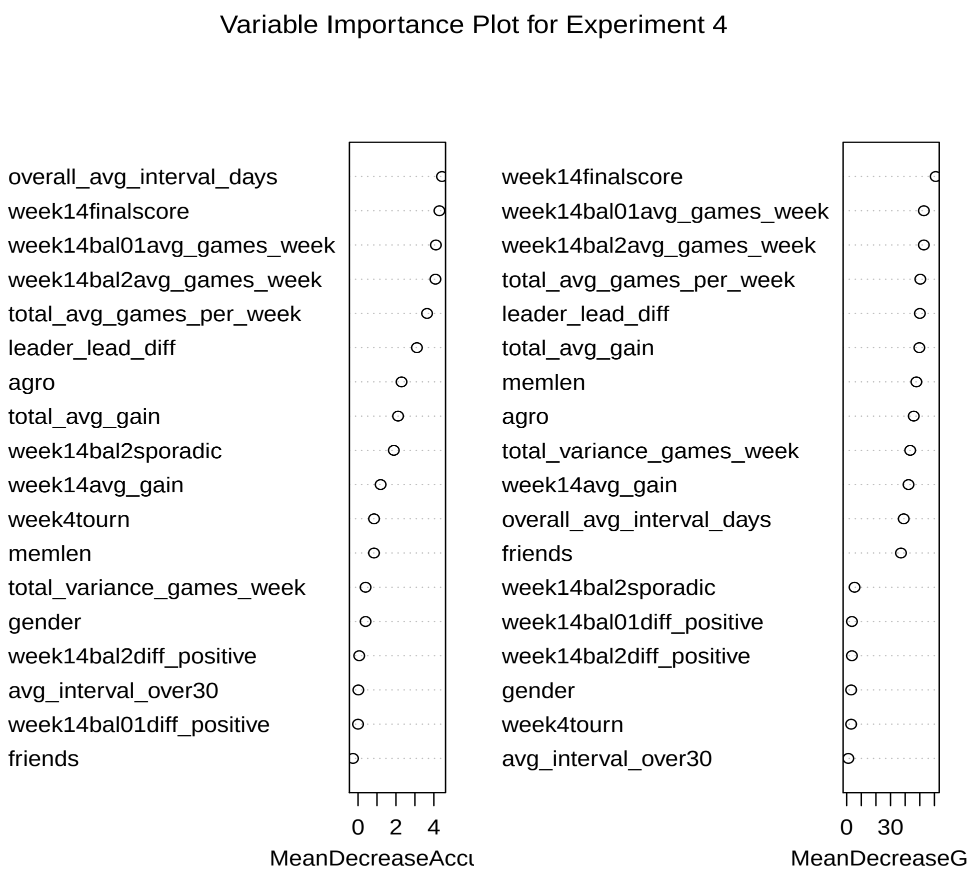

FIGURE 4.1: Variable Importance Plot for Experiment 4

Chapter 5

Temporal Approach - Mixture Markov Modeling

5.1 Introduction

This Chapter describes the implementation of an approach to classification of churning users using the temporal representation of the the user's lifetime real-money gaming activity. The effort stands in the focus of my thesis and tests the hypothesis that considering a time series (or a sequence) as a temporally ordered collection of observations rather than a collection of its basic statistical features helps to uncover additional information, hidden in the state transitions across time. This approach represents a combination of unsupervised machine learning methods and probabilistic modeling that help modify, preprocess and index the multivariate time series which represent the users' activity stories. My algorithm follows the general approach to a similar problem, described by Yang et al [17], with added modifications of the algorithm in order to add robustness and account for the data specifics. The authors of the article propose the solution for churn classification in non-contractual setting by implementing the sequence of steps mentioned in Listing 5.1

```
Discretize the multivariate sequences in order to transform them into univariate
    state transitions
Cluster the sequences by performing Markov mixture modeling
Classify/Index the sequences
```

Listing 5.1: Time Series Classification/Indexing Algorithm Pseudocode

The reason I have selected the algorithm proposed in [17] is the similarity of the business case of the Company with Telecom industry: churn of non-contractual customers. The customer lifetime in both cases can be represented via multivariate time series of varying lengths (as we have seen earlier in the histogram). While similar on a general level, the transactional data of the Company has peculiarities of its own (such as sparsity and the range of sequence lengths) and thus requires certain modifications in the approach in order to achieve a similar success.

5.2 Theoretical Basis: Markov Property,First Order Markov Chains, Mixtures

The algorithm, outlined by Listing 5.1, is based on the assumption that the sequences are stochastic processes that can be presented as the Markov chains. The Markov property of a sequence is defined in the literature the following way [?]:

Definition 5.1. A stochastic process has the Markov property if the conditional probability distribution of future states of the process depends only upon the present state, not on the sequence of events that preceded it.

Definition 5.1 is also referred to as memoryless property. As it follows form the name, the process ”forgets” the previous states. This important property has allowed to develop a probabilistic framework for modeling sequences: the sequence is represented as an instance of Markov chain, where the chain is defined by the transition probability distribution from each state to the new one. This touches on another fundamental concept, that of *random variable*, which is formally defined as [?]:

Definition 5.2. A random variable is a variable whose value is subject to variations due to chance.

Random variable is a powerful concept that allows to define and model uncertainty. Armed with this definition, we are now able to apply it, together with Markov property defined earlier, in order to formally define the concept of a first order Markov chain [27],pp. 606-607:

Definition 5.3. A Markov chain is a sequence of random variables $X_1, X_2, ..X_n$ with the Markov property, or $P(X_{n+1} = x|X_1 = x_1, X_2 = x_2, ...X_n = x_n) = P(X_{n+1} = x|X_n = x_n)$.

Definition 5.3 represents an abstraction of a sequence using probabilistic notation. Here, states can represent any objects (such as symbols or collections of values). Translated to multivariate time series, each state of such time series represents a collection of measurements at a given time. Thus, such framework can be also used in order to model multivariate time series. Figure 5.1 depicts an example of Markov chain of length k.

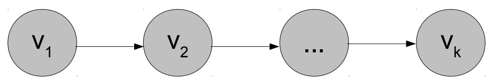

FIGURE 5.1: Markov Chain

So, how is Definition 5.3 useful to classification of times series for churn classification? In order to answer this question, one more component is needed, and that is one of mixture modeling. Given a collection of time series, it is possible to model(parametrize) such collection as a mixture of probability distributions. We have seen such univariate distributions when we explored the *gain* variable. It is, however, possible to represent other (multivariate and even temporal) objects as such mixture. Thus, mixture modeling is a probabilistic approach to unsupervised learning, or finding groups within a collection. Going back to the question of churn in non-contractual setting, given two classes - churned and non-churned - each of the sub collection can be modeled by means of mixture modeling (with a defined number of components). The multistep algorithm defined in previous section relies on such modeling in order to overcome the issue of unequal lengths of the time series.

Final component in the workings of the classification algorithm is the concept of Bayesian probability, an interpretation of probability that involves the assumptions of prior beliefs about the random variable and its outcomes, which later get updated with the incoming relevant evidence [**?**]. Definitions 5.1-5.3 lay out the basic framework for the classification process described in the next sections.

5.3 Data Understanding. Determining the Temporal Context

As mentioned and proven in the previous Chapters, the choice of dimensions or features is of significance to the outcome of the classification. In this section I describe the motivation of the representation of the data, as well as the process of variables selection/acquisition.

The motivation of representing the user lifetimes as multivariate sequences stems from the natural representation of the transactional data in the Company's database: gaming activity and payments history is stored in tables that represent aggregated measurements with daily/weekly/monthly frequencies. It is also possible to acquire information with second precision. Thus, it is evident that the user lifetime can be represented with a series of such measurements. There are, however, some questions to be answered in order to acquire a sample to perform modeling and classification:

- which users to include into the sample
- how to define the time axis (there are alternative representations possible)
- which specific measurements to include into the dataset
- assigning class labels

The following subsections address these four issues.

5.3.1 Sample Selection

The user base of the Company is extremely heterogeneous. However, for the purpose of training a churn classifier and noise reduction, not all of the users can be considered. Thus, certain restrictions had to be included into the SQL queries according to the business rules that could help weed out noise at the early stage and avoid reduced effectivity of the classifier and retraining.

The following business rules have been used in order to select the manageable, yet representative sample of user ID's :

- include users that have started playing cash at least 8 months before the time of the experiment (in order to allow for significant confidence in churn - 5 months of activity)
- include users that have started playing paid version of the game after 2012-01-01 (in order to address the effect of the new paid offering introduced in 2012)
- exclude "accidental" users with 0 payments
- exclude "accidental" users who only were active less than a certain amount of days in cash gaming (1 day)
- exclude test ID's since they do not correspond to regular gamers

Molding such a sample yielded a total of 2492 user IDs which were used in order to query the database for transactional (temporal) data.

5.3.2 Selecting the Time Axis and Time Units

Selection of time axis and frequency of measurements for time series definition is one of the success factors in the classification. There are two possibilities to define the time axis:

1. use the number of games played as a relative time unit
2. the actual time unit (such as day, hour etc.)

After careful consideration of these two options, I have decided to go with option 2 - to use the actual time units for measurement. The motivation for selecting the actual time units was the natural correspondence that such definition provides with the actual data stored in the database, thus providing more consistency. Another reason for the selection of Option 2 was that by selecting number of games played as the time-axis and measuring all the other dimensions relative to this axis removes the number of games played from the tracking uniformly with all other dimensions.

Having set up the time axis, the second decision concerns selecting the frequency of measurements used for the time series. While using high-level measurements (like months) can improve algorithm performance by resulting in shorter time series, there is a great potential loss of information of state transition details due to aggregation. On the other hand, taking measurements that are too detailed (such as per hour basis), can lead to overfitting of the model due to noise and significantly complicate/slow down the algorithm. Considering the lengthy - years magnitude - histories of some users and micro-gaming format of the game, and taking into account the results of the experiment in the previous Chapter, it is reasonable to assume that daily measurements are able to reflect enough information regarding the user patterns and yet be manageable in size (length up to hundreds of days).

The final aspect of data preparation step is selecting the start date of the sequence: while the users might have performed gaming activities before becoming a paying customer, the time series starting day (or first measurement) is considered the day when any of the paid activities (defined in Chapter 3) have occurred for the first time, or conversion. This denotes the beginning of the paying customer lifetime. Likewise, the end of the customer lifetime is denoted by the last purchasing-related activity. First observations show that a typical starting action is depositing money into user account.

It should be noticed that the resulting sequences do not yet represent the completely consistent multivariate time series since they only include the days on which activities have occurred, but do not deal with the inactive days yet. This will be addressed in the next section.

5.3.3 Selecting Features/Dimensions of the Time Series and Assigning Class Labels

Among many possible tracked activities that are recorded in the database tables, the most relevant ones to the paid activities . The dimensions chosen for the time series are based on the preceding research and are trimmed down to six vital dimensions (displayed in table 5.1).

The selected dimensions were extracted from various tables containing daily measurements, and combined into a large dataset of transactions (sorted by user ID). The resulting dataset was augmented in order to represent days of inactivity with 0's (in consistency with "zero-padding" approach) [28]. The resulting sparse dataset combines all the multivariate time series and represents a sort of

Dimension	**Type**
real money games played	discrete numeric
average gain per game (real money)	discrete numeric
current score (real money games)	discrete numeric
payment amount	discrete numeric
tournament participation	boolean
free/premium games played	discrete numeric

TABLE 5.1: Time Series Dimensions

transactional database organized by User ID and preserving the order/frequency of measurements for each user. The lengths of sequences vary from just 2 days to several hundred. An exemplary extract from the fundamental dataset to be used for classification, (named *lifedata*) can be seen on Listing 5.2.

```
> head(lifedata,20)
   user_id games2 current_score    gain   payment_amount tournament games01
1        a     37          4582      -1             5000          1      14
2        a     65          4262      -6                0          0       0
3        a    328          8530      -2             5000          0       0
4        a    259          4581     -15                0          0       0
5        a    181          1742     -17                0          0       0
6        a    242          6232     -18            10000          0       0
7        a    187          5099      -7                0          0       0
8        a     65          2943     -38                0          0       0
9        b     55          6371       8             6000          0       0
10       b     35          6111      -2                0          0       0
11       b      0             0       0                0          0       0
12       b      0             0       0                0          0       0
13       b      0             0       0                0          0       0
14       b      0             0       0                0          0       0
15       b      0             0       0                0          0       0
16       b      0             0       0                0          0       0
17       b      0             0       0                0          0       0
18       b      0             0       0                0          0       0
19       b      0             0       0                0          0       0
20       b      0             0       0                0          0       0
```

LISTING 5.2: Classification Dataset

The purpose of this method application is longer-term churn prediction according to Definition 3.1. Thus, each user ID has been labeled 1 for users who have churned (inactive for equal or more than 90 days) , and 0 for those still active (or inactive less than 90 days). There are 1634 users who have churned and 858 who are considered still active.

5.4 Time Series Classification

This section tells an in-depth account of the classification process, its implementation and challenges related to it. As defined in Listing 5.1, there are three main steps in the algorithm, which are driven by the goal of preprocessing the multivariate time series of unequal length length into standardized objects which can be later classified using one of the available classification methods. Each step receives careful consideration and is supported with relevant theoretical concepts.

5.4.1 Inferring States via Clustering

The multivariate sequences sample, acquired in the previous step, provides a dataset for experiment. An example of one such user time series can be examined on Figure 5.2. However, in order to be able to represent the time series as a sequence of states, the set of states needs to be defined. In case of the *lifedata* dataset, each record presents a certain combination of measurements 6 measurements. Although there are no truly continuous dimensions, so no discretization is required for any dimension, the ranges of some of the dimensions are of magnitude 10e+5, which does not allow a possibility of manually assigning a specific state to each combination. Thus , a different approach of grouping combinations of measurements is needed. A natural choice is applying a clustering technique in order to determine groupings within measurements records.

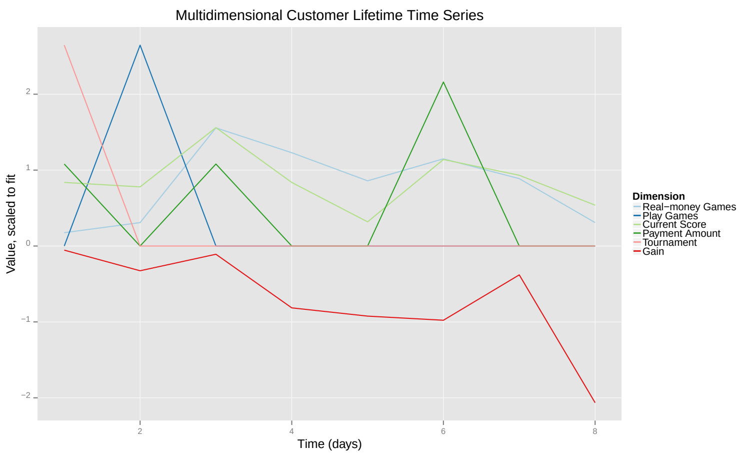

FIGURE 5.2: An Example of a Customer Lifetime as Time Series

Due to a large volume of data (a total of 535334 transactional records among all users), a method must be used that can effectively handle such great amount of data. CLARA algorithm, described in detail in previous Chapters, is a good candidate for clustering large applications and thus was selected to perform this task.

Arguably the most important aspect of clustering the transactional data is selecting the optimal number of clusters. There needs to be as much as possible balance between sacrificing the quality of clusters which will be used in the next steps of the algorithm, and keeping their number as low as possible. The reason for the first constraint is that assigning the point (a combination of measurements) to the cluster that does not correctly represent it might negatively affect the classification outcome. The latter condition is important in order to satisfy the performance requirement: the number of clusters (states) is directly related to the size of transition matrix (n^2 for n states). Since the computational power of the machine is limited by its architecture - in my case it is 64Bit and memory limitations, as well as limitations within the statistical software R.

As a final sample preparation step, I have removed user IDs, converted the boolean dimension into discrete numeric and scaled the resulting 6-column matrix similarly to the clustering in Chapter 2. Listing 5.3 illustrates the resulting sample.

```
> head(lifedata.staticclustering)
  games2 current_score.x gain.x payment_amount tournament games01
1     37            4582     -1           5000          1      14
2     65            4262     -6              0          0       0
3    328            8530     -2           5000          0       0
4    259            4581    -15              0          0       0
5    181            1742    -17              0          0       0
6    242            6232    -18          10000          0       0
```

Listing 5.3: Scaled Clustering Matrix

After iterative clustering attempts with varying sample sizes and number of clusters, the optimal number of clusters was found to be 17 - a minimum available to create discernible states, with average silhouette width = .62. This indicates a reasonably good separation of data. As a result of clustering via medoids, we now have a collection of 17 states which represent the separable combinations of measurements. Each record in the transactional dataset can be labeled with the cluster label it was placed in. The collection of multivariate time series can now be presented via sequences of states, drawn from a distribution of the resulting 17 states. The state labels are the same as cluster numbers, i.e. S={1,...,17} is the set of possible states. A sample of the resulting sequences is seen on Listing 5.4.

```
> sample(lifedata.univar,3)

[[1]]
  [1]  7  7  7  8  7  6  2  6  7  8  4  2  4  4  4  4  4  2  2  2  8  8  6  7  6
     8  7  4  4  8  7  6  4  2  2  2  2  8  8  8
 [41]  8  8  8  7  8  8 10 10  6  4  7  2  8  2  7  2  6  8  8  8  8  8  4  8  6
     4  8  2  7  8  7  8  8  8  8  8  8  8  8  8
 [81]  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  7
     4  8  8  8  8  8  8  8  8  8  8  8  8  8  8
[121]  8  8  8  8  8  8  6  2  8  8  4  7  8  8  4  8  8  8  8  4  2  2  2  8  2
     2  2  6  8  8  8  2  4  3  8  2  8  2  8  8
[161]  8  8  8  8  8  8  8  8  8 12 13  8  8 12  8 13  8  8  8  8  8  8  8  8  8
     8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
[201]  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  7  2  8 14  8 13  8  8 12 14
    12  8  8 14 12 14 13  8 14 12 12 12 15 15  8
[241] 12 13 14 12 13  2 13  8  8  4  4  8  2 14

[[2]]
 [1]  6  8  8  8  8  8  8  8  8  8  8  8  8  6  2  2  8  8  8  2  8  8  8  2  8
    2  8  8 10 10  2  8  8  8  6  8  8  8  2 10  8
[42]  8 13  8  8  6

[[3]]
  [1]  5  4  4  2  8  7  2  4  2  2  2  4  4  2  2  2  2  6  8  8  2  4  2 12 13
     8  8  8  8  8  8  8  8  8  8  8 13  8 13  8
 [41]  8 13 12 13  2 13  8  8 13 13  8  8 14 13  8  3  4  2  4 10  8  8  4  6 14
    12 13  8  8  8  8  8  8  8  8  8 13  8 12  8
 [81]  8  8  8 13  8  8  8  8  8 13  8  8 13  8  8  8  8  8  8  7  2  2 10  2  4
     2  6 12  8  8  8 13 12  8  8  8  8  8  8 12
[121] 13 13 13  8  8 13 12  2  2  8  2  8  2 12 13 13  8  8  8 13  8 13  7 10  8
     6  2  2  6 12  8 13  8  8 13 13 13  8 13  8
[161]  8  8  8  8  8  8  8 12  8 13 13  7  2  8  4 12 12  8  8  2  6
```

Listing 5.4: Sample of Derived Univariate Sequences

5.5 Markov Mixture Modeling

5.5.1 Defining a Markov Mixture Model

In order to be able to cluster the sequences resulting from the previous step of the algorithm, the parameterical mixture model needs to be defined in the context of Markov chains. Informally, such model is a collection of latent (or hidden) components that are each defined by their parameter values and their marginal weight in the model. It is easy to notice how such structure correlates to the concept of clustering. Formally, a mixture model is given in terms of discrete random variable by Definition 5.4 [27].

Definition 5.4. A parametric mixture model is a probabilistic model given by the formula:

$$p(x|\Theta) = \sum_{k=1}^{K} \pi_k p(x|c_k, \Theta), \tag{5.1}$$

where x is a random variable with discrete distribution, and $\pi_k, k = 1..K$ is the kth component of the model, and $p(x|c_k, \Theta)$ is the probability of the random variable given the parameters of model component c_k (*conditional probability distribution*), π_k is the marginal probability of the kth component (cluster).

The $\pi_k = p(c_k = 1|\Theta)$ satisfy the following conditions: $0 \leq \pi_k \leq 1$ and $\sum_k \pi_k = 1$ and are called *mixing coefficients.* Thus, Equation 5.2 represents the summarization of joint probability distribution of random variable x over all clusters (components c_k) in the model, also called *marginal distribution* of x.

Definition 5.4 sets up a probabilistic framework for modeling the outcome of a random variable. However, for the case described in this thesis (Markov chains of varying length), there is one piece missing and needs to be defined - the conditional probability $p(x|c_k)$. In order to define that, the probabilistic notation of Markov chain needs to be employed.

The next step is to define the components of a mixture, given that a discrete random variable is an instance of a Markov chain, and length of this chain needs to be variable. Using the memoriless property of the first-order Markov chain representation, conditional probability distribution can be defined in terms of start and transitional probability given by discrete distributions. It can be done via the following two steps: firstly, present each cluster as a collection of first-order Markov Chains, each described using sufficient statistics[17]:

Definition 5.5. Each Markov mixture model component is defined by $\overrightarrow{v}$ - start probability vector, and T_p - transition probability matrix:

$$\overrightarrow{v}_p = \begin{pmatrix} a_1 \\ a_2 \\ \dots \\ a_n \end{pmatrix}, T_p = \begin{pmatrix} t_{11} & t_{12} & \dots & t_{1n} \\ t_{21} & t_{22} & & t_{2n} \\ \dots & \dots & \dots & \dots \\ t_{n1} & t_{n2} & \dots & t_{nn} \end{pmatrix}, \tag{5.2}$$

where

$$a_i = \frac{\sum_{t=1}^{k} I(c_{t1} = i)}{k} \tag{5.3}$$

and

$$t_{ij} = \frac{\sum_{t=1}^{k} \sum_{s=1}^{i_t - 1} I(c_{ts} = i, c_{ts+1} = j)}{\sum_{t=1}^{k} \sum_{s=1}^{i_t - 1} I(c_{ts} = i)}, \tag{5.4}$$

where n is the number of states () k is the number of sequences in a specific cluster and $I(x) = 1$ if x is true and 0 otherwise.

The elements of start probability vector sum up to 1, as well as each row of the transition probability matrix, and are nothing other than discrete probability distribution with n possible outcomes.

Now, with the content of the clusters defined, we can move on to define the mixture Markov model as a collection of components defined by probability distribution of vector v in the model. In order to account for varying lengths of the sequences, the authors in [17] represent each sequence x as vector $v = v_1, v_2, ..., V_L$, where L is the arbitrary length of the sequence, and use Equation 5.2 from definition 5.4 to define the conditional probability distribution (density) for each model component:

$$p(x_i|\theta^j) = p(v|\theta^j) = p(v_1|\theta^j)\prod_{i=2}^{L} p(v_i|v_{i-1},\theta_j) = \prod_{m=1}^{M}(\theta^j_{0m})^{\gamma_i(m)}\prod_{n=1}^{M}\prod_{m=1}^{M}(\theta^j_{nm})^{\delta_i(n,m)} \tag{5.5}$$

where θ^j is the collection of start probability distribution vector and transition probability matrix of the model component $c_j, j = 1...K$. Value $\delta_i(n,m)$ denotes the number of times the transition from state n to state m occurs, and $\gamma_i(m) = \begin{cases} 1 \text{ if } & = m \\ 0 \; otherwise \end{cases}$. Equation 5.9 helps describe the characteristics of the sequences in each cluster, such as starting state and the order of transitions, where each transition depends only on current state (first-order Markov chain) and the number of transitions can be arbitrarily.

The definitions above consider the collection of sequences as a corresponding probabilistic model (or a group of those). The next step of the clustering process is to estimate the model parameters given a collection of sequences, in order then to "softly" position every sequence in the cluster defined by a specific component [27].

5.5.2 Likelihood and Maximum Likelihood Estimation

While the previous section sets up a framework for representing the collection of sequences, it is only part of the clustering task. Finding the parameters of a mixture model (start probabilities vector and transition probability matrix for each of the components) still needs to be performed. Finding parameters is not a trivial task: they cannot be calculated directly due to creating a system of interlocking equations. Thus, another approach is needed. Estimating the parameters (or finding values that are as close as possible to the "true" values) is the most popular alternative, with Maximum Likelihood Estimation via Expectation-Maximization (EM) algorithm being the leader in such estimation algorithms, and I have selected it for use in this thesis. In order to explain the EM algorithm in the context of variable-length sequences, the concept of *likelihood* needs to be introduced [27]:

Definition 5.6.

$$L(\theta|x) = P(x|\theta), \tag{5.6}$$

In other words, likelihood of a set of parameters given the outcome of the random variable is the probability of the outcome of the random variable given the model parameters.

In case of discrete random variable, specifically, this means the probability of this discrete variable, X, taking on the value x, when θ is given (or $P(X = x|\theta)$). The definition thus provides us with a possibility of an objective function for a clustering algorithm: find such parameters for the model which maximize the likelihood of this model, given the data (or outcomes, to preserve the terminology). We can also write it as a formula: $argmax(L(\theta|x_1, ..., x_n)$, where $x_1, ...x_n$ are the data collection (or outcomes). Now we are getting closer to the definition of Maximum Likelihood Estimation. When given a collection of independent outcomes (or independent data - in our case it is sequences), likelihood of the model given a collection of data can be rewritten as the joint probability of the collection given the model , $f(x_1, x_2, ..., x_n|\theta) = f(x_1|\theta)f(x_2|\theta)...(x_n|\theta)$ - due to the conditional independence of the observed data points. It is customary to take the logarithm of the product for maximization purposes, because of the similarity of the growth between any value ant its logarithm (if we increase the value, logarithm also monotonously increases). Taking a logarithm allows for easier computations (the product gets replaced by a sum of the likelihood terms, and sum is easier to maximize). I denote the likelihood function $f(x_1, x_2, ..., x_n|\theta)$ as $L(X|\Theta_K)$ for use in the implementation part, where X is the set of sequences and Θ_K is the mixture model. Notation for the logarithm of likelihood function is ”log-likelihood” function.

As it is seen from the previous paragraph, it is possible to find the parameters of the model that maximize the likelihood of the model. An iteration algorithm that is widely used for maximization is Expectation-Maximization, introduced by Dempster et al [29]. The algorithm finds maximum log-likelihood by repeating two steps until (local) maximum is reached:

Expectation Step. In the Expectation step, log-likelihood function expected value is estimated with respect to the conditional distribution of hidden variables (Z) given the data (X) under the current parameters estimate:

$$Q(\theta|\theta^{(t)}) = E_{Z|X,\theta^{(t)}}\{\log L(\theta; X, Z)\} \tag{5.7}$$

Maximization Step. Find the new parameters that maximize the function from the previous step:

$$\theta^{(t+1)} = \arg\max Q(\theta|\theta^{(t)}) \tag{5.8}$$

E- and M-steps are iteratively repeated until the maximum is reached. One caveat of the MLE by EM-algorithm is being stuck in local maxima, meaning it is a non-deterministic problem. However, for the purpose of this thesis, I will pursue finding the local maximum.

EM-algorithm itself provides a proven scheme to achieve maximization of log-likelihood function. However, implementing Maximum Likelihood Estimation the solution can be analytically hard, due to an integral that arises while maximizing the estimated function. Bayesian Probability concepts provide tools in order to be able to find the maximizing model parameters analytically.

5.5.3 Bayesian Inference and MAP log-likelihood

Bayesian Probability is based on the notion of prior beliefs and the fundamental Bayes formula of conditional probability:

$$P(A|B) = \frac{P(A|B)P(A)}{P(B)} \tag{5.9}$$

In Bayesian Probability, it is assumed that there exists a prior belief about the distribution of the model (which can also be parametrized). Posterior probability, thus, is a prior belief about the distribution updated when incoming evidence is taken into account, and can be achieved with using formula 5.9:

$$P(prior|evidence) = \frac{P(evidence|prior)P(prior)}{P(evidence)}, \tag{5.10}$$

where$P(evidence|prior)$ is likelihood and $P(prior|evidence)$ is posterior probability, which in relationship to a probabilistic model denotes how probable the observed data (evidence) is given model parameters [27],p. 22. Denominator is the "normalization constant" . Thus, Formula 5.10 can be simplified to the following:

$$posterior \propto likelihood \times prior. \tag{5.11}$$

With this new notation, we can update the log-likelihood function from the original MLE algorithm. The function $L(X|\Theta_K)$ gets augmented the following way [30]:

$$L(X|\Theta_K) = \sum_{i=1} N \log f(X_i|\Theta_K) + \sum_{j=1}^{K} \sum_{n=0}^{M} \log p(\theta_n^j|a_n^j), \tag{5.12}$$

where the second term denotes the prior probability with respect to the parameters $a_n^j > 0$, defining the distribution of the model according to some prior knowledge regarding it. These parameters are called *hyperparameters*. The updated objective function for maximization bears the name Maximum a Posteriori (MAP) log-likelihood function [27] .

5.5.4 Dirichlet Conjugate Priors and Calculating the Model Parameters

The second summation term of MAP log-likelihood function (Equation 5.12) is prior probability distribution. Since the clusters (or model components) has been defined in terms of discrete probability distribution -M x M transition probabilities matrix, with row sums equal 1, and similarly a start probabilities vector of length M (number of states) - there is a way to parametrize the prior belief regarding the distribution of the model parameters by using corresponding conjugate prior, which has the same analytical form as discrete multinomial distribution. Since posterior probability can be expressed via prior and likelihood (Formula 5.11), by using conjugate prior it is possible to express posterior using its analytical form. For discrete multinomial distribution conjugate distribution happens to be Dirichlet distribution, expiressed via Gamma function and multinomial hyperparameters $\alpha_i > 0$:

$$Dir(x|\alpha) = \frac{\Gamma(\sum_{i=1}^{K} \alpha_i)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \prod_{i=1}^{K} (\theta_n^j)^{\alpha_i - 1} \tag{5.13}$$

where θ_n^j is the nth row of the matrix defining the multinomial distribution of the transition probability (and for $n = 0$, distribution of start vector probability).

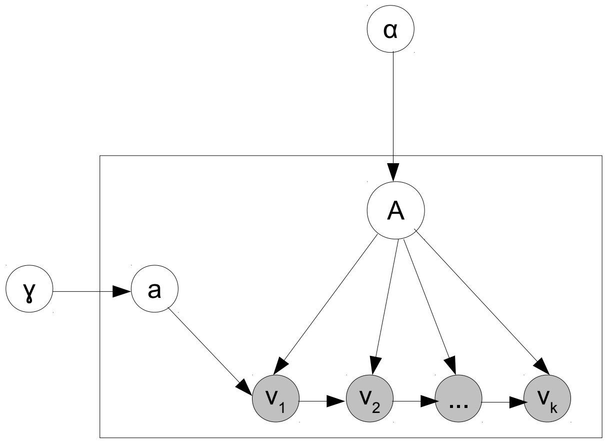

FIGURE 5.3: Markov Model with Dirichlet Priors

Here, the conjugate priors do not necessarily have to reflect an actual belief (non-informative priors). Figure 5.3 depicts the model with transition probability matrix A, hyperparameters for A (α), start probabilities vector a and its Dirichlet prior hyperparameters vector γ, and vector v which is a series of state transitions (v_1 is the first state) of length k.

Now the $\arg\max L(X|\Theta_K)$ can be analytically computed in the M-Step, and the steps acquire the following form:

E-Step. Calculate the conditional expectation values of the hidden variables [30]:

$$p(j|X_j) = \frac{\pi_j^{(t)} p(X_i|\theta^{j(t)})}{\sum_{j'=1} K p(X_i|\theta^{j'(t)})} \tag{5.14}$$

M-Step. Maximize the MAP log-likelihood function (by taking first derivative with respect to each component's parameters), which result in a series of equations involving hyper parameters:

$$\pi_j^{t+1} = \frac{\sum_{i=1}^{N} z_{ij}^{(t)}}{N}, \theta_{nm}^{j\ (t+1)} = \begin{cases} \frac{\sum_{i=1}^{N} z_{ij}^{(t)} \gamma_i(m) + \alpha_{0m}^{j} - 1}{\sum_{i=1}^{N} z_{ij}^{(t)} + \sum_{m'=1}^{M} (\alpha_{0m'}^{j} - 1)}, & \text{if } n = 0 \\ \frac{\sum_{i=1}^{N} z_{ij}^{(t)} \delta_i(n,m) + \alpha_{0m}^{j} - 1}{\sum_{i=1}^{N} z_{ij}^{(t)} \sum_{m'=1}^{M} \delta_i(n,m') + \sum_{m'=1}^{M} (\alpha_{nm'}^{j} - 1)}, & \text{if } n > 0 \end{cases} \tag{5.15}$$

Equation 5.15 thus provides the updated model parameters for the next iteration.

EM-algorithm which maximizes MAP log-likelihood objective function, finds a point estimate of the posterior distribution (or its mode).

5.6 Iterative Algorithm Implementation and Optimizations

5.6.1 Programming the Algorithm and Preparing Sample for Mixture Modeling

I have implemented the algorithm described in the previous subsection by programming the methods in `R` language. This statistical language provides powerful architecture for working with multidimensional data. After a thorough assessment of available libraries for mixture modeling and Markov chains, I have not found a suitable library to apply directly. Thus, I have made a decision to create a library of functions and sub-functions which perform step-by-step calculations and define data structures that store the model components. The functions have been built in order to interact with the existing data structures derived in previous section. An example of a function representing likelihood of sequence x_i in the model can be seen on Figure 5.5.The reader can examine the functions in Appendix B.

```
likelihood_function_one_seq<-function(sequence,modelK,statescollection){
  modlen<-length(modelK)
  p<-0
  for (i in c(1:modlen)){p<-p+modelK[[i]]$marginal.prob*p_seqi_modj_for_model(
    sequence,modelK[[i]],statescollection)}
  p
}
```

Listing 5.5: Example Function - Likelihood

A mixture model is conveniently represented by `list` structure available in `R` which represents a list of mixture components, each containing three subcomponents: marginal probability (mixing parameter) , transition probability matrix and start probabilities vector.

In order to estimate the mixture model parameters, I have used half of the sample of each class (1 and 0), leaving the other half of the sample for testing and classification. I have sequentially clustered first class 1, then class 0.

5.6.2 Initializing the Model

As a first step of applying the Markov mixture modeling algorithm which I have implemented, the model needs to be initialized. I have chosen to start with one component. A widely accepted way to initialize a Markov model is to set the transition probability matrix as the sum of all transitions across all sequences in the sample, and start probability vector as a sum of all occurrences of given state as a first state in the sequences. After examining the transition probability matrix and start probabilities vector, I have noticed that they contain 0's, which could prevent the algorithm from executing. In order to avoid it, I have selected a smoothing strategy of adding minimal pseudo counts to the transitions and starting states which were not represented in the model and then normalizing the matrix to sum up to 1. The updated initialized model can be seen in Listing 5.6. Another part of the initialization is setting hyper parameters for the Dirichlet conjugate priors. I have set all multinomial hyperparameters α_n^j to be close to 1 (1.001), thus creating uniform non-informative priors. With initial model parameters set, I could now perform the iterations algorithm. After reaching local maximum with one component and proving the concept, I have moved on to work with more than one component. I have run into several challenges, described in the next section.

```
> modelK
[[1]]
[[1]]$transition.matrix
         [,1]    [,2]    [,3]    [,4]    [,5]    [,6]    [,7]    [,8]   ...
 [1,] 1.0e-17 1.0e-01 1.0e-01 5.0e-02 1.0e-17 2.5e-01 1.0e-01 1.5e-01   ...
 [2,] 8.9e-03 2.7e-01 3.0e-03 6.5e-02 1.0e-17 1.5e-01 3.0e-02 3.0e-01   ...
 [3,] 2.0e-02 1.0e-01 2.9e-01 1.2e-01 6.1e-02 6.1e-02 6.1e-02 1.4e-01   ...
 [4,] 2.4e-02 2.2e-01 4.9e-02 2.0e-01 8.1e-03 1.1e-01 7.3e-02 1.5e-01   ...
 [5,] 4.8e-02 4.8e-02 1.9e-01 1.4e-01 9.5e-02 9.5e-02 9.5e-02 4.8e-02   ...
 [6,] 3.8e-03 1.6e-01 1.5e-02 3.4e-02 1.0e-17 1.5e-01 3.8e-02 4.0e-01   ...
 [7,] 1.6e-02 1.6e-01 3.1e-02 1.6e-01 2.3e-02 1.3e-01 1.1e-01 2.3e-01   ...
 [8,] 1.5e-03 2.4e-02 9.8e-04 2.9e-03 9.8e-04 2.4e-02 9.3e-03 8.4e-01   ...
 [9,] 8.7e-03 1.7e-02 2.6e-02 6.1e-02 1.0e-17 1.4e-01 1.0e-17 2.5e-01   ...

[[1]]$start.vector
 [1] 2.0e-02 2.0e-02 2.0e-02 4.0e-02 1.6e-01 4.0e-02 5.4e-01 1.0e-17   ...

[[1]]$marginal.prob
[1] 1
```

Listing 5.6: Initialized Model with One Component

5.6.3 Overcoming Challenges of Implementation - Underflow, Performance

Initial sequential mixing of the model has highlighted the issues that arise with trying to train a mixture model using first order Markov Chains as its components, given the sparse character of the sequences and sometimes very long sequences. I need to specifically mention that sparse means sequences containing a lot of inactivity days (denoted by state ”8” in the clustering results). As seen from the previous steps of implementation, upon clustering of the multivariate measurements data, it is highly possible that not all states and transitions will be represented in a sequence, while the others might be overrepresented. This, combined with length, creates the possibilities of extremely small likelihood values which might cross the limit of digits that `R` can store - 321 digits after decimal point. In order to illustrate this, I will use an example of a sequence that contains a state transition $s1 \rightarrow s2$ which has a probability of 1e-10 in the model, 100 times. Using Formula 5.9, we get a term of magnitude that is rounded to 0 in the software. This is referred to as ”underflow” and needs to be regularized, otherwise the algorithm will try to take the logarithm of such value and get a -Infinity value. As an in-process fix, I have approximated 0 with the smallest possible value: 1e-321. Although in the mathematical sense such value is the same as 0, the algorithm can still continue maximization, while sacrificing precision in the very small digits. The discussion of this problem will continue in the evaluation section.

One of the problems of then EM-algorithm is its slow convergence rate - given a large collection of states and sequences, the number of iterations needed to achieve convergence can be high and cannot be determined a-priori. It has been confirmed with my experiment with the churned sequences - after 80 iterations with just 100 sequences of average length 100 and 17 states, the algorithm still have not reached convergence. I have defined convergence in terms of mathematical limit notation: I consider algorithm converged when the difference between the new estimate and old estimate (δ) is less than an arbitrarily small number (ϵ), which I have set to 1e-17. Such a slow rate of convergence given just 2 components was not acceptable for my purposes - knowing from the data exploration, that the other class of sequences (class 0, or "non-churners"), consists of sometimes much longer sequences, I have searched for approaches to improve the algorithm's robustness

The first step towards improving algorithm robustness is based on **??** . The authors of the article suggest a novel approach - "Incremental Mixture Modeling". The approach suggests sequentially adding each new component to the mixture and performing a partial EM-algorithm (only with respect to the parameters of the new component, while keeping all the other values fixed). This is performed using the algorithm depicted in Listing 5.7.

```
present the  likelihood function as a sum of existing value and the likelihood
  of new component
maximize the new objective MAP log-likelihood function with respect to the
  parameters of the new component
adjust the mixing coefficients for existing model according to the new value of
   the optimized mixing coefficient of the new component.

```

Listing 5.7: Partial EM-algorithm

The new MAP log-likelihood function can be presented by Formula 5.16.

$$L_k = \sum_{i=1}^{N} \log \{(1-\pi^*) f(X_i|\Theta_k) + \pi^* p(X_i|\theta^*)\} + \sum_{n=0}^{M} \log p(\theta_n^*|a_n), \tag{5.16}$$

where Θ_k is the existing model with k components, θ^* is the new model component, π^* is the mixing coefficient of the new component. With the the maximization objective defined this way, the E- and M-steps can be expressed in terms of exclusively parameters of the new model.

Another step in order to implement the partial EM-algorithm is to initialize the new model component (there are thee terms to be initialized: transition probability matrix, start probabilities vector and mixing coefficient). For my experiment, I have set each row of the matrix and the vector to be uniformly distributed (i.e. each parameters is equal 1 divided by the number of states N=17), while setting the mixing coefficient π^* to $1/(k+1)$. This allows me to start with values that are not too extreme (see previous subsection) and make no uninformed assumptions regarding the distribution in the new component.

Now, with all the inputs ready for the partial EM-algorithm (trained model with 1 component and the new initialized new component), I have run the function `partial_em()` which I implemented using the same structure as the general EM function, with adjustments to account for the algorithm modification, which resulted in mixing coefficient $\pi^* = \pi_2 = 0.049$. The mixing coefficient of the first component has thus become $\pi_1 = 1-0.049 = 0.951$. This means there are about 5 per cent of sequences in the training collection which have higher likelihood under the parameters of the second component.

Now, the general EM-algorithm is initialized with the updated (and optimized) parameters, resulting from the partial step . Convergence with two components has been reached within approximately 20 iterations, winning over the original algorithm.

5.6.4 Adding Robustness to the Algorithm via Parallel Processing

While implementing the algorithm, I have found that the way calculations are implemented are of utter importance. Calculating matrix parameters with the sample of 100 sequences involves repetition on multiple levels. In the initial implementation attempt, I have used nested `for` loops, which have proven to be extremely slow. As the first optimization, I have replaced all the original loops with the list calculation functions - such as the function `apply()`, which works on lists of objects and delivers speed-up over the original timing. I have iteratively tested the optimization of each `for` loop and replaced with `apply()`-type functions, gaining robustness. However, each iteration with 2 components still stayed time-consuming, and with 3 components not manageable on my machine (each iteration took up to 2 hours).

As the final step in optimization of my implementation, I have used the functionality of parallel computing in `R`. I have used the functions of libraries `parallel`, `doMC` and `foreach` which allows to register the number of workers (processors or cores) and distribute the processing through available resources, using such functions as `mclapply,registerDoMC and foreach`. By parallelzing the execution of the algorithm, I have achieved time savings of 50 per cent with the dual core processor, making iterative testing of the algorithm possible. I have chosen to parallelize composite tasks which did not depend on each other and could be executed in arbitrary order, and left repetitive tasks with less calculations out of scope for parallelizing, because the overhead on such tasks - to combine results, for example - exceeds time savings. Further improvements can be potentially achieved with multiple cores. The comparisons of performance of parallelized algorithms can be seen in `R` output in Listing 5.8, with 100 sequences from training sample for class 0 and a model with one component.

```
# non-parallelized e- and m-step
> system.time(test<-e_step_z_i_j_t(res1002nq[[1]],worksample100nonquit,states))
     User      System verstrichen
   554.109      10.415     1301.476
> system.time(testres<-m_step(test,worksample100nonquit,alphamatrix,res1002nq
    [[1]],states,2,100))
     User      System verstrichen
  6425.331      21.502    6480.020

# parallelized e- and m-step
> system.time(test<-estep_par(res1002nq[[1]],worksample100nonquit,states))
     User      System verstrichen
   243.886       1.415     566.655
> system.time(testres<-m_step_par(test,worksample100nonquit,alphamatrix,res1002nq
    [[1]],states,2,100))
     User      System verstrichen
  3129.008      21.502    3489.221
```

Listing 5.8: E-step and M-step Execution Times - Simple vs Parallel

5.6.5 Iterative Convergence of the Model

Finally, after the algorithmic and programmatic optimizations of the implemented EM-algorithm, I have witnessed monotonous convergence of the objective MAP log-likelihood function to the local maximum. Due to resources and time constraints, I have managed to implement a 3-component model describing class = 1 ("churners") and 1-component model describing class=0 ("non-churners"). The graphs on Figures 5.4 and 5.5 illustrate the convergence process of the algorithm with each step and improvement of the objective function with addition of new clusters.

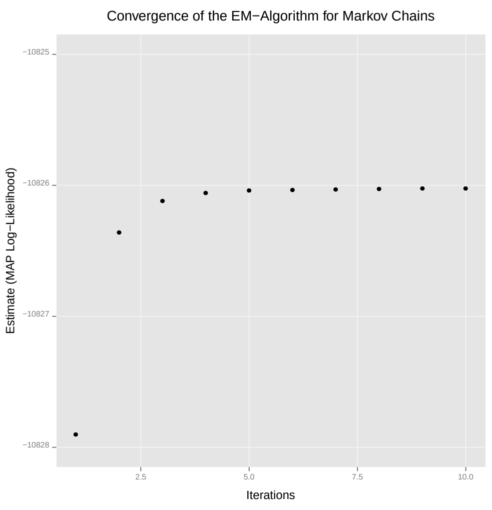

FIGURE 5.4: Convergence of the EM MAP log-likelihood algorithm

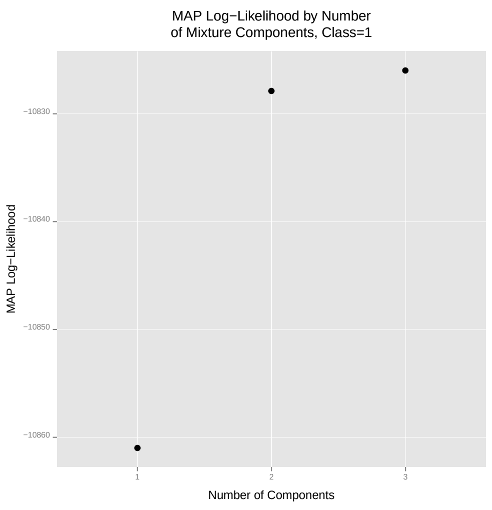

FIGURE 5.5: MAP log-likelihood by number of components

5.7 Classification of Churn Using Naive Bayes Method

The last step of the algorithm is to classify the user sequences from test sample based on the mixture models resulting from the previous step. The authors of [17] suggest the Naive Bayes Method, which is a natural continuation for the previous steps, since it uses the notion of likelihood. The implementation of the EM-algorithm provides the framework and functions for calculating likelihood. The Furthermore, the method allows not just to classify the test sequences but also to index them. Translated into business language, this technically allows to calculate the risk index of a certain customer churning. The risk index can be calculated as Formula 5.17.

$$Score = \frac{prob\_churn}{prob\_churn + prob\_nonchurn}, \tag{5.17}$$

where $prob\_positive$ and $prob\_negative$ is the probability of the test sequence given model parameters for positive and negative model respectively. The test sequence is labeled as class 1 ("churned") if $prob\_churn \geqslant prob\_nonchurn$, and such probabilities represent the likelihood of the sequence (Formula 5.9). It is also possible to take logarithm of the equation and measure in logarithmic terms instead.

I have calculated the the likelihoods of sequences from the training sample from both class Class 1 and Class 0. Initial results returned a high degree of bias towards Class 0 (60 per cent of sequences from Class 1 got misclassified into Class 0, however such misclassification rate was low for Class 0). After research of possible factors, I have noticed that due to the non-churners sequences being on average 3 times longer than the churners sequences, the mixture trained on them was ”overtrained” compared to the model trained on the shorter sequences. Thus, a normalizing parameter was needed. For this solution, I have experimentally arrived at the parameter to be around 1e+3 (or 3 in logarithmic notation). The argument for correctness of such solution is that the sequences are sparse (meaning, similar state - the one representing 0 - is repeated many times), it is possible to scale the likelihood. After multiplying the top part of the formula by the normalizing constant, I have received significant improvement: misclassification rate was drastically reduced for Class 1 (75 per cent of the test sequences were correctly labeled this time), while not significantly reducing the characteristics for Class 0 (68 percent). While such a normalizing constant has been used as an ad-hoc measure, it did bring convincing results: the models trained with 100 sequences from training sample have been used to test 817 sequences for Class 1 and 604 for Class 2.

```
> head(quitters.test.risk)

   uid risk.index
1  u01 0.6724255
2  u02 1.0000000
3  u03 0.9975812
4  u04 0.0002743
5  u05 0.9988836
6  u06 0.9994875
```

Listing 5.9: Users and Their Risk Index

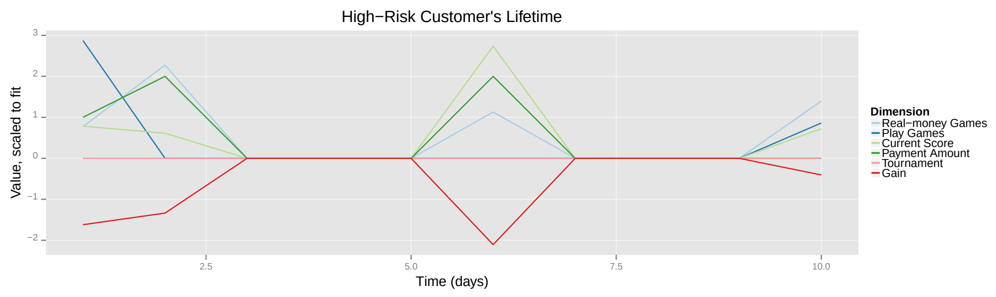

Figure 5.6: An example of a customer with high risk of churn

A sample of users with risk indexes, calculated using Formula 5.17, can be seen in the Listing 5.9. This allows the users to be split into "high-risk", "medium-risk" and "low-risk" of churn. Additionally, an example of a multivariate sequence representing one of the high-risk users, is displayed on Figure 5.6.

This concludes the implementation and testing of the compound algorithm for long-term churn classification. Further evaluation is discussed in the next chapter.

Chapter 6

Algorithms Evaluation

6.1 Evaluation

In order to evaluate the both algorithms which represent and combine the temporal information in various ways, it is important to distinguish that each of the algorithms has a different context, such as prediction horizon (short-term vs longer term prediction), as well as data acquisition was performed differently. Thus, they will be evaluated in isolation according to the available criteria of classification goodness regarding their potential usefulness in churn prediction as well as possibilities for improvements for both methods.

It is important to define the "success" of a classification experience in the context of customer churn. Here, two factors are at play: the fluidity of the non-contractual churn definition and the nature of the subjects of prediction. As opposed to,for example, physical processes that run their course according to certain rules, modeling human behavior and intentions is a task that potentially involves multitude of latent variables. An incomplete list of such factors is change of income, change of taste, change in family status of the user, illness and other life events that cannot be recorded in the database. Thus, performance of such prediction/classification can be considered a success if it presents improvement over random chance (50 per cent) by a certain margin, perhaps 20-30 per cent. The evaluation that follows takes these considerations into account, and looks at the methods from the perspective of potential for improving the algorithms in the given context.

6.1.1 Random Forest Classification - Advantages and Pitfalls

The method has been used in order to predict the future behavioral pattern of users with the short horizon (1 month), or predicting users at risk of quitting (Definition 3.2) based on a collection of features which both represented statistical features collected on a weekly basis. This representation of the features was an attempt to represent temporal changes in aggregated view (for instance, to represent an increasing trend as a boolean value) in order to prepare the customer lifetime characteristics - across the whole lifetime and recent - for input into the classifier. The performance of such classifier appeared to be mixed, and dependent on how strict the combination of outcomes for each of the four weeks of the short-term prediction period is. Figure 6.1 shows such measures as level of bias of the classifier towards one class, and the average error rate depending on the strictness of the prediction in the experiment. The experiments are described in Table 4.2.

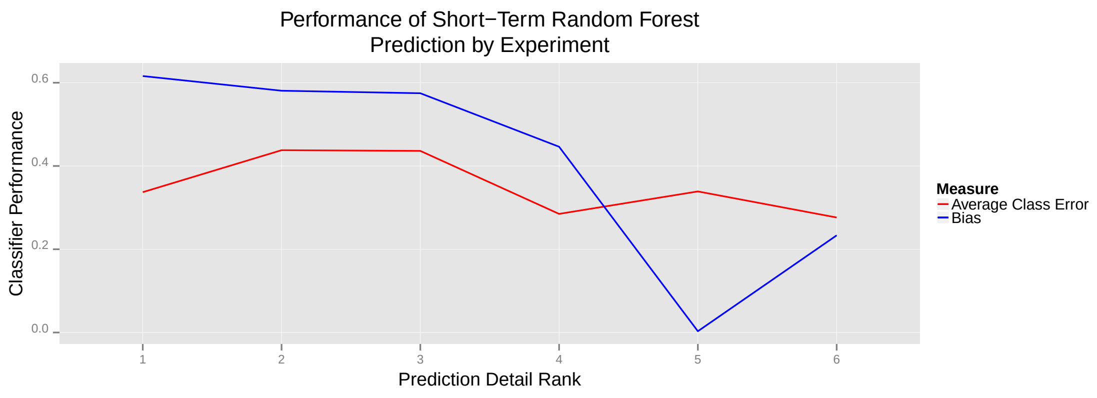

FIGURE 6.1: Classifier Error Rate and Bias in relation to the Strictness of Predicted Responses

Another factor in bias is the proportion of each class labels in the response. Figure 6.2 illustrate correlation between the proportion of class label and classifier bias.

The above illustrations show the factors that affect Random Forest Classifier's bias. However, seeing that the classifier does perform well in some cases sets basis for belief that with the help of iterative improvements it is possible to find the optimal fit of the classifier. Such improvements could potentially be:

- add weights to more influential features

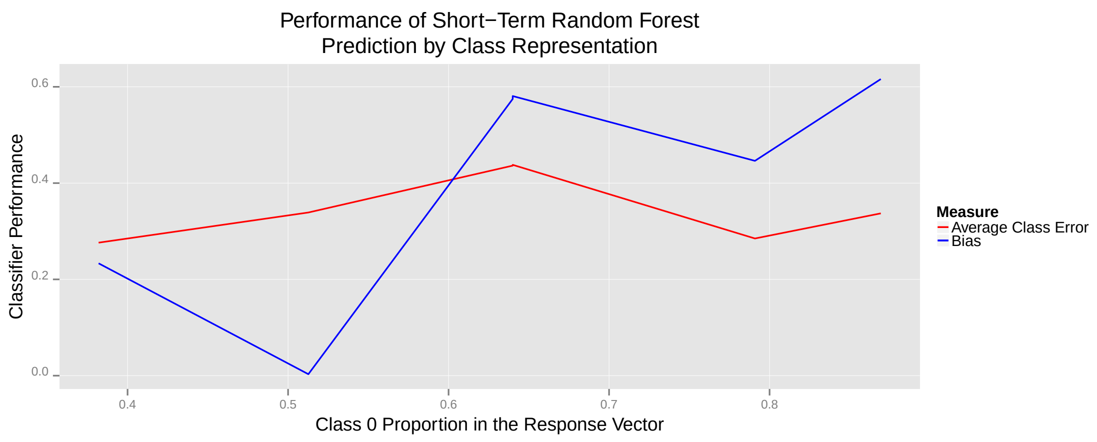

Figure 6.2: Classifier Error Rate and Bias by Class Representation

- filter out the features that insignificantly (or even negatively) affect the classification outcome, according to various information criteria
- rebalance the sample to represent the classes more equally
- select additional features that might have been missed
- change the length of the subsequence of the customer lifetime that is looked at in greater detail

Given that Random Forest Classifier takes any features that have discriminative values as an input, there are variations possible in representation of the overall and current customer lifetime in order to improve the classifier performance.

6.1.2 Mixture Markov Modeling and Naive Bayes Classifier

As mentioned in the previous Chapter, this approach has been marked by it's relative success: average error rate of (0.32+025/2=0.285). Such an error rate has showed that the framework including the churn definition, selected measured features, granularity of the time axis units and the probabilistic/temporal approach to classification is indeed a powerful tool to classify customer churn on non-contractual basis. However, as the implementation and testing of the method has shown, there are certain challenges in applying the method given the peculiarities of the dataset that I have worked with. The most outstanding shortcoming of the method can be named "dimensionality curse" in the following sense:

- the variety of combinations might lead to a relatively large number of clusters, needed in order to preserve the meaning of the states,
- varying (from single digits to hundreds of transitions) length of sequences prevents any simple scaling of the algorithm,
- necessity of the small subsample of the sequences to work with (100) due to computational resources limitation such as RAM,
- "sparsity" of sequences - in some cases the "inactive" state is prevalent among the transitions; however, it cannot be excluded in the given representation, since daily measurements are taken
- working in probability space (very small numbers)
- necessity for approximations and manual decisions for optimal number of clusters.

These problems, coupled together, make the model prone to create suboptimal results due to very small likelihood values. Additional factors that affect the success of model's precision and recall, are:

- underflow problem as combination of the model and computer architecture
- bias of the model towards smaller sequences (which show to be more "likely" in the model than the longer ones)
- relatively slow convergence of the algorithm, even with the iterative improvement outlined in the previous Chapter.

In order to address the above problems and improve the aspects of the model performance, a number of approaches can be taken. Assuming a Hidden Markov Model instead of mixture of first-order Markov model might be helpful in working with longer sequences. Such approach might alleviate the extremities the length of the sequences cause in some cases, as well as help regularize the sequences. Also, further understanding and improvement of bias according to the length of the sequence is needed: while in some cases a simple constant can solve the problem, there are more complicated scenarios (where the sequences do not represent a specific sequential pattern in state change). In these cases, an analytical expression needs to be derived to account for more scenarios of the state transitions within the sequence.

In order to avoid convergence rate problem and complexities of the Expectation-Maximization algorithm, there is a number of other methods available for finding the best fit of the probabilistic model: for instance, variational Bayes methods, which allow to select a gradient of the log-likelihood values instead of the point estimate. This can potentially increase robustness of the model.

Another way to improve the modeling outcome for probabilistic mixtures is selecting more features with predictive potential while removing noisy features. As a separate way to improve the Naive Bayes classification, the probabilistic mixture could potentially be an ensemble of both Markov mixture and mixture of distributions of other static dimensions which are not included in the Markov modeling. By iteratively determining weights for components of such ensemble, it is potentially possible to further the possibility of distinguishing between the classes.

6.2 Conclusion - Final Thoughts and Future Possibilities

In this thesis I have explored in detail two approaches to the problem of non-contractual customer churn on an example dataset provided by an online gaming company with vast and fluid customer base. The outcomes of the experiments have underlined the importance of defining the context and axiomatic rules in order to give a meaningful model of a business problem, as well as strengths and weaknesses of each method. The hypothesis that I have put forward in the beginning of this paper - that looking for information within temporal transitions adds possibilities for long-term predictions - has been proven with limitations. Additionally, the thesis has shown that static representations of temporal features can in some cases predict if customer is at risk of churn, while probabilistic representation can be useful in cases where direct numerical values of the features do not provide much knowledge. Finally, the experiments have been able to prove that various knowledge engineering methods can work well in ensemble and there is potential for improvements using a wider variety of tools and methods.

The above conclusions provide potential areas for future research. One of the examples is early churn classification (which bases predictions on the early stages of customer lifetime). Other areas might include more detailed description of the role churn plays in overall Customer Lifetime Value, combined with other characteristics of the customers - and vice versa. The possibilities of creating sophisticated classifiers addressing these problems create a truly positive outlook on the help Machine Learning and Data Mining can provide to the companies in the area of Customer Relationship Management.

Appendix A

Visualizations

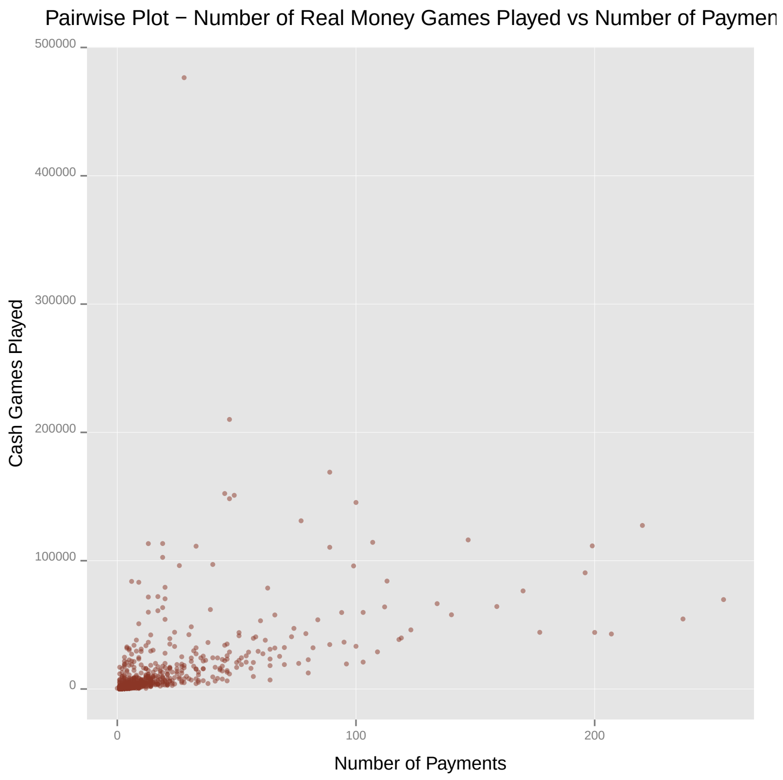

Figure A.1: Pairwise Correlation Plot - Number of Payments vs Real Money Games Played

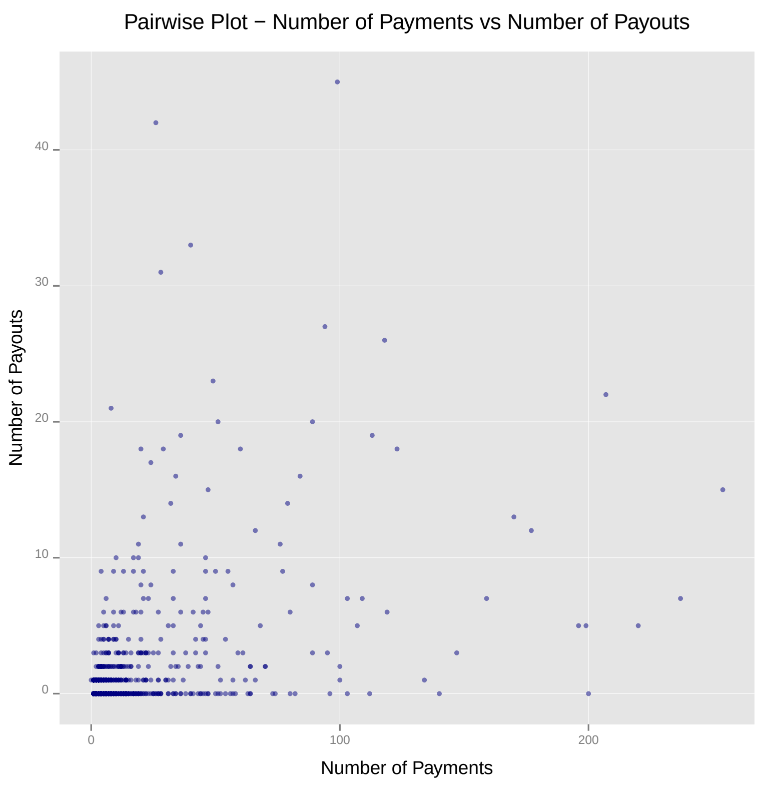

FIGURE A.2: Pairwise Correlation Plot - Number of Payouts vs Number of Payments

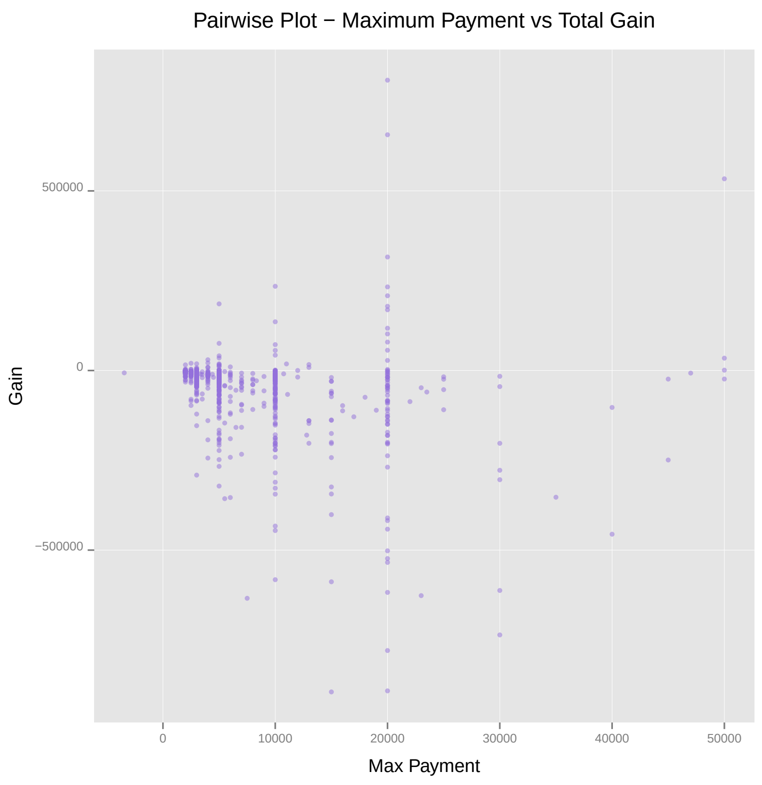

FIGURE A.3: Pairwise Correlation Plot - Maximum Payment vs Gain

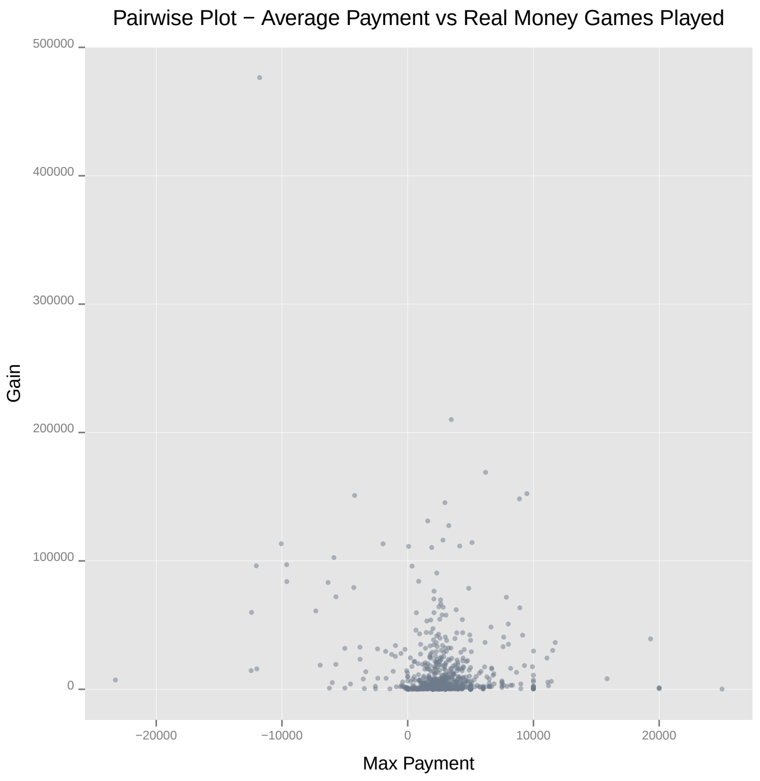

Figure A.4: Pairwise Correlation Plot - Average Payment vs Number of Real Money Games Played

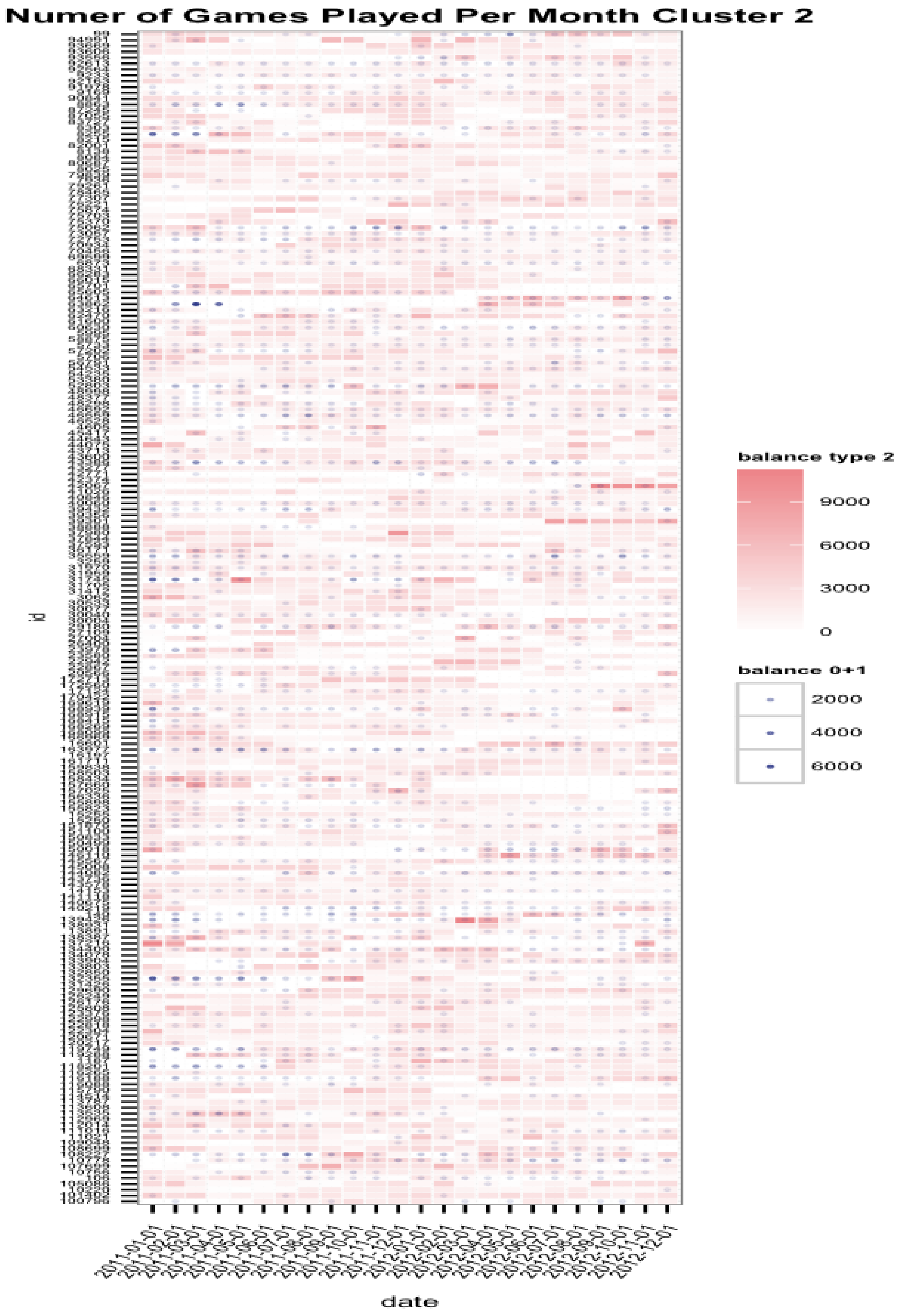

FIGURE A.5: Diversity of Activity Patterns in Cluster 2 (Dimensions - Free Games (Balance Type 0+1), Cash Games (Balance Type 2)

Appendix B

R Code Implementing Incremental Markov Mixture Modeling

```
#
# Incremental_Markov_Mixture_Modeling.R
# This file contains functions implementing Mixture Markov Modeling
# by means of iterative EM algorithm
# Author: Natalya Furmanova, TUHH, #21044449
# 10/2013
#

library(parallel)
library(foreach)
library(doMC)

# Function to calculate first-order Markov chain transition matrix
trans.matrix.proper <- function(X,states_collection, prob=T)
 {
  transmat<- mat.or.vec(nr=length(states_collection),nc=length(states_collection)
    )
    #if(is.numeric(X)) {X=matrix(rep(X,2),nrow=2,byrow=T)}
  for (ii in c(1:dim(transmat)[1]))    {
    for (jj in c(1:dim(transmat)[2])) {
      transmat[ii,jj]<-0
      for (kk in c(1:(length(X)-1))) {if((X[kk]==ii)&&(X[kk+1]==jj)){
          transmat[ii,jj]<-transmat[ii,jj]+1}}
      if(is.na(transmat[ii,jj])){transmat[ii,jj]<-0}
    }
  }
  rs<-rowSums(transmat)
  rs[which(rs==0)]<-1
     transmat<-transmat/rs
```

```
      transmat[which(transmat==0)]<-10^(-17)
      transmat
 }

# Function to calculate transition matrix for a collection of sequences
trans.matrix.for.collection <- function(seq_collection,states_collection, prob=T)
 {
  transmat<- mat.or.vec(nr=length(states_collection),nc=length(states_collection)
    )
    #if(is.numeric(X)) {X=matrix(rep(X,2),nrow=2,byrow=T)}
  for (i in c(1:length(seq_collection))) {
  for (ii in c(1:dim(transmat)[1]))    {
    for (jj in c(1:dim(transmat)[2])) {
      #transmat[ii,jj]<-0
      for (kk in c(1:(length(seq_collection[[i]])-1)))
      {if((seq_collection[[i]][kk]==ii)&&(seq_collection[[i]][kk+1]==jj))
        {transmat[ii,jj]<-transmat[ii,jj]+1}}
      if(is.na(transmat[ii,jj])){transmat[ii,jj]<-0}
    }
  }
  }
  rs<-rowSums(transmat)
  rs[which(rs==0)]<-1
    transmat<-transmat/rs
    transmat[which(transmat==0)]<-10^(-17)
    transmat
 }

# Function to calculate start probabilities vector of a sequence
start.prob.vector.sequence<-function(seqi,states_collection){
  vec<-sapply(states_collection,function(x){ifelse(x==seqi[1],1,10^(-17))})
  vec
}

# Function to calculate start probabilities vector for a collection of sequences
start.prob.many.seq<-function(sequence_collection,states_collection){
  vec<-rep(0,length(states_collection))
  for (i in c(1:length(sequence_collection))) {
     vec<-vec+sapply(states_collection,function(x)
     {ifelse(x==sequence_collection[[i]][1],1,0)})
  }
  vec<-vec/length(sequence_collection)
  vec[which(vec==0)]<-10^(-17); vec
 }

# Function to build Dirichle conjugate priors matrix
build.alphaMatrix<-function(startVector,probMatrix){
  alphamat<-mat.or.vec(nr=dim(probMatrix)[1]+1,nc=dim(probMatrix)[2])
  alphamat[1,]<-0.1*startVector
  alphamat[c(2:dim(alphamat)[1]),]<-0.1*probMatrix
  alphamat
}

# Model initializer with one component
initModel<-function(first_component){
  mod<-list(first_component)
}


# Initializer for a mixture Markov model component
initModelComponent<-function(transmat,startprobs,marginalprobability){
  modelcomp<-list(transmat,startprobs,marginalprobability)
  names(modelcomp)<-c("transition.matrix","start.vector","marginal.prob")
  modelcomp
 }

# Helper function to calculate gamma_i_m coefficient
gamma_i_m<-function(observationM,vector_xi){
  a<-ifelse(vector_xi[1]==observationM,1,0)
  a
}

# Helper function to calculate delta_i_n_m coefficient
delta_i_n_m<-function(observationN,observationM,vector_xi){
  dd<-0; seqlen<-length(vector_xi)-1
    for(k in c(1:seqlen)){dd<-ifelse(((vector_xi[k]==observationN)
      &&(vector_xi[k+1]==observationM)),dd+1,dd)}
    if(is.na(dd)){dd<-0}
    dd
 }


# Function to calculate likelihood of a sequence in the model with K components
likelihood_function_one_seq<-function(sequence,modelK,statescollection){
  modlen<-length(modelK)
  p<-0
  for (i in c(1:modlen))
    {p<-p+modelK[[i]]$marginal.prob*p_seqi_modj_for_model(sequence,modelK[[i]],
      statescollection)}
  p

}

# Function to calculate log-likelihood function value over a collection of
    sequences,
# given parameters of model with K components and a collection of n states
log_likelihood_function_all_seq<-function(seq_collection,modelK,collection){
  sum(log(sapply(sapply(seq_collection,function(x)
  {likelihood_function_one_seq(x,modelK,collection)}),function(x)
    {ifelse(x==0,1e-17,x)})))
}

# Function to calculate log-likelihood function using parallel processing
log_likelihood_function_all_seq_par<-function(seq_collection,modelK,collection){
  sum(log(unlist(mclapply(unlist(mclapply(seq_collection,function(x)
    {likelihood_function_one_seq(x,modelK,collection)})),function(x)
      {ifelse(x==0,1e-17,x)}))))
}

# Helper function to calculate the second term of MAP log-likelihood function
```

```
sum_of_dirichlet_priors<-function(modelK,alphaMatrix,states){
  func<-0
  for (j in c(1:length(modelK))) {
      for (n in c(1:length(states))){
        func<-func+log(dirichletPrior(
          modelK[[j]]$transition.matrix[n,],alphaMatrix[n+1,]))
      }
      func<-func+log(dirichletPrior(modelK[[j]]$start.vector,alphaMatrix[1,]))
    }
  func

 }

# MAP-log-likelihood function
map_log_likelihood_function<-function(modelK,sequenceCollection,alphaMatrix,
    states){
  func<-0
  func<-func+log_likelihood_function_all_seq(sequenceCollection,modelK,states)

  for (j in c(1:length(modelK))) {
      for (n in c(1:length(states))){
        func<-func+log(dirichletPrior(modelK[[j]]$transition.matrix[n,],
          alphaMatrix[n+1,]))
      }
      func<-func+log(dirichletPrior(modelK[[j]]$start.vector,alphaMatrix[1,]))
    }
  func
}

# MAP log-likelihood function using parallel processing
map_log_likelihood_function_par<-function(modelK,sequenceCollection,alphaMatrix,
    states){
  func<-0
  func<-func+log_likelihood_function_all_seq_par(sequenceCollection,modelK,states
    )

  for (j in c(1:length(modelK))) {
      for (n in c(1:length(states))){
        func<-func+log(dirichletPrior(modelK[[j]]$transition.matrix[n,],
          alphaMatrix[n+1,]))
      }
      func<-func+log(dirichletPrior(modelK[[j]]$start.vector,alphaMatrix[1,]))
    }
  func
}

# Probability of a sequence in a Markov model of another sequence
p_seqi_modj<-function(seqi,modj,collection){
        p<-1
   modj.trans.matrix<-trans.matrix.proper(modj,collection)
    for (i in c(1:dim(modj.trans.matrix)[1])) {
      for (j in c(1:dim(modj.trans.matrix)[2])) {
        p<-p*((modj.trans.matrix[i,j])^(delta_i_n_m(i,j,seqi)))
      }
     }
    start.vector<-start.prob.vector.sequence(modj,collection)
    for ( i in c(1:length(start.vector))) { p<-p*start.vector[i]^(
                    gamma_i_m(i,seqi))}
  p
}

# Probability of a sequence in a model
p_seqi_modj_for_model<-function(seqi,modj,collection){
        p<-1

    for (i in c(1:length(modj$start.vector))) {
        p<-p*((modj$start.vector[i])^(gamma_i_m(i,seqi)))}
    for (i in c(1:dim(modj$transition.matrix)[1])) {
      for (j in c(1:dim(modj$transition.matrix)[2])) {
        p<-p*((modj$transition.matrix[i,j])^(delta_i_n_m(i,j,seqi)))
      }
    }
p }

# Helper function - log-likelihood distance between two sequences
log_likelihood_distance<-function(seqi,seqj,states_collection){
  dist<-0.5*(log(p_seqi_modj(seqi,seqj,states_collection))+
  log(p_seqi_modj(seqi,seqj,states_collection)))
  dist
}


# Function to calculate Dirichlet prior given a row of TPM and a row of
# Hyperparameters
dirichletPrior<-function(matrix_row,alpha_row){
  dirichl<-((gamma(sum(alpha_row)))/
    (prod(gamma(alpha_row))))*prod(c(matrix_row^(c(alpha_row-1))))
  dirichl
}


# Function that builds a distance matrix for a collection of sequences
build_distance_matrix<-function(list_of_sequences,states_collection){
  m1<-mat.or.vec(length(list_of_sequences),length(list_of_sequences))
    for (i in c(1:(length(list_of_sequences)-1))){
      for ( j in c((i+1):length(list_of_sequences))) {
          m1[j,i]<-m1[i,j]<-
          (-log_likelihood_distance(list_of_sequences[[i]],
          list_of_sequences[[j]],states_collection))

      }
    }
  m1[which(is.na(m1))]<-2*max(m1[which(!(is.na(m1)))])
  m1
  }

# Helper function to create a list of sequence from the dataframe
create_sequence_list<-function(seq_df){
  seqlist<-list(NULL)
  for ( i in c(1:length(unique(seq_df$uid)))) {
    seqlist[[i]]<-c(seq_df$state[which(seq_df$uid==unique(seq_df$uid)[i])])
  }
  seqlist
}

# E-step: calculate conditional posteriors
e_step_z_i_j_t<-function(modelK,sequence_collection,collection){
  zposteriors<-list(NULL)
  zposteriors<-lapply(seq(1:length(modelK)),function(j){ sapply(
        seq(1:length(sequence_collection)),function(i){
        numerator<-((modelK[[j]]$marginal.prob)*(p_seqi_modj_for_model(
          sequence_collection[[i]],modelK[[j]],collection)))
        denominator<-(likelihood_function_one_seq(
          sequence_collection[[i]],modelK,collection))
        if (numerator==0) {numerator<-1e-321}
        if (denominator==0) {denominator<-1e-321}
        numerator/denominator
          })
        })
  zposteriors
}

# E-step in parallel
estep_par<-function(modelK,sequence_collection,collection){
  zposteriors<-list(NULL)
  zposteriors<-mclapply(seq(1:length(modelK)),function(j){
        unlist(mclapply(seq(1:length(sequence_collection)),function(i){
        numerator<-((modelK[[j]]$marginal.prob)*(p_seqi_modj_for_model(
            sequence_collection[[i]],modelK[[j]],collection)))
        denominator<-(likelihood_function_one_seq(
            sequence_collection[[i]],modelK,collection))
        if (numerator==0) {numerator<-1e-321}
        if (denominator==0) {denominator<-1e-321}
        numerator/denominator
        }))
      })
  zposteriors
}
```

./imm_part12.R

```

# Helper fuction for M-step - calculate start vector
modify_start_vector_j<-function(posteriors_j,sequences_collection,alphaMatrix,
            states_collection){
  sapply(states_collection,function(m){
    (sum((posteriors_j)*(sapply(sequences_collection,function(x){
        gamma_i_m(m,x)})))+alphaMatrix[1,m]-1)/
          (sum(posteriors_j)+sum(alphaMatrix[1,]-1))})
}

# Helper for M-step - calculste start vector using parallel processing
modify_start_vector_j_par<-function(posteriors_j,sequences_collection,alphaMatrix
    ,
              states_collection){
  unlist(mclapply(states_collection,function(m){
    (sum((posteriors_j)*(sapply(sequences_collection,function(x){
      gamma_i_m(m,x)})))+alphaMatrix[1,m]-1)/
        (sum(posteriors_j)+sum(alphaMatrix[1,]-1))}))
}

# Helper for M-step: calculate new transition probability matrix
modify_transition_matrix<-function(posteriors_j,sequences_collection,
              alphaMatrix,states_collection,N){
  newmatrix<- mat.or.vec(nr=length(states_collection),nc=length(states_collection
    ))
  for (ii in c(1:dim(newmatrix)[1]))    {
    newmatrix[ii,]<-unlist(lapply(c(1:dim(newmatrix)[2]), function(jj){
      numerator_part1=sum((posteriors_j)*(sapply(sequences_collection,
              function(x){delta_i_n_m(ii,jj,x)} )))
      numerator_part2=alphaMatrix[ii+1,jj]-1
      denominator_part1=sum(sapply(c(1:N),function(i){posteriors_j[i]*(sum(
        sapply(c(1:length(states)),function(m){
            delta_i_n_m(ii,m,sequences_collection[[i]])})
        ))}))
      denominator_part2=sum(alphaMatrix[ii+1,]-1)
      newmatrix[ii,jj]<-(numerator_part1+numerator_part2)/
              (denominator_part1+denominator_part2)
          }))
  }
  newmatrix
}

# Helper to calculate new TPM for M-step using parallel processing
modify_transition_matrix_foreach<-function(posteriors_j,sequences_collection,
              alphaMatrix,states_collection,N){
  newmatrix<- mat.or.vec(nr=length(states_collection),nc=length(states_collection
    ))
  newmatrix<-foreach(ii=c(1:dim(newmatrix)[1]),.combine='rbind') %dopar%  {
    unlist(mclapply(c(1:dim(newmatrix)[2]), function(jj) {
      numerator_part1=sum((posteriors_j)*(sapply(sequences_collection,
              function(x){delta_i_n_m(ii,jj,x)} )))
      numerator_part2=alphaMatrix[ii+1,jj]-1
      denominator_part1=sum(sapply(c(1:N),function(i){posteriors_j[i]*(sum(
              sapply(c(1:length(states)),function(m){
                delta_i_n_m(ii,m,sequences_collection[[i]])})
        ))}))
      denominator_part2=sum(alphaMatrix[ii+1,]-1)
      res<-(numerator_part1+numerator_part2)/(denominator_part1+denominator_part2
    )
          }))
  }
  newmatrix
}

# Implementation of M-step of EM-algorithm
m_step<-function(posteriors,sequence_collection,alphaMatrix,modelK,states_
    collection,K,N){
  newmodelK<-modelK
  for (j in c(1:K)){
    newmodelK[[j]]$marginal.prob<-(sum(posteriors[[j]]))/
      (length(sequence_collection))
    newmodelK[[j]]$start.vector<-modify_start_vector_j(
      posteriors[[j]],sequence_collection,alphaMatrix,states_collection)
    newmodelK[[j]]$transition.matrix<-modify_transition_matrix(
      posteriors[[j]],sequence_collection,alphaMatrix,states_collection,N)
  }
  newmodelK
}

# Implementation of M-step of EM-algorithm using parallel processing
m_step_par<-function(posteriors,sequence_collection,alphaMatrix,modelK,states_collection,K,N){
  newmodelK<-modelK
  newmodelK<-mclapply(c(1:K),function(j){
    marginal.prob<-(sum(posteriors[[j]]))/(length(sequence_collection))
    start.vector<-modify_start_vector_j(posteriors[[j]],sequence_collection,
      alphaMatrix,states_collection)
    transition.matrix<-modify_transition_matrix_foreach(posteriors[[j]],
      sequence_collection,alphaMatrix,states_collection,N)
    newmodelK[[j]]<-initModelComponent(transition.matrix,start.vector,marginal.prob)
  })
  newmodelK
}

# Implementation of the iterative EM-algorithm with K components Markov mixture
emK<-function(seq_collection,modelK,states,alphaMatrix) {
  map_likl<-map_log_likelihood_function(modelK,seq_collection,alphaMatrix,states)
  print("original map log likelihood")
  print(map_likl)
  iterations<-0
  eps<-10^(-17)
  dlta<-1
  while(dlta>=eps&iterations<15) {
  print("iteration:")
  print(iterations)
  z_posteriors<-e_step_z_i_j_t(modelK,seq_collection,states)
  modelK_new<-m_step(z_posteriors,seq_collection,alphaMatrix,modelK,states,
    length(modelK),length(seq_collection))
  map_likl_new<-map_log_likelihood_function(modelK_new,seq_collection,alphaMatrix,states)
  if (map_likl!=-Inf&&map_likl_new!=-Inf) {dlta<-map_likl_new-map_likl}
  modelK<-modelK_new
  modelK_new<-NULL
  map_likl<-map_likl_new
  map_likl_new<-NULL
  print("new map log-likelihood:")
  print(map_likl)
  iterations<-iterations+1
  }
  res<-list(estimate=map_likl,modelK,delta=dlta,iterations=iterations)
}

```

```
# Implementation of the EM-algorithm using parallel processing
emK_par<-function(seq_collection,modelK,states,alphaMatrix) {
  map_likl<-map_log_likelihood_function(modelK,seq_collection,alphaMatrix,states)
  print("original map log likelihood")
  print(map_likl)
  iterations<-0
  eps<-10^(-17)
  dlta<-1
  # perform  1 iteration
  while(dlta>=eps&iterations<20) {
  print("iteration:")
  print(iterations)
  z_posteriors<-estep_par(modelK,seq_collection,states)
  modelK_new<-m_step_par(z_posteriors,seq_collection,alphaMatrix,modelK,
    states,length(modelK),length(seq_collection))
  map_likl_new<-map_log_likelihood_function(modelK_new,seq_collection,alphaMatrix
    ,states)
  if (map_likl!=-Inf&&map_likl_new!=-Inf) {dlta<-map_likl_new-map_likl}
  modelK<-modelK_new
  modelK_new<-NULL
  map_likl<-map_likl_new
  map_likl_new<-NULL
  print("new map log-likelihood:")
  print(map_likl)
  iterations<-iterations+1
  }
  res<-list(estimate=map_likl,modelK,delta=dlta,iterations=iterations)
}
```

./imm_part2.R

```
#
# Partial_EM.R
# This file contains functions implementing partial EM-algorithm
# by means of iterative EM algorithm
# Author: Natalya Furmanova, TUHH, #21044449
# 10/2013
#

source('Incremental_Markov_Mixture_Modeling.R')
library(parallel)
library(foreach)
library(doMC)

# Partial MAP log-likelihood function
partial_map_function<-function(modelK,newComp,alphaMatrix,
          seq_collection,states){
  map<-sum(log(sapply(seq_collection, function(x){
    p1<-p_seqi_modj_for_model(x,newComp,states)
    p1<-ifelse(p1==0,1e-321,p1)
    res<-(1-newComp$marginal.prob)*likelihood_function_one_seq(
        x,modelK,states)+newComp$marginal.prob*p1
        })))+sum(log(sapply(c(1:dim(newComp$transition.matrix)[1]),
        function(x){dirichletPrior(newComp$transition.matrix[x,],
          alphaMatrix[x+1,])})))+
        log(dirichletPrior(newComp$start.vector,alphaMatrix[1,]))
 }

# Partial E-step (calculate conditional posterior for one component)
posterior_newcomp<-function(modelK,seq_collection,newComp,states){
  numerators<-sapply(seq_collection,function(x){
          newComp$marginal.prob*p_seqi_modj_for_model(x,newComp,states)})
  denom<-sapply(seq_collection, function(x){
          (1-newComp$marginal.prob)*likelihood_function_one_seq(
          x,modelK,states)+newComp$marginal.prob*p_seqi_modj_for_model(
          x,newComp,states)})
  numerators[which(numerators==0)]=1e-321
  denom[which(denom==0)]=1e-321
  posteriors_newcomp<-(numerators)/(denom)
 }

# Partial E-step using parallel processing
 posterior_newcomp_par<-function(logLikelihoodFunctionK,seq_collection,newComp,
                states){
  numerators<-unlist(mclapply(seq_collection,function(x){
    pp<-p_seqi_modj_for_model(x,newComp,states)
    newComp$marginal.prob*ifelse(pp==0,1e-321,pp)
    })
    )
  denominators<-unlist(mclapply(seq_collection, function(x){
    temp<-(1-newComp$marginal.prob)*logLikelihoodFunctionK
      +newComp$marginal.prob*p_seqi_modj_for_model(x,newComp,sd)
    ifelse(temp==0,1e-321,temp)
    })
    )
  posteriors_newcomp<-(numerators)/(denominators)
 }

# Partial EM - iterative process
partial_em<-function(modelK,newComp,sequences,states,alphaMatrix,model_length){
  eps<-1e-17
  dlta<-1
  while(dlta>eps){
  map_likl<-partial_map_function(modelK,newComp,alphamatrix,sequences,states)
  print(map_likl)
  newposter<-posterior_newcomp(modelK,sequences,newcomp,states)
  print(newposter)
  newCompnew<-newComp
  newCompnew$marginal.prob<-sum(newposter)/(length(sequences))
    newCompnew$start.vector<-modify_start_vector_j(newposter,
                sequences,alphaMatrix,states)
    newCompnew$transition.matrix<-modify_transition_matrix(newposter,
                sequences,alphaMatrix,states,length(sequences))
  print(newCompnew$transition.matrix)
  map_likl_new<-partial_map_function(modelK,newCompnew,alphaMatrix,sequences,
    states)
  dlta<-map_likl_new-map_likl
  newComp<-newCompnew
  }
 new_marginal_prob<-1-newComp$marginal.prob
 for(i in c(1:model_length)) {
   modelK[[i]]$marginal.prob<-new_marginal_prob*modelK[[i]]$marginal.prob
   }
 modelK[[model_length+1]]<-newComp
 res<-list(modelK,estimate=map_likl_new,dlta)
}
```

./Partial_EM.R

Bibliography

[1] David R. Brillinger. *Time series*. Holt, Rinehart and Winston, Inc., New York, 1975. Data analysis and theory, International Series in Decision Processes.

[2] Natalya Furmanova. Data mining in crm. Project work, TUHH, 2013.

[3] Jr. Bayardo, R.J., R. Agrawal, and D. Gunopulos. Constraint-based rule mining in large, dense databases. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 188–197, 1999. doi: 10.1109/ICDE.1999.754924.

[4] Shi Na, Liu Xumin, and Guan Yong. Research on k-means clustering algorithm: An improved k-means clustering algorithm. In *Intelligent Information Technology and Security Informatics (IITSI), 2010 Third International Symposium on*, pages 63–67, 2010. doi: 10.1109/IITSI.2010.74.

[5] Philippe Esling and Carlos Agon. Time-series data mining. *ACM Comput. Surv.*, 45(1):12:1–12:34, December 2012. ISSN 0360-0300. doi: 10.1145/2379776.2379788. URL http://doi.acm.org/10.1145/2379776.2379788.

[6] Fabian Mörchen. *Time series knowlegde mining*. PhD thesis, Marburg University, 2006.

[7] Jan G. De Gooijer and Rob J. Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443 – 473, 2006. ISSN 0169-2070. doi: http://dx.doi.org/10.1016/j.ijforecast.2006.01.001. URL http://www.sciencedirect.com/science/article/pii/S0169207006000021. ¡ce:title¿Twenty five years of forecasting¡/ce:title¿.

[8] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning*,

ICML '06, pages 1033–1040, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143974. URL http://doi.acm.org/10.1145/1143844.1143974.

[9] Eamonn J. Keogh, Selina Chu, David Hart, and Michael J. Pazzani. An online algorithm for segmenting time series. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pages 289–296, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1119-8. URL http://dl.acm.org/citation.cfm?id=645496.657889.

[10] Sam Blasiak and Huzefa Rangwala. A hidden markov model variant for sequence classification. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*, pages 1192–1197. AAAI Press, 2011.

[11] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowl. Inf. Syst.*, 7(3):358–386, March 2005. ISSN 0219-1377. doi: 10.1007/s10115-004-0154-9. URL http://dx.doi.org/10.1007/s10115-004-0154-9.

[12] Chotirat (Ann) Ratanamahatana and Eamonn J. Keogh. Three myths about dynamic time warping data mining. In *SDM*, 2005.

[13] Huizhi Yang and Jianguo Shi. A three-stage svm ensemble algorithm for chaotic time series prediction. In *Education Technology and Computer Science (ETCS), 2010 Second International Workshop on*, volume 3, pages 345–347, 2010. doi: 10.1109/ETCS.2010.477.

[14] G. Guimaraes. Temporal knowledge discovery for multivariate time series with enhanced self-organizing maps. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 6, pages 165–170 vol.6, 2000. doi: 10.1109/IJCNN.2000.859391.

[15] Shaozhi Wu, Yue Wu, Ying Wang, and Yalan Ye. An algorithm for time series data mining based on clustering. In *Communications, Circuits and Systems Proceedings, 2006 International Conference on*, volume 3, pages 2155–2158, 2006. doi: 10.1109/ICCCAS.2006.284925.

[16] Kin-Pong Chan and A.W.-C. Fu. Efficient time series matching by wavelets. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 126–133, 1999. doi: 10.1109/ICDE.1999.754915.

[17] Yiming Yang, Qiang Yang, Wei Lu, Jialin Pan, Rong Pan, Chenhui Lu, Lei Li, and Zhenxing Qin. Preprocessing time series data for classification with application to crm. In *Proceedings of the 18th Australian Joint conference on Advances in Artificial Intelligence*, AI'05, pages 133–142, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-30462-2, 978-3-540-30462-3. doi: 10.1007/11589990_16. URL http://dx.doi.org/10.1007/11589990_16.

[18] Michael Berry and Gordon Linoff. *Mastering Data Mining: The Art and Science of Customer Relationship Management*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1999. ISBN 0471331236.

[19] Paul W. Farris, Neil T. Bendle, Phillip E. Pfeifer, and David J. Reibstein. *Marketing Metrics: The Definitive Guide to Measuring Marketing Performance*. Wharton School Publishing, 2nd edition, 2010. ISBN 0137058292, 9780137058297.

[20] Dirk Van Den Poel and Bart Lariviere. Customer attrition analysis for financial services using proportional hazard models. *Journal of Operational Research*, 157:196–217, 2003.

[21] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: an introduction to cluster analysis*. Wiley, 1990.

[22] CharuC. Aggarwal, Alexander Hinneburg, and DanielA. Keim. On the surprising behavior of distance metrics in high dimensional space. In Jan Bussche and Victor Vianu, editors, *Database Theory CDT 2001*, volume 1973 of *Lecture Notes in Computer Science*, pages 420–434. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-41456-8. doi: 10.1007/3-540-44503-X_27. URL http://dx.doi.org/10.1007/3-540-44503-X_27.

[23] James C. Bezdek, Richard J. Hathaway, and Jacalyn M. Huband. Visual assessment of clustering tendency for rectangular dissimilarity matrices. *IEEE T. Fuzzy Systems*, 15(5):890–903, 2007. URL http://dblp.uni-trier.de/db/journals/tfs/tfs15.html#BezdekHH07.

[24] Anne-Laure Boulesteix, Silke Janitza, Jochen Kruppa, and Inke R. Koenig. Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics. *Wiley Int. Rev. Data Min. and Knowl. Disc.*, 2(6):493–507, November 2012. ISSN 1942-4787. doi: 10.1002/widm.1072. URL http://dx.doi.org/10.1002/widm.1072.

[25] Zhengzheng Xing, Jian Pei, Philip S. Yu, and Ke Wang. Extracting interpretable features for early classification on time series. In *SDM*, pages 247–258, 2011.

[26] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002. URL `http://CRAN.R-project.org/doc/Rnews/`.

[27] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

[28] A Method for Comparing Multivariate Time Series with Different Dimensions. *PLoS ONE*, 8(2):e54201+, February 2013. doi: 10.1371/journal.pone.0054201. URL `http://dx.doi.org/10.1371/journal.pone.0054201`.

[29] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. ISSN 00359246. URL `http://dx.doi.org/10.2307/2984875`.

[30] Andreas Kakoliris and Konstantinos Blekas. Incremental training of markov mixture models. *Knowledge Discovery from Data Streams*, page 47.