



Universiteit Leiden

Opleiding Informatica

Sequence Prediction
under multiple constraints

Name: Nikolaos Papathanasiou
Date: 27/05/2016
1st supervisor: Dr. Wojtek Kowalczyk
second supervisor: Prof. Dr. Thomas Bäck

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

ACKNOWLEDGEMENTS

I would like to start by thanking my second thesis supervisor Prof. Dr. Thomas Bäck from Leiden Institute of Advanced Computer Science (LIACS) for helping me get in contact with the company and providing me the opportunity to work on this thesis project. He supported me during the start of my project with any problems I encountered considering the arrangements that were necessary to be done between the company, me and the university.

I would like to thank my first supervisor Dr. Wojtek Kowalczyk from LIACS for his amazing assistance throughout the whole project. He always answered any questions very fast by email, even in late hours, and he arranged meetings and checked my progress, at least once per two weeks, while I knew he had a very strict and busy schedule. Our meetings have always proved really helpful and improved my understanding over the problem and the possible directions which could result into a satisfactory solution. His directions were always pointing on widening my knowledge over each specific solution or stimulating my thinking by smart and quick reading suggestions.

I want to thank also my internship supervisor Saskia Groenewegen from the R&D department of the INDG company for her assistance in any company-related issue that I had. She also gave me a lot of useful advice while I was implementing possible solutions, helped me understand the goals of the company in relation with my project and she was also the bridge of communication between me, project managers, the head developers and the CTO (head of R&D and my project) when needed.

I would also like to thank the HR Manager, Lisette Meerman from the Human Resources department of the INDG company for giving me the opportunity for an interview with the company and finally Daniel Haveman the CTO of INDG who believed in me and assigned to me this interesting project, which led to a working prototype.

Contents

1	Introduction	4
2	Problem Statement	5
3	Requirements	7
4	Related Work	7
4.1	Frequent Itemset Mining for Association Rules	8
4.2	Terminology	8
4.2.1	Frequent itemset Mining	8
4.2.2	Association Rules	10
4.2.3	Taxonomy	11
4.3	Markov Chain Prediction Model	12
4.3.1	Transition Matrix of higher order	13
4.4	Collaborative Filtering and Recommender Systems	13
4.4.1	Distance/Similarity Measure	13
4.4.2	Similarity Matrix	14
4.4.3	Item Based Recommendation	15
4.5	Compact Prediction Tree	15
4.5.1	Data Structures	16
5	Implementations	17
5.1	Association Rule Mining	18
5.1.1	Frequent Itemsets and Association Rules Generation	18
5.1.2	Prediction with Association Rules	19
5.2	Markov Chain	20
5.2.1	Markov Chain Model building Procedure	20
5.2.2	Markov Chain Extension	23
5.2.3	Prediction With Markov Chain	24
5.3	Item Based Recommender	25
5.3.1	Preprocessing	25
5.3.2	Building Item Based Recommender	26
5.3.3	Prediction Choice	26
5.4	Compact Prediction Tree	27
5.4.1	Building of CPT	27
5.4.2	Data Structures Storage	28
5.4.3	Prediction	29
6	Datasets	30
7	Experiments	31
7.1	10-fold Cross Validation	32
7.2	Evaluation Method	32
7.2.1	First Order Markov Chain Extension and its effect	32
7.3	Outliers Removal and its effect	33
7.3.1	Validation of Outliers	33

7.3.2	Common Sense Strategy	33
7.3.3	Mean Accuracy over Different Methods Experiment	34
7.4	Prediction Times Experiment	35
7.4.1	CPU comparison for computation times analogy	35
7.4.2	Prediction Times	35
7.5	Model Loading Times Experiment	36
7.6	Model RAM Consumption Experiment	36
7.7	Model SSD Size Consumption Experiment	36
7.8	Building Times Experiment	37
8	Results	37
8.1	Markov Chain First Order (MarkovM)	37
8.2	Markov Chain Extension First Order(MarkovM1)	37
8.3	Item Based	38
8.4	Markov Combined (Markov12)	38
8.5	Markov Combined Extension (MarkovM12)	38
8.6	CPT	38
9	Future Work/Recommendations	39
10	Conclusion	39

Abstract

To enhance a home decoration application, we want to predict the next item a user will add to their virtual room. As the application runs locally on a smartphone, we need to balance several constraints where some of them are usually not considered for sequence prediction algorithms: high prediction accuracy, prediction time, model loading time and specific memory requirements. We compare several sequence prediction methods on common benchmark problems with a 10-fold evaluation scheme. As a result, we present a model that can handle a very large number of sequences and unique items, while providing the best accuracy, while satisfying our constraints.

1 Introduction

Nowadays a lot of sequential data is available, from companies or organizations. INDG company (<http://indg.com/>) is one of them and they would like to use their data for mining interesting patterns in order to be able to predict the behaviour of a random user. Sequential data consists of **sequences** which basically are ordered lists of elements. We know that given a sequence of elements and some historical data of previous sequences, of the same context, it is possible to predict or recommend the next element(s), at least to a certain degree of success. There is a range of different approaches that can be followed for solving this problem, but what remains unknown is which method is more suitable for adding this predictive functionality in an application without a server connection, considering some specific constraints. A smartphone device application has some limitations from which these constraints are emerging. In such case we have some requirements imposed on memory consumption, loading time (time required for the application to load the necessary files for prediction) and prediction time of a model. It is also preferred for a model to use as little SSD (sold state disk) space as possible. Additionally, when applying the model to fresh data the less amount of time possible in order to build it is required, as long as we do not affect the performance of the constraints and accuracy. The goal of this paper is to satisfy these constraints for an application owned by the company INDG, while producing an accurate prediction. The approach we followed for solving this problem was to try different implementations, suggest possible extensions and analyze their results following the aforementioned constraints. First we present our problem in detail in Section 2 and then analyze the requirements based on the constraints in Section 3. The related work is discussed in Section 4 and the implementations that were applied to the problem are presented in Section 5. We suggest an extension of the standard Markov Chain in Section 5.2.2, that achieves a high accuracy while satisfying the constraints. Furthermore, as we know that sequential data can change or evolve over time, we conclude that the model will have to be applied again to the new data. This resulted in a desire to be able to see how each algorithm is going to be affected by such changes. This is why we chose three more datasets which are shown in Section 6 and compared the algorithms on them with respect to the accuracy and the constraints in Section 7. In order to compare these algorithms, an evaluation scheme is required, the scheme that was followed is analyzed in Section 7.2. Finally, we know that it is very common for a user based data to have some noise. Therefore, we present an outliers detection solution for the company's dataset in Section 7.3. The outcome of the experimentations is analyzed for each algorithm in Section 8 and we recommend some possibilities for future work in Section 9.

2 Problem Statement

Amikasa (<http://amikasa.com/>) by INDG is a virtual room builder application that supports Windows and IOS devices. The user is able to build a virtual room (see Figure 1), and decorate it by choosing items from a catalogue of 3D graphic models of real items. We want to build a system that helps the users to choose their next item(s) by suggesting items which are more probable for them to choose. To do this, we formulate the problem as a sequence prediction task:

Given a catalogue of items $C = \{item_1, item_2, \dots, item_n\}$, a sequence of items (a room configuration) is denoted as $Seq = [c_1, c_2, \dots, c_m]$, where $c_i \in C$ and $1 \leq i \leq m$. The company has some historical data collected, in a form of sequences and the database is denoted as $D = \{Seq_1, Seq_2, \dots, Seq_s\}$ is the set of these historical sequences. We want to use these sequences in order to create a model to predict the next item, which a user could possibly place, based on what is already placed in the room at the moment (see Figure 4). We can see two examples of the desired functionality in Figure 2 where the input sequence is given and a sequence of ranked predictions is returned. In order to run this model in the mobile application's system, we need to have a model produced from the building procedure (see Figure 3) and then load it from the device's SSD to its RAM and use a prediction method in order to get a result (see Figure 4). As Amikasa is an application running on mobile phones and tablets, we have to consider several constraints: we want to minimize *RAM consumption*, *prediction time* and *model loading time*. We also want to maximize the *prediction accuracy* and minimize the *building time* and the *SSD space usage*. Even though the building time is not a critical constraint, it is an important aspect in the scope of our project in order to be able to re-build the model easily and quickly. The limited storage space of a smartphone device indicates that an application should have some limits, considering that it is not the only application installed. Moreover, the user can easily get discouraged by downloading a mobile application which is too large. To find the most suitable method for predicting items given those constraints, we have to compare several sequence prediction algorithms.



Figure 1: An example of a random virtual room configuration in Amikasa app



Figure 2: Two examples of sequence prediction. In the first row we have a long input sequence of items and we get 4 ranked predictions (we can choose the number of predictions). In the second row we have a short input sequence and we get 4 predictions.

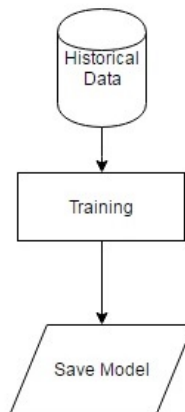


Figure 3: Building procedure of a sequence prediction model.

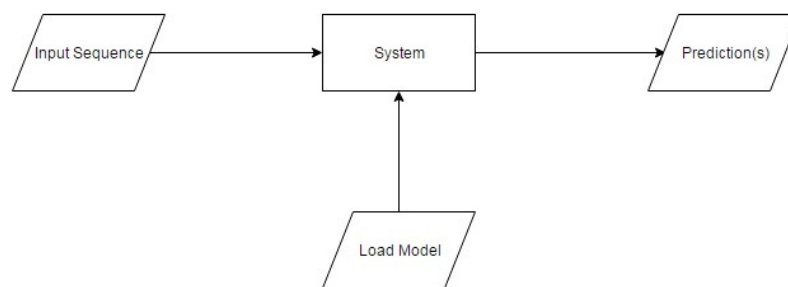


Figure 4: In this Figure the sequence prediction model is loaded to the Application's System, the system takes the input and uses the model to produces a prediction.

3 Requirements

At the very beginning of this project the company set some specific requirements that needed to be fulfilled in order to integrate the functionality of the sequence prediction into the application. The requirements were set based on the constraints that we mentioned in the previous section and the reason behind them is the overall smooth experience that the company seeks to provide to the user.

RAM Limitations: The baseline devices (minimum hardware support) for Amikasa are the iPhone 4S and iPad2. In IOS, depending on which device is running it, there is a limit on the RAM which an application can use; which usually is roughly 50% of the available RAM. The baseline devices have 512MB of RAM, that means that an application consuming more than 256 MB of RAM will crash. Assuming that a user is not going to use only one application at a time and as the application uses a lot of RAM due to the graphical component assets (3D graphics, textures etc.) the company has set the preferable limit in RAM consumption on **110 MB** for the whole application system. The **90%** of this limit is already fulfilled which leaves a maximum of **11MB** available for the predictive component of system.

Prediction Time: We call the time the system needs in order to calculate the prediction(s), prediction time. The virtual room builder application is built on a game engine and it consists of heavy a graphical component. Additionally, there is a desirable limit of 30 Frames Per Second (FPS) set by the company, which ensures the smooth running the application. During the moments where the graphical component is involved (drugging furniture, choosing items from catalogue etc.) the CPU usage is 100%. However, while graphical component is not involved the application basically uses roughly 25% of the CPU (on the baseline devices). Overall this creates a time window of maximum **13ms per frame** available for computation of the prediction. Assuming that a user will take at least half a second (0.5 s) before moving the object or selecting another one we have $15 \text{ FPS} * 13 \text{ ms} = 190 \text{ ms}$ available. Thus, the maximum limit for the baseline processor Apple A5 dual-core ARM Cortex-A9 CPU is set at **190 ms**.

Loading Time: Loading time is the time that the application needs in order to move the prediction model from the SSD of the smartphone to RAM. Loading of the graphic components and the environment of the virtual room builder already can take some seconds. The initial loading of the application (a fresh install) takes 7 seconds on the baseline devices. Moreover, if the user starts the application again, the loading time may be even longer, in case a configuration of a room is saved. For a user it matters a lot if they have to wait 6 seconds or 8 seconds for the application to become responsive, thus the prediction model should be able to be loaded from the application within some time limits. The company has set this loading time limit within **250-350 ms**.

4 Related Work

The problem of finding the next element(s) in a sequence is not limited to a specific methodology. It is possible to see the problem as a sequence prediction based on the previous inputs like Markov Chain [14], [15], [16] or as a recommender system like [6]. Another approach is the use of Association Rules (AR) as a prediction model [9] or use a novel lossless technique called Compact Prediction Tree [7]. In this section we provide an overview of the methods that were used in the experimentation and the implementation details are going to be

discussed in Section 5.

4.1 Frequent Itemset Mining for Association Rules

The initial scope of this method was to find interesting associations between elements of an itemset but it can also be interpreted as a prediction model. The methodology that is proposed by [10] consists of two parts: Frequent Itemset Mining (FIM) and Association Rules (AR) generation.

4.2 Terminology

Let a sequence of items be $S = [item_1, item_2, item_3]$, in order for the sequence to be an *itemset* it should not contain repetitions as an itemset is just a set of individual items. Then if S contain no repetitions then S is an itemset which have three *itemsubsets*, $S_1 = \{item_1, item_2\}$, $S_2 = \{item_1, item_3\}$, $S_3 = \{item_2, item_3\}$ and an *itemsuperset* of S can be $S_{super} = \{item_1, item_2, item_3, item_4\}$. An itemset's or an item's *support* denotes the frequency of a particular itemset or item in a given dataset. Then a support threshold can be set and used to prune itemsets below the threshold (which are considered infrequent) and keep the ones that are considered *frequent*. Finally, the algorithm finds all the frequent itemsupersets and *generates candidates* through their itemsubsets.

- **itemset** is a set of individual items.
- **itemsubset** is a subset of an itemset.
- **itemsuperset** is an itemset which includes another itemset or itemsets.
- **support** is the frequency of an item or itemset inside a database. A specific support value can be set as a threshold.
- **frequent itemset** is a set of individual items which has support higher than the user defined support threshold.
- **candidate generation** is the generation of subsets from frequent itemsupersets which produce itemsets that are potentially frequent.

4.2.1 Frequent itemset Mining

Frequent Itemset Mining (FIM) is a technique which is widely studied in the Association Rule research field. The main idea behind the FIM is based on the Apriori principle [1] which states that every subset of a frequent itemset also frequent. There are several algorithms for FIM and most of them are producing similar results in the sense of frequent itemsets generated, while they usually have different run times and ways for candidate generation. An example of a dataset containing itemsets is shown in Table 1 and the support value of each item is shown in Figure 2. There are three different types of frequent itemsets that can be produced by a FIM algorithm standard, closed and maximal (Fig 5):

- Standard type describes all the itemsets that are frequent.
- A frequent itemset is **closed** if there exists no itemsuperset that has the same support count as this original itemset. [3]

- A frequent itemset is a **maximal** if it is a frequent itemset and no super-itemset of this is a frequent itemset. [2]

Table 1: This is an example database of itemsets

TID	itemset
1	{b,a}
2	{c,b,d}
3	{a,d,c,e}
4	{e,a,d}
5	{b,c,a}
6	{b,c,a,d}
7	{a,d,b}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

Table 2: Support of the items that are contained in the example DB of the Table 1

item	support
a	8
b	8
c	6
d	6
e	3

It is very common for a FIM algorithm to produce a very high number of frequent itemsets. Thus in some cases where the number of items is very big, it would make sense to use a smaller number of FIs like closed or maximal. The final product of FIM procedure is a list of frequent itemsets storing a support value for each itemset. Then we can use these frequent itemsets as an input for the Association Rule procedure. An example of the output given the example database seen in Table 1 is shown in Table 3 where the itemsets having support lower than 2 are pruned.

Table 3: Result of FIM with support threshold = 2 on the example DB of the Table 1

frequent itemset	support
{b,a}	6
{c,b}	5
{d,a}	5
{c,a}	4
{d,b}	4
{c,b,a}	3
{d,b,a}	3
{d,c}	3
{d,c,a}	2
{d,c,b}	2
{e,a}	2
{e,c}	2
{e,d,a}	2
{e,d}	2

Maximal vs Closed Itemsets

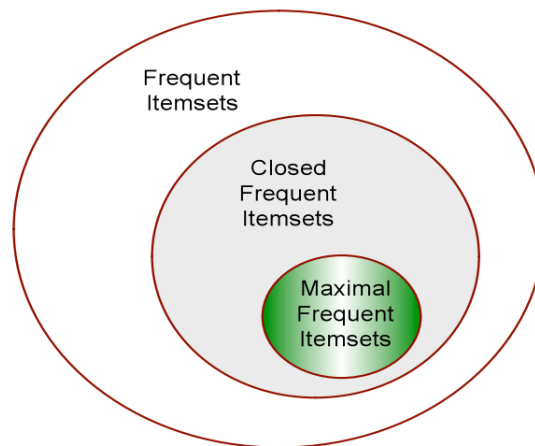


Figure 5: In this figure we can see the connection between the different types of Frequent Itemsets. It is apparent that maximal FIs are a subset of closed FIs which are also a subset of the standard FIs

4.2.2 Association Rules

After the frequent itemsets coupled over a support threshold are produced from the FIM procedure, the next step is the generation of Association Rules. The mining of Association Rules is based on the affinity analysis which discovers associations between specific groups of items (itemsets) or individual items from a catalogue. Thus we want to find interesting associations between our catalogue's available items based on the historical data available. An

association rule consists of three different parts: an *antecedent*, a *consequent* and *confidence*. In order to calculate the confidence of a rule, we use the following formula:

$$confidence = \frac{support(antecedent \cup consequent)}{support(antecedent)} \quad (1)$$

This formula captures the ratio between the number of occurrences of the antecedent and each consequent and provides an insight into how confident we are to use this rule. The closer the value is to 1 the higher confidence we have. Let an itemset $X = \{a, b, c\}$ with support $s_x = 56$ be the antecedent and itemset $Y = \{c, d\}$ be the consequent of a rule, the union of the two itemsets has support $s_u = 70$, we then calculate the importance score $conf = s_x/s_u = 1.25$ which represents the significance of the rule. Thus the general form of an association rule is $X \rightarrow Y : confidence$.

Example: To follow our examples from Section 4.2.1, we show a sample of 10 association rules that are produced among others from FIM procedure based on the frequent itemsets of the Table 3. If we have an input containing only an item $\{b\}$ and we want to predict the next item we need to find all the rules containing the antecedent b and calculate the confidence of each rule. Then we consider which of the values is closest to 1, thus the choice for a prediction from the rules in Table 4, for item b as an input, would be the consequent $\{a\}$ as it has confidence 0.75 on the example dataset.

Table 4: In this table we can see the a sample of the rules produced based on the frequent itemsets of Table 3. The support of the rule shows how frequent a rule is and the confidence shows how accurate a rule is.

antecedent	consequent	support	confidence
{c,b}	{a}	3	0.60
{d,b}	{a}	3	0.75
{d,a}	{c}	2	0.40
{d,c}	{b}	2	0.66
{e,d}	{a}	2	1.00
{b}	{a}	6	0.75
{b}	{c}	5	0.62
{b}	{d}	3	0.37
{a}	{e}	2	0.25

4.2.3 Taxonomy

One variation/modification of the standard algorithms is to add a *taxonomy* or hierarchy on the items, so, in case an item from the leaves of the taxonomy (see Figure 6) is not *frequent*, it can be swapped with its *parent*. This technique can be proved helpful, because it produces more general Association Rules [5]. By generality of ARs we mean that we can have specific levels in the rules like $chair(dinning(brand.1)) \rightarrow table(dinning)$ and, as we go higher in the tree levels, the rules produced have a more general sense for example: $chair(dining) \rightarrow table$. There are two directions of this modification: in the first case it is proposed to add the parents of each item in each transaction (even if the items in the transaction share the same father, the father should be added only once) and then apply

any of the algorithms available for FIM and produce these general rules; in the second case we can run any of the algorithms to each of the different levels of the taxonomy and produce level specific rules. [5].

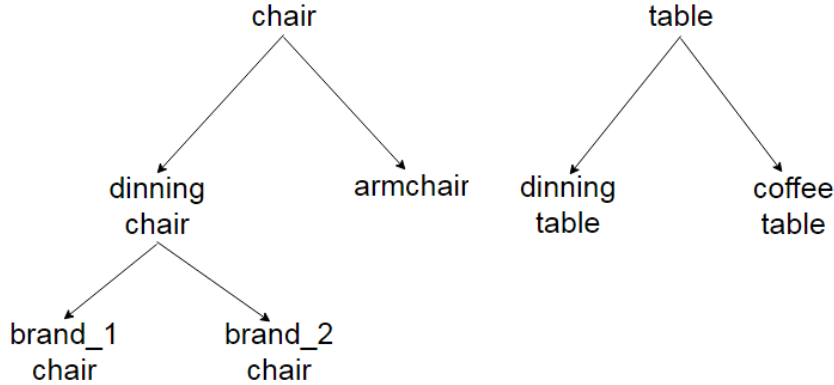


Figure 6: Example of a taxonomy from the company’s dataset.

The problem that is possible to be encountered in the case of using the taxonomy is that this way the dimensionality and also the sparsity of the sequences is increased exponentially and the candidate generation can get more computationally expensive and memory consuming.

4.3 Markov Chain Prediction Model

The key idea behind Markov Chain is the assumption that the probability p of the occurrence of the *next element* in a sequence depends only on the *current element*. If s_i is the next element of a sequence S , then the probability of the occurrence of the next element s_i is $p(s_i|s_1, \dots, s_{i-1}) = p(s_i|s_{i-1})$. By calculating the probabilities of each possible transition between items we can create a *transition matrix*. The transition matrix contains the maximum likelihood estimates of the probabilities. If we have 5 available items in a catalogue $C = a, b, c, d, e$ and n_{ij} is the occurrence of the transition from item i to item j , in order to estimate the probability of n_{ij} we use Formula 2.

$$\hat{p}_{ij} = \frac{n_{ij}}{\sum_{s=1}^5 n_{is}} \quad (2)$$

The transition matrix then can be used as a model for sequence prediction by finding the biggest probability of the next occurrence of an input item. A transition matrix can be of different orders: in the first order Markov Chain we only get the combination of single items and the transition probability between them, in Markov Chain of orders $n > 1$ we get the transition probability between a sequence of length n any of the available items.

	a	b	c	d	e
a	0	0.333	0	0.666	0
b	0.142	0	0.571	0.285	0
c	0.400	0.200	0	0	0.400
d	0	0.500	0.500	0	0
e	1.000	0	0	0	0

Table 5: Transition matrix based on the database from example in Table 1. We can see that it is possible to have zero occurrences of a transition between some items. In such cases we can use the Laplace Estimator which is explained in detail in Section 5.2.1.

4.3.1 Transition Matrix of higher order

The concept of a higher order Markov Chain is very appealing, especially when at least in theory the combination of transition matrices of different orders can give a better accuracy. Unfortunately, in order to calculate the occurrences of the transitions for order two we need all possible combinations of items and in order to calculate the transitions between a pair and an item. The procedure for higher order obviously brings even more combinatorial complexity (all possible combinations of combinations and single items for order 3 etc.). One possible solution is the way All Kth Order Markov model (AKOM) by [12] works which uses *selective Markov Chains* of different orders.

4.4 Collaborative Filtering and Recommender Systems

Collaborative Filtering (CF) is a well known method, which is applied in many commercial services like NetFlix or Amazon, for proposing items or movies to a client, based on their previous choices. Even if it is not a sequence prediction algorithm, the CF methodology could be followed in order to produce ‘predictions’. We know that with CF techniques it is possible to capture the similarity between the users/transactions or the items themselves, as long as a distance or similarity measure between them exists. There are two basic types of collaborative filtering techniques: the *User-based* and *Item-based*. User-based method can either be used for predicting a specific value of rating or recommending individual items based on user similarity. Item-based CF recommends items based on the precomputed similarities between the items with the help of a similarity measure. The first variation of the CF techniques is not considered as a viable solution for our problem, as the computation that takes place for calculating the distances between the historical data and the input is online. Thus the method was not created to separately save and load the model as it is required by the company’s system specifications. Therefore the method is not optimized to store a model containing the distances between data points in an efficient and easily accessible way. Nevertheless, a similar to the User-based technique, in the aspect of finding similar users or transactions, called Compact Prediction Tree (Section 4.5) is applied to the problem. This is a specialized case because with the use of some specific data structures and binary computations, it proves to be faster than a normal CF User-based recommender.

4.4.1 Distance/Similarity Measure

Usually a representation of the data points having the same length is needed. Then it is possible to measure the distances between an input transaction and all the data points, by using a similarity measure. There are numerous similarity measures that can be used, but for the purpose of this paper the following measures were the ones that we tried:

Hamming Similarity:

In order to calculate the Hamming Distance between two binary vectors, we need to calculate the XOR between them and then count the number of ones. The number then gets normalized by dividing by the length of the vectors n . Finally in order to get the similarity we subtract the hamming distance from one.

$$HamSim = 1 - \frac{\sum_{i=0}^n |Vector_{1i} - Vector_{2i}|}{n} \quad (3)$$

Cosine Similarity:

For the Cosine Similarity calculation includes the dot product of the two vectors divided by the product of the magnitudes of the two vectors. Dot product between two vectors is denoted by $(Vec_1 \cdot Vec_2)$.

$$CosSim = \frac{(Vector_1 \cdot Vector_2)}{\|Vector_1\| \|Vector_2\|} \quad (4)$$

Pearson Correlation Similarity:

In order to calculate Pearson Correlation Similarity we need the covariance of the two vectors divided by the product of their standard deviations. Let n be the length of the vectors and \bar{x} , \bar{y} be the mean of the elements of the two different vectors \vec{X} and \vec{Y} , which we want to calculate the similarity between them. Then Pearson Correlation then is calculated by:

$$PearsonSim = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (5)$$

4.4.2 Similarity Matrix

In order to store the similarity between items we need to create a *similarity matrix*. The similarity matrix is a symmetric matrix stores and is created by calculating the distances of each item with all the others. In order to create we usually preprocess the data into a representation that is needed for the similarity calculation an example of this representation is given in Table 6. Then we use this data representation to calculate the similarity between items by comparing the columns (see Figure 7) that represent each item with all the other columns and store the similarities into the similarity matrix.

TID	a	b	c	d	e
1	1	0	0	0	0
2	0	1	1	1	0
3	1	0	1	1	1
4	1	0	0	1	1
5	1	1	1	0	0
6	1	1	1	1	0
7	1	1	0	1	0
8	1	1	1	0	0
9	1	1	0	1	0
10	0	1	1	0	1

Table 6: Preprocessed representation of the database from Table 1

TID	a	b	c	d	e
1	1	0	0	0	0
2	0	1	1	1	0
3	1	0	1	1	1
4	1	0	0	1	1
5	1	1	1	0	0
6	1	1	1	1	0
7	1	1	0	1	0
8	1	1	1	0	0
9	1	1	0	1	0
10	0	1	1	0	1

Figure 7: In order to calculate the similarity between item b and item d in this example we calculate the similarity between the two column vectors representing these items.

4.4.3 Item Based Recommendation

From the CF techniques the method that is possible to be applied to our problem, as it contains an offline step, is the Item-based recommender. In this case rather than matching similar users, we try to match the user’s already selected items with other similar items to them [6] and try to produce a recommendation list. The process is very simple but computationally expensive and the time complexity grows exponentially as the data set size and number of individual items (catalogue) grows. The advantage of this technique is that if m is the number of data points, then a similarity matrix of size $m \times m$ is created that stores the distances between items and can be loaded as a prediction model. The disadvantage is that, while the number of items is increasing the time needed to re apply the model to fresh data is increasing exponentially.

4.5 Compact Prediction Tree

One slightly different technique was presented by Ted Gueniche , Philippe Fournier-Viger , and Vincent S. Tseng [7] as a *lossless* method for sequence prediction. Lossless prediction means that the algorithm has the capability of considering the whole input sequence length for calculating a prediction. The way this method works is very similar to the usual User-based CF (k-NN), but instead of calculating distances between the input and the whole dataset, we have 3 smart data structures that help to improve a lot the computation time

and memory consumption. An extra step of work is needed in order to apply this technique in our problem. We need to store these 3 data structures and use them as the model that is loaded to the application's system.

4.5.1 Data Structures

The first data structure the *Prediction Tree* (PT) is representing/compressing the dataset by using a simple N-ary tree structure which is known as *trie*. In the trie data structure each node can have an arbitrary number of children, one parent, the current node's value and a count variable. Each node's value represents an element and each branch represents a sequence. No ordering or preprocessing is needed, as the process is being included in the creation of the data structures. An example of the PT is shown in Figure 8.

The second data structure is the *Inverted Index* (II). As we can see in Figure 8 II can be seen as matrix, where each row represents all the sequences contained in the dataset and the columns represent all possible items. Thus, if n is the number of all available unique items and m is the number of sequences of items contained in the dataset, we have a matrix of size $n \times m$. For each row of the matrix, representing an item, each element of the row contains the number 1 if the item is found inside a sequence and the number 0 if the item is not contained in a sequence (binary). This representation, similarly to the CF techniques, is helpful for fast computation of the similarity between historical sequences and the input sequence.

The third data structure is the *Lookup Table* (LT). The Lookup Table provides fast and easy access to each sequence in the dataset. It is basically an array with size equal to the length of the dataset. Each element in this array is pointing to the last node/element of the sequence in the tree, that is represented as the index in the LT. The connection between the PT and the LT can be seen in Figure 8.

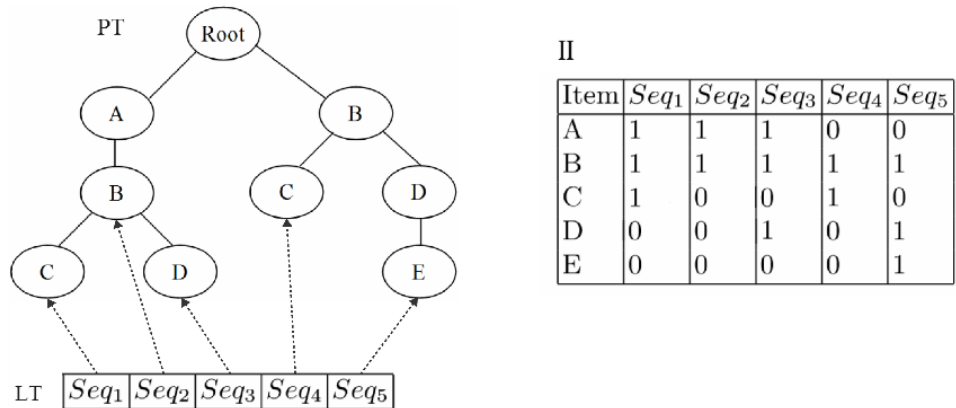


Figure 8: Compact Prediction Tree: Data Structures

We can see that the trie contains only an element in each node, each branch represents a sequence from the database. This tree representation compresses the dataset before making any calculations: if two or more sequences contain identical subparts they will only be included to the tree once. The Inverted Index stores the information of the sequences in a representation that will help to compute a distance between them in the most efficient way. Finally the Lookup Table stores a pointer of the last node of each branch in order to help accessing every sequence without actually storing them.

5 Implementations

The implementations that we are going to describe in this section were used for making an appropriate comparison between various algorithms, in order to satisfy the constraints to the fullest. For all the methods there are two parts implemented; the *building process* and the *prediction model*. The building process takes place as an offline computation outside of the application and all the methods were implemented and run in **Python 2.7.11**. The reason that Python was used is that the performance of the building procedure is not an issue of the constraints of the baseline devices and it is easy to install it and use on any computer. We only consider general building times in order to understand which method is the fastest. All the prediction model methods were implemented in **C# .NET 4.0**, with only LINQ library included, as the application itself is compiled in this language. For each model the parts that were used from other libraries are specified and the parts implemented are described by providing the related pseudocode. In order to gain some milliseconds on prediction time, we set a constant number of predictions for all prediction methods on three. By using the method shown in pseudocode 1, we find the 3 most probable items for prediction, efficiently. Not all data structures that are using this function are lists but we want to show the general idea behind the speed efficiency given from this method. Each different algorithm uses a modified *find_three_best* function towards the data structure that it uses.

Algorithm 1 find_three_best(list)

```
max1 = -1
max2 = -1
max3 = -1
for value in list do
  if value > max1 then
    max3 = max2
    max2 = max1
    max1 = value
  else
    if value > max2 then
      max3 = max2
      max2 = value
    else
      if value > max3 then
        max3 = value
      end if
    end if
  end if
end for
return max1, max2, max3
```

5.1 Association Rule Mining

For the building of this implementation a custom library for Python was used which is called *PyFim* and can be found in <http://www.borgelt.net/pyfim>. PyFIM contains all famous algorithms for FIM. Most of the algorithms are able to produce all three types of FIs, standard, closed, maximal, but for the final tests the FP-Growth algorithm was used, set to generate closed FIs.

5.1.1 Frequent Itemsets and Association Rules Generation

The procedure that was followed in order to build this prediction model includes first the generation of Frequent Itemsets and then the generation of Association Rules. In order to generate the Frequent Itemsets we apply the FP-Growth algorithm proposed by [18] on our sequential dataset. FP-Growth uses a N-ary tree data structure called *trie* in order to store all frequent itemsupersets from the database and then generates all possible frequent itemsets from their subsets.

We know that on the standard Association Rule methodology we extract associations between frequent itemsets of arbitrary length. In our case, during the Association Rules generation, we consider only associations from one or more items to one item because, this way we can use the rules for sequence prediction. In order to do that, we iterate through the Frequent Itemsets and choose each of the elements of the itemset as a consequent and the remaining itemset as antecedent. Then we calculate the importance measure, confidence, and store it along with each rule. A minimum confidence value can be given and used as a threshold in order to prune rules that empirically are considered not confident. The procedure can be seen in pseudocode 2.

Algorithm 2 Rule Generation

```
Require: Frequent Itemsets, minConf
Association Rules = Empty Pair List
for each FI in Frequent Itemsets do
  for each item in FI do
    consequent = item
    antecedent = FI.Except(item)
    rule = Pair of List and Double
    ruleList = [antecedent, consequent]
     $s_a$  = Support of antecedent
     $s_{fi}$  = Support of FI
    confidence =  $s_{fi}/s_a$ 
    rule.Key = ruleList
    rule.Value = confidence
    if confidence > minConf then
      Association Rules.Add(rule)
    end if
  end for
end for
return Association Rules
```

5.1.2 Prediction with Association Rules

Even if this method was not made for sequence prediction, it is possible to use it for sequence prediction. The prediction algorithm based on Association Rules is very simple: After the generation of the Association Rules is over and the rules are stored, the rules are loaded to the application system. The prediction algorithm takes as an input the user's current room configuration and finds all the antecedents of the association rules that matches this input. Then, it stores each consequent and its score to a list of pairs and returns as a prediction the consequent of the association rule with the highest importance score.

The list of pairs (Association Rules in the pseudocode 3) can be implemented as a list of tuples or a list of lists in Python. Both implementations support indexing such as if *rule* is a list/tuple storing an association rule and its corresponding importance score from the list of pairs, then *rule*[0] is the antecedent, *rule*[1] is the antecedent and *rule*[2] is the confidence of the rule.

Algorithm 3 Prediction for AR

```
Require: inputSeq, Association Rules
possiblePredictions = Empty Pair List
for rule in Association Rules do
  //if the input sequence is equal to any of the rule antecedents
  if inputSeq == rule[0] then
    pair = Empty Tuple
    pair[0] = rule[1] //Add consequent
    pair[1] = rule[2] //Add score
    possiblePredictions.Add(pair)
  end if
end for
predictions = find_three_best(possiblePredictions)
return predictions
```

5.2 Markov Chain

For the Markov Chain method implementation the simplest model possible, based on the constraints, was created by using a matrix data structure (list of lists in Python) for the transition matrix. In order to store the probabilities of all transitions from all possible states, we need to calculate the normalized occurrence between each of the states. After the building process in Python is finished the transition matrix is saved to a text file and it is loaded from the C# library, containing the prediction method.

5.2.1 Markov Chain Model building Procedure

The algorithm takes as an input the sequential dataset and iterate through each sequence in order to calculate the occurrence of each possible transition. For the normalization's sake we also need to store the occurrences of all individual items in the dataset (support value). This is why a dictionary data structure is used to store each individual item as a key and each support of the item as a value. Then the support of each item is used in order to normalize the occurrences of each row and produce each transition probability. The transition probability is produced by dividing the total count of each *next item* transition by the total number of the occurrences of each *current item*. The output of the building procedure is a transition matrix that can be used as a model for sequence prediction. A transition matrix as we explained in Section 4.3.1 can have different orders, for our dataset and our systems available it was possible to implement the process with order as high as two.

You can see the First Order Markov model building process in 4

Algorithm 4 Markov first Order Building Process

```
Require: dataset, catalogue
size = (length(catalogue), length(catalogue))
initialize t_matrix[size] of float zeros
index = 0
for index < length(catalogue) do
  item = catalogue[index]
  for sequence in dataset do
    if item in sequence then
      if item != last item then
        next_item = find_next_item(sequence, item)
        indexNext = find_index(catalogue, next_item)
        t_matrix[index][indexNext] = t_matrix[index][indexNext] + 1
      end if
    end if
  end for
  index = index + 1
end for
for row in t_matrix do
  temp_val = sum(row)
  rowIndex = 0
  for rowIndex < length(row) do
    row[index] = row[index] / temp_val
    rowIndex = rowIndex + 1
  end for
end for
return t_matrix
```

For the Second Order Markov Chain the implementation is similar, but instead of iterating through the catalogue, we need to iterate through the indexed pairs of all combinations of the catalogue called *couples* in pseudocode 5. We can ignore all the combinations of the same items in our case as the repetitions are not included in the dataset. The reason behind that decision is that the way the application works; after an item is selected, by a user, it remains there available to be selected again (until the user makes another choice). Thus it would be redundant to predict the same item.

The Second Order Markov model building procedure goes as follows:

Algorithm 5 Markov second Order Building Process

```
Require: dataset, catalogue
couples = all possible permutations of catalogue
mat_size = (length(catalogue), length(couples))
initialize t_matrix = matrix of float zeros with size mat_size
pairIndex = 0
for pairIndex < length(couples) do
  for sequence in dataset do
    if couples[pairIndex] exists in sequence then
      pairItem = couples[pairIndex][1] //second item in pair
      if pairItem != last item in sequence then
        next_item = find_next_item(sequence, pairItem)
        indexNext = find_index(catalogue, next_item)
        t_matrix[pairIndex][indexNext] = t_matrix[pairIndex][indexNext] + 1
      end if
    end if
  end for
  pairIndex = pairIndex + 1
end for
for row in t_matrix do
  temp_val = sum(row)
  for index of item in row do
    row[index] = row[index] / temp_val
  end for
end for
return t_matrix
```

The same logic can be applied for order $o > 2$, but the combinatorial complexity of the permutations is increasing exponentially.

Laplace Estimator


In case of transition matrices and especially for orders $o > 1$, we have a higher chance to have many zero occurrences. A smart tweak proposed by an acclaimed French mathematician of the 18th century, Pierre Laplace, is to add into these zero occurrences a small constant divided by the number of the available items [8] (Section 4.2 p. 91). Let a probability p be $p_i = x_i/y_i$, and in total we have n states thus n different probabilities for each state such as, $\sum_{i=0}^n p_i = 1$, then in order to replace zero probabilities we apply Formula 6:

$$p_i = \frac{x_i + \mu/n}{y_i + \mu} \quad (6)$$

In Formula 6 μ is a small value constant that is usually 1, but there is no restriction to a specific value and it is possible to experiment with it. In our final experiments we use the value $\mu = 1$.

Example: In order to understand how such process will affect the transition probabilities we can use the example from Section 4.3. In order to see the differences with Table 5 we show both transition matrices in Figure 9.

	a	b	c	d	e
a	0	0.333	0	0.666	0
b	0.142	0	0.571	0.285	0
c	0.400	0.200	0	0	0.400
d	0	0.500	0.500	0	0
e	1.000	0	0	0	0




	a	b	c	d	e
a	0.023	0.309	0.023	0.595	0.023
b	0.142	0.017	0.517	0.267	0.017
c	0.366	0.199	0.033	0.033	0.366
d	0.166	0.500	0.500	0.166	0.166
e	1.000	0.500	0.500	0.500	0.500

Figure 9: Here we can see the differences between the First Order Transition Matrix from Table 5 and the one build using the Laplace Estimator.

5.2.2 Markov Chain Extension

The limitation of the first order Markov chain is that it uses only the current item to predict the next one. The second order Markov chain uses both the current and the previous items to predict the next item, which usually leads to higher accuracy. However, the cost of this higher accuracy is the incrementation of the size of the model, from $O(n^2)$ to $O(n^3)$, where n denotes the number of items in a catalogue. In our project, where the model size must be highly limited, it makes the deployment of second order Markov models impossible. However, we found a simple extension of the first order Markov model which increases the accuracy, while preserving low RAM consumption. This extension is based on an observation that when a user decides which item should be put next to the room, he/she usually has several items in mind, and the order of selecting them is not relevant. For example, when a user wants to add a table followed by a chair, it doesn't matter if the system first suggests a table and then a chair, or first a chair and then a table. In this situation, both 'a table' and 'a chair' are correct suggestions that could be produced by the system. This observation leads to the following procedure for calculating an *extended first order Markov model of depth k* . Let D be a set of sequences over a catalog $C = item_1, \dots, item_n$. We define the probability of transition from $item_i$ to $item_j$, p_{ij} , as the ratio m_{ij}/n_i , where m_{ij} denotes the number of sequences in D , which contain $item_i$ that is followed by $item_j$ at one of the next k positions in the sequence (after $item_j$), and n_i denotes the number of sequences that contain $item_i$. Clearly, given this definition, one should modify the definition of model accuracy: when comparing predictions of a model to an actual sequence, we count a predicted item as correctly predicted if it occurs within the next $k = 5$ positions from the current state (correlationStep). In Figure 10 we can see the difference between the standard first order transition matrix (left) and the first order transition matrix after the extension, based on the example database from Table 1.

	a	b	c	d	e
a	0	0.333	0	0.666	0
b	0.142	0	0.571	0.285	0
c	0.400	0.200	0	0	0.400
d	0	0.500	0.500	0	0
e	1.000	0	0	0	0



	a	b	c	d	e
a	0	0.272	0.181	0.454	0.090
b	0.272	0	0.363	0.272	0.090
c	0.285	0.142	0	0.285	0.285
d	0	0.333	0.333	0.0	0.333
e	0.500	0	0	0.500	0

Figure 10: Here we can see the differences between the First Order Transition Matrix from Table 5 and the one build using the the extension proposed in Section 5.2.2.

We follow this procedure for sequence 'windows' of length *correlationStep* which value should be found empirically for every sequence. The result of such procedure will change the transition probabilities in a way that its possible for the model to produce a better accuracy while still containing the simplicity and effectiveness on the constraints, of the first order

Markov Chain prediction model. Of course it is possible to apply this extension in a Markov Chain model of any order.

Algorithm 6 Markov First Order Extension with correlationStep = 2

```
...
itemIndex = 0
for itemIndex < length(catalogue) do
  item = catalogue[itemIndex]
  for sequence in dataset do
    if item in sequence then
      if item != last item then
        // here the modification takes place
        if item != one before last item then
          second_next_item = find_second_next_item(sequence, item)
          indexSecond = find_index(catalogue, second_next_item)
          t_matrix[itemIndex][indexSecond] = t_matrix[itemIndex][indexSecond] + 1
        end if
          next_item = find_next_item(sequence, item)
          indexNext = find_index(catalogue, next_item)
          t_matrix[itemIndex][indexNext] = t_matrix[itemIndex][indexNext] + 1
        end if
      end if
    end for
  itemIndex = itemIndex + 1
end for
...
```

5.2.3 Prediction With Markov Chain

The prediction procedure which is implemented in C# contains a lookup of the row, representing the last item of the input sequence, and the search for the max value/s (it depends on how many prediction we want to produce) inside the row vector. The prediction is made by choosing the element of the row that has the maximum transition probability. For maximum prediction time performance a constant prediction length l can be set that will avoid the sorting of the whole row of probabilities by finding the l biggest probabilities, that their indexes are not contained in the input, and take their indexes in order to make a prediction list.

Algorithm 7 Markov Model Prediction

Require: t_matrix, catalogue, inputSeq
index = find_index(catalogue, inputSeq[-1]) // find index of last item of input in catalogue
t_matrix_row = t_matrix[index]
for item in t_matrix_row **do**
 if item in inputSeq **then**
 t_matrix_row[item_index] = 0
 end if
end for
predictions = find_best_three(t_matrix_row)
return predictions

5.3 Item Based Recommender

Item Based Recommender takes as an input the database of transactions (sequences) from users and produce a matrix where each element represents the similarity between two items. After the building process is done, the similarity matrix is saved in a text file, from the Python script. Then the similarity matrix is loaded from the application's system and can be used in order to make a prediction. For this implementation *numpy* and *scipy* libraries were used. The reason numpy was used was for storing the dataset in the representation that is required for the scipy spatial methods used for calculation of the distances.

5.3.1 Preprocessing

For the implementation we need to have a data representation in a form where the length of each data point (sequence from the dataset) should be equal to the length of all the others. For example in our virtual room builder problem we have lists of unequal length, which are representing the rooms shared by a user. So, we represent these rooms by creating binary vectors of equal length, which is the length of the items in the catalogue. The vectors are then populated with one if an item was selected in a room and zero otherwise.

You can see how we apply the preprocess in the implementation, described by pseudocode 8.

Algorithm 8 ItemBasedPreProcess

Require: dataset, catalogue
catalogueSize = length(catalogue)
dataSize = length(dataset)
trainList = Empty List
seqIndex = 0
for seqIndex < dataSize **do**
 sequence = dataset[seqIndex]
 itemIndex = 0
 binaryList = list of zeros of length catalogueSize
 for item in sequence **do**
 itemIndex = find_index(catalogue, item)
 binaryList[itemIndex] = 1
 end for
 trainList.Add(binaryList)
 seqIndex = seqIndex + 1
end for
return trainList

5.3.2 Building Item Based Recommender

During the building of this model we need to calculate the similarity between the items based on the historical transactions available. The output of the building procedure is the similarity matrix (list of lists in Python) which contains these similarity values, that the model will use in order to make a recommendation. The calculation of the similarity between two items is being done by measuring the similarity of the two column binary vectors, who represent in which transactions each of the items were chosen (1 for being chosen 0 for not being chosen) (see Table 6).

Algorithm 9 Item Based Building

```
Require: dataSet, catalogue
trainList = ItemBasedPreProcess(dataset, catalogue)
trainSize = length(trainList)
catSize = length(catalogue)
simMatrix = matrix of length (catSize, catSize/2)
catIndex = 0
for catIndex < catSize do
    chosenItems = Empty List
    for binaryList in trainList do
        if binaryList[catIndex] == 1 then
            seqIndex = 0
            for seqIndex < catSize do
                if seqIndex != catIndex then
                    if binaryList[seqIndex] == 1 then
                        chosenItems.Add(seqIndex)
                    end if
                end if
                seqIndex = seqIndex + 1
            end for
        end if
    end for
    for itemIndex in chosenItems do
        simMatrix[catIndex][itemIndex]=calcSimilarity(catIndex_column, itemIndex_column)
    end for
    catIndex = catIndex + 1
end for
return simMatrix
```

5.3.3 Prediction Choice

The building process creates an $m \times m/2$ matrix. In this matrix a row represents each item and it contains the *scores* of similarity of the specific item with every other item. Then, by aggregating these scores we can find the most similar items to the ones which the input sequence contains. An important data structure is a list of pairs which will contain the item and its aggregated score in order to make a decision for more than one predictions.

Algorithm 10 Item Based Prediction

```
Require: inputSeq, catalogue, simMatrix
scoreList = Empty Pair List
catalogueSize = length(catalogue)
index = 0
for index < catalogueSize do
  score = 0
  for item in inputSeq do
    inputIndex = find_index(item)
    score = score + simMatrix[inputIndex][index]
  end for
  scoreList.Add((index, score))
  index = index + 1
end for
predictions = find_three_best(scoreList)
return predictions
```

5.4 Compact Prediction Tree

The choice of Compact Prediction Tree was made because it was proved to have the best performance comparing with other popular methods in the comparison that was presented in their paper [7]. The comparison considered several state of the art sequence prediction algorithms like Prediction By Partial Matching, Dependency Graph and All-k-th-Order Markov Chain (Selective Markov Model). The CPT algorithm, as it was presented by the authors, was not created to have an step to store the prediction model. Thus some modifications were employed in order for the methodology to work in the two steps of building and predicting. In the first step we create all the three data structures and we store them in some files. The model then loads these data structures and uses them for the prediction.

5.4.1 Building of CPT

During the building procedure we need three data structures to be created and stored efficiently. We have the Prediction Tree which is a *trie*, the Lookup Table which can be seen as a list of trie nodes and the Inverted Index which can be implemented as a dictionary that has items of the catalogue as keys and bitvectors of size equal to the number of sequences as values. Finally, we need two custom functions in order to serialize and deserialize the PT in an efficient way.

Algorithm 11 CPT Building Procedure

```
Require: dataset, catalogue
lookUpTable = Empty List
PredictionTree = Empty Trie Node
for sequence in dataset do
  insert_node(PredictionTree, sequence)
  last_node = find_last_node(PredictionTree, sequence)
  lookUpTable.Add(last_node)
end for
inverseIndex = createInverseIndex(dataset, catalogue)
return root, lookUpTable, inverseIndex
```

The procedure for creating the inverseIndex is shown in pseudocode 12

Algorithm 12 Inverse Index creation

Require: dataset, catalogue
initialize dictionary inverseIndex with keys each item of the catalogue and values zero bitvectors of length the size of the dataset.
for catIndex < length(catalogue) **do**
 item = catalogue[catIndex]
 for sequence in dataset **do**
 if item in sequence **then**
 inverseIndex[catIndex].Add(1)
 end if
 end for
end for
return inverseIndex

5.4.2 Data Structures Storage

After the building of the model, the next step is to store the data structures in files and to be able to load them to the model. The Lookup Table can get easily stored in a text file by finding each path of each branch of the tree and writing it in a list. The bit arrays of each key from the Inverse Index dictionary can be stored in a binary file. The ‘tricky’ part is the serialization of our N-ary Prediction Tree. The procedure of the serialization that was used is described in DWARF v3 in Section 2.3 p. 13 [13] where it states: *“The tree itself is represented by flattening it in prefix order. Each entry is defined either to have child entries or not to have child entries. If an entry is defined not to have children, the next physically succeeding entry is a sibling. If an entry is defined to have children, the next physically succeeding entry is its first child. Additional children are represented as siblings of the first child. A chain of sibling entries is terminated by a null entry.”*. What this statement basically says, is that, given a tree root node, we serialize it to a list, in which each element is either a pair (node.data, has_children boolean value) or null (which represents the end of children for a node). The procedure of this serialization method is shown in pseudocode 13.

Algorithm 13 Prediction Tree Serialization

Require: PredictionTree
initialize empty list of pairs nodesList
Function recursiveSerialize(node)
 if node has children **then**
 pair = (node.node, True)
 nodesList.Add(pair)
 for child in node.children **do**
 recursiveSerialize(child)
 end for
 pair = Null
 nodesList.Add(pair)
 else
 pair = (node.node, False)
 end if
 end Function recursiveSerialize(PredictionTree)
return nodesList

Now we have a representation of the tree that can be stored in a text file. What follows is the deserialization of the tree in order to be loaded to the model. We can see the deserialization

process in pseudocode 14

Algorithm 14 Prediction Tree Deserialization

```
Require: nodesList
initialize empty node PredictionTree
PredictionTree.node = nodesList[0][0]
initialize empty stack parentStack
parentStack.Add(PredictionTree)
for pair in nodesList[1:] do
    {iterate through the nodesList except the first pair}
    if pair != Null then
        initialize empty node treeNode
        treeNode.node = pair[0]
        treeNode.parent = parentStack[-1] //last element of parentStack list
        if pair[1] == True then
            parentStack.Add(treeNode)
        end if
    else
        parentStack.Pop()
    end if
end for
return PredictionTree
```

5.4.3 Prediction

After the building is finished and the data structures are stored to their files, it is possible to apply the following methodology in order to retrieve one or more predictions. The prediction procedure contains some online (on device) computation but as the computation takes place on binary values it is very fast. We use the Inverted Index data structure in order to find similar sequences by applying logical AND to *all* the sequences binary vectors plus the input sequence. This way we get all sequences that contain the items of the input sequence in arbitrary order. The next step is to iterate through the similar sequences and find the *consequent* of the input sequence. In order to find the best possible prediction, we create a Count Table (CT) as a dictionary which stores each element that is contained in a similar sequence and it is not contained in the input sequence plus the count of it. Then, the ordering of the CT with respect to the count value is taking place and the element/s that corresponds to the maximum count are chosen for a prediction. This measure of count is called frequency support and while there are other measures that can be calculated like confidence. As it was mentioned by the authors of [7], that support is the one measure which produces the best results. Additionally the authors have introduced a variable (stepLength) for controlling the length of the number of steps that contribute for a prediction. The introduction of this variable is making the model no longer lossless but it helps in order to cover some cases where the longest steps possible will not produce any predictions. Another addition proposed by the authors of [7], is a recursive method called *recursive divider*. Recursive divider is partitioning the long sequences in to shorter ones and calculates the scores of each key from the CT through them. This choice provides a better coverage and a boost in the accuracy but obviously increases the prediction time, especially for longer sequences.

The implementation presented does not contain this recursive method as it would affect the prediction time limitations the were set. The stepLength for our implementations is set in 4.

You can see the prediction procedure in pseudocode 15

Algorithm 15 Prediction Method of CPT

Require: PredictionTree, inverseIndex, lookUpTable, inputSequence, stepLength

```

if length(inputSequence) > stepLength then
    inputSequence = the last stepLength elements of the sequence
end if
initialize empty dictionary CountTable
itemIndex = find_index(catalogue, inputSequence[0])
similar = inverseIndex[itemIndex]
//iterate through inputSequence except the first element
for item in inputSequence[1:] do
    itemIndex = find_index(catalogue, item)
    similar = similar & inverseIndex[itemIndex] //Apply logical AND
end for
simIndex = 0
for simIndex < length(similar) do
    if similar[simIndex] == True then
        tempSequence = get_sequence(lookUpTable[simIndex])
        consequent = get_consequent(tempSequence, inputSequence)
        if consequent Not Empty then
            for cosItem in consequent do
                if cosItem in CountTable keys then
                    CountTable[cosItem] = CountTable[cosItem] + 1
                else
                    CountTable[cosItem] = 1
                end if
            end for
        end if
    end if
    simIndex = simIndex + 1
end for
if length of CountTable keys > 0 then
    predictions = find_three_best(CountTable)
end if
return predictions

```

6 Datasets

The experiments that are presented in Section 7, were run in order to find to what degree any method is fulfilling the requirements while having a satisfying accuracy. However, the desired conclusion of the experiments would also give an insight on performance of each method in different possible transformations of the current company's dataset. Thus we

chose three additional real life sequential datasets to compare with the INDG dataset results. These datasets were used for benchmarks in some older competitions for Association Rule mining. There are two main different insights that these datasets can provide; first how well the methods can perform in case we have a *large number of items* like the FIFA dataset (see Table 7) and the performance when a dataset containing a *large number of sequences* like the Accidents dataset. In four datasets all sequences having length lower than 3 are removed, as a sequence of that length is possibly not a valid room configuration for our dataset.

1. **INDG** is the INDG company’s dataset which contains sequences of items (rooms configurations). It is used for the building of the final product of this research. INDG’s dataset is not publicly available, as it contains confidential information. The items repeated in the dataset are removed, as once an item is chosen it can be picked again instantly in the application, so there is no need to recommend it again.
2. **BMS1** contains click-streams in the form of integer sequences representing webpages and it is available for download at KDD-Cup-2000 archive, [19].
3. **FIFA** is also a dataset of click stream data from the website of FIFA World Cup 98 and was created by processing a part of the web logs from the world cup. It is available through SPMF Open Source Data Mining Library, [20].
4. **Accidents** contains anonymized traffic accident data available at SPMF Open Source Data Mining Library, [20].

In this table we can see the number of sequences, the number of unique items and the average length of each sequence contained in each database.

DataSets	#Sequences	Unique Items	Avg. Sequence Length
INDG	27,006	103	8.07
BMS	15,806	495	6.01
FIFA	28,979	3,301	33.94
Accidents	340,183	468	33.80

Table 7: The datasets used for the experiments. Unique Items and average sequence length is shown in order to see the differences between these datasets.

7 Experiments

For the scope of our project the results of INDG dataset are the most important and will play the main role in the final choice of the prediction model that will be used by the company. However, it is obvious that the dataset will surely change overtime, thus we need a stable method that will have a good trade off between the accuracy of the predictions and the degree to which the constraints are satisfied. The experiments are categorized in 6 different cases for all the aforementioned datasets: *Prediction Accuracy*, *Prediction Time*, *Building Time*, *Model Loading Time*, *Model RAM Consumption* and *SSD space consumption*.

7.1 10-fold Cross Validation

It is understandable that there is no training set and test set in a sequential dataset. The universal solution for evaluating the performance of a prediction model on an independent dataset is to use the N-fold cross validation technique. The validation method initially divides the given dataset into N different parts, then each prediction model is build on the N-1 parts of the dataset and gets tested on the remaining part. This process is iterated N times in such way that every different part will serve as a test set. Finally the mean accuracy over the N folds is calculated. For our experiments we chose $N = 10$.

7.2 Evaluation Method

The evaluation scheme that we used in the experiments creates a *prefix* and a *suffix* from each *test set sequence*. The *test set* is determined by the cross validation technique described in 7.1. Each test set sequence is divided in prefix and suffix by choosing a random index. The minimum prefix length is set to 2 and the maximum suffix length is set to 5. The prefix becomes the input sequence and the testing takes place on the suffix of the testing sequence. A prediction is considered as *correct* if it appears on the suffix of the test set vector. Usually, when evaluating sequence prediction algorithms the suffix length is set to 1. However in our case there is not only one right choice, after an item selection, as the correlation between items can make the user chose the same items in different orders. This evaluation method is similar to the one proposed by the authors of [7] but we do not construct a *context* part of the sequence which is discarded in their case.

The accuracy is calculated by:

$$Accuracy = \frac{\#correctPredictions}{length(testSet)} \quad (7)$$

7.2.1 First Order Markov Chain Extension and its effect

For the first experiment we evaluate the performance of the extension proposed in Section 5.2.2 with different *correlationStep* values. In order to have a better estimate of the mean accuracy we run the 10-fold validation, 10 times for each different *correlationStep* value. This procedure is useful for finding the value representing the correlation ‘window’ discussed in Section 5.2.2 that will give the best performance in a given dataset. In Figure 11 below we can see the differences in the mean accuracy with different *correlationStep* and the boost that it provides for the 4 first values. The average length of a room is 8 and there are only 1638 rooms which have length over 16. The final choice of the range of *correlationStep* is (1,13). Moreover, it seems that for values over 4 the mean accuracy is not boosting anymore. Apparently, the mean std over 100 runs of each *correlationStep* value over 4 is $std = 0.0099$, thus we could say that the accuracy over that value has a steady small range.

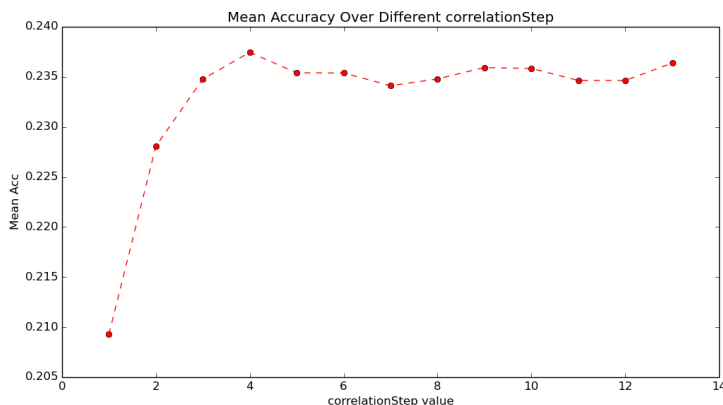


Figure 11: Here we can see the mean accuracy for different values of *correlationStep* used by Markov Chain Extension proposed in Section 5.2.2.

7.3 Outliers Removal and its effect

Identifying outliers is a wide research field and usually the methodology that is followed consists of either some assumption of the data distribution or the use of a non-contaminated training set and one class classifier. It is still questionable if, even while identified, the outliers should be removed or not. Usually the answer to this decision depends on the degree of knowledge that we have considering a specific dataset quality. The reason behind the focus on noise reduction for the company’s dataset, is that a lot of the data points contained are known to be shared as tests by the developers, and the number of those tests remains unknown. Due to this fact the dataset can include very short sequences (of length 4-5) that contain items that are not correlated.

7.3.1 Validation of Outliers

For any direction that can be followed we need a validation technique that will show in some way the effect of the outliers reduction. In order to achieve that we will follow a similar procedure to the Robust Regression technique for automatic outliers reduction. In the case of Robust Regression [8] (Section 7.4, p. 313) we have a classifier and we train the classifier with every data point contained in the dataset, then we test in the same dataset and remove a portion of the data points which produce the biggest error and check if the error is reduced overall. In our case of sequence prediction we have no error value as the prediction is either right or wrong. Thus for our dataset we use a common sense strategy (Section 7.3.2) for finding outliers and then we iteratively remove 10% portions of them and see if the average accuracy over 10 runs is improved or not. For the validation we use our best model the extended version of first order Markov Chain and *correlationStep* value set to 4.

7.3.2 Common Sense Strategy

When it comes to outliers detection it is not unusual to use common sense combined with an understanding over the data. One common feature, for sequence prediction models, that can be used is the length of each room. The reason that this feature can be used is because we have an understanding of the average length of a sequence contained in the database. We know that the average length of the rooms contained in the dataset is 8, thus we can choose different filtering techniques over the dataset depending on some predicate related to the length. One assumption based on common sense and knowledge over the dataset is that

the developers were testing the functionality of storing room configurations by creating their own *test rooms*. These test rooms should be not longer than 4-5 items and the items should be random, as they did not have the time to create longer ones containing a big variety of items. Based on that logic we can say that we could consider only sequences (rooms) of length longer than $averagelength/2$. In order to show the effect of such procedure we show the accuracy boost after the outliers reduction by 10% increments in Figure 12

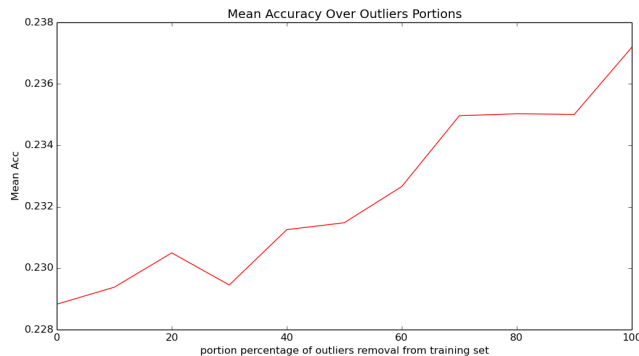


Figure 12: In this figure we can see the small gain in accuracy over the same dataset while removing portions of possibly contaminated data points from the training set.

7.3.3 Mean Accuracy over Different Methods Experiment

In Table 8 the mean accuracy over the 10 folds is shown for each dataset. In the experiments *Markov1* is the first order Markov Chain, *Markov12* is the combined model of first and second order Markov Chain and finally *MarkovM* represents the extension suggested in Section 5.2.2, of the standard Markov Chain of any of two orders. Experiments for association rules are not included as they produced results lower than 15% accuracy and also the memory consumption of the building procedure (FIM and Rule Generation) is too much to be a viable solution and easily re-buildable, especially in the case of large sequences. It has to be noted that for the FIFA and Accidents datasets the combination of first and second order Markov Chain model did not run. The reason is that the number of the combinations of the items in catalogue is too high to be computed between logical amounts of memory for the first case and also it takes too long to calculate the second order transition matrix for both datasets. Finally, as the accuracy in the experiment is the average accuracy over the 10 folds, the standard deviation is also shown in order to give a stability measure over the results of each method.

DataSets		ItemBased	Markov1	MarkovM1	Markov12	MarkovM12	CPT
INDG	Accuracy	0.2024	0.2090	0.2374	0.2182	0.2302	0.2922
	std	0.0076	0.0063	0.0061	0.0108	0.0117	0.0192
BMS	Accuracy	0.1758	0.2935	0.3262	0.3177	0.3401	0.2788
	std	0.0334	0.0254	0.0190	0.0287	0.0258	0.0283
FIFA	Accuracy	-	0.6241	0.6367	-	-	0.5410
	std	-	0.0133	0.0167	-	-	0.0128
Accidents	Accuracy	-	0.2546	0.3144	-	-	0.3250
	std	-	0.0281	0.0840	-	-	0.0275

Table 8: Mean Accuracy over 10-folds

7.4 Prediction Times Experiment

7.4.1 CPU comparison for computation times analogy

As an integration of the functionality of sequence prediction to the application system’s was not possible at the time that this thesis was written, we had to run the experiments on an Intel i7-4790k CPU. Due to this fact we need to compare this CPU with the baseline devices CPU (Apple A5 800-1000 MHz, 2 cores). We can compare these processors in some degree by using some standard benchmark tests: Integer Performance (million operations per second), Floating Point Performance (million operations per second), Finding Prime Numbers (million primes per second) and String Sorting (thousands strings per second). The tests were run using *Performance Test 8.0 Evaluation Version* (PassMark Software), by us on the two specific devices Desktop PC containing Intel i7-4790k and iPad2 containing the Apple A5 CPU. If we average the results of the 4 tests in table we can see that the i7-4790k is roughly 33 times faster than A5 in all tests. In the worst case scenario on integer calculations i7-4790k is 59 times faster than A5, this extreme case we will use in order to understand if the prediction times are within the limits of the requirements.

	i7-4790k	Apple A5
Integer Calculations (mops)	19825	333
Floating Point Calculations (mops)	8718	282
Finding Prime Numbers (mpps)	25.6	0.7
String Sorting (tsps)	9081	1271

Table 9: CPU Benchmark Comparison between i7-4790k that was used for experiments and Apple A5 that is the application’s baseline device’s CPU.

7.4.2 Prediction Times

For the prediction time calculation we run the methods for a prediction list of length 3 and the times are presented in milliseconds (*ms*). The times between the two variations of first order Markov chain will be the same, while also the times between the variations (extensions) of the combinations of first and second order Markov Chain will be the same, thus only the times for the standard Markov models will be presented. For the time calculation we used Stopwatch from System.Diagnostics in C# which returns the time calculation in milliseconds. Each of the model prediction times in the table are the prediction times tested on the i7-4790k CPU and their times are multiplied by the 59 which proved to be the biggest difference possible between the two processors (i7 and A5) in Section 7.4.1.

DataSets	ItemBased (<i>ms</i>)	Markov1 (<i>ms</i>)	Markov12 (<i>ms</i>)	CPT (<i>ms</i>)
INDG	118	177	177	413
BMS	118	177	177	531
FIFA	-	263	-	354
Accidents	-	177	-	18585

Table 10: Mean prediction time over 10-folds

7.5 Model Loading Times Experiment

All the models that were used for experimentation were saving in some a prediction model to the hard drive of an external system (other than the application’s system). This prediction model needs to be loaded to the application’s system, in order to start using the predictive functionality in the application. In Table 11 the loading times are shown in milliseconds where these ms are added to the loading time of the application. The loading times were calculated also with the help of Stopwatch and the final milliseconds are the aggregated milliseconds of each loading part of the model. As it was not possible to test the loading times in the baseline devices, the experiments run on Western Digital WD7500BPVX Hard Disk. The iPad 2 device and the WD hard disk were tested on their performance considering read speed and proved to have similar performance, 72MB/s for WD hard disk and 75 MB/s for iPad2 SSD.

Note: o.a.m. = over a minute

DataSets	ItemBased	Markov1	Markov12	CPT
INDG	33 ms	32 ms	975 ms	462 ms
BMS	425 ms	237 ms	o.a.m.	674 ms
FIFA	-	8850 ms	-	6344 ms
ACCIDENTS	-	228 ms	-	18601 ms

Table 11: Loading Time

7.6 Model RAM Consumption Experiment

This experiment is done for the scope of finding the most suitable method considering the memory requirements set by the company. The same logic considering the times of Markov Chain different models applies to this experiment too, as the modifications have the same RAM consumption with the standard models. In order to get the ram consumption value we loaded each model and run the C# command *GC.GetTotalMemory*.

DataSets	ItemBased	Markov1	Markov12	CPT
INDG	4.23 MB	4.23 MB	22.82 MB	31.14 MB
BMS	19.56 MB	10.47 MB	1142 MB	19.31 MB
FIFA	-	134.94 MB	-	19.43 MB
ACCIDENTS	-	11,67 MB	-	1120 MB

Table 12: RAM consumption for each method

7.7 Model SSD Size Consumption Experiment

The model that is loaded to the application system has to be stored in the internal storage SSD space of a device running the application. Thus, the size that is added from the model

files will be added to the application size. Although we do not have a restriction, it is preferable to use the smaller model in size.

DataSets	ItemBased	Markov1	Markov12	CPT
INDG	228 KB	220 KB	5.2 MB	5.1 MB
BMS	3.1 MB	990 KB	369 MB	8.9 MB
FIFA	-	34 MB	-	110.7 MB
ACCIDENTS	-	951 KB	-	262 MB

Table 13: Functionality Size

7.8 Building Times Experiment

We have 4 different approaches to consider as we want to see the deviation between their building times. The building times experiment run in Python as we only wanted to get a general performance insight. The whole part of each dataset was used for the building.

DataSets	ItemBased	Markov1	Markov12	CPT
INDG	59.17 min	0.11 min	22.75 min	0.21 min
BMS	8.18 min	0.15 min	60.22 min	0.09 min
FIFA	-	2.44 min	-	1.31 min
Accidents	-	4.08 min	-	2.67 min

Table 14: Building times over the whole datasets

8 Results

8.1 Markov Chain First Order (MarkovM)

The Markov Chain of First Order was considered as a possible solution as it is a very simple model that can be easily stored and loaded into a matrix. It is easy to implement it in any language as it does not contain any complex data structures, and in general it has been proven to produce good results in sequence prediction problems. The constraints are successfully fulfilled from this model as it generates predictions fast and has low memory consumption within the minimum limits and also fast model loading. It is also easy to rebuild the model with building times of less than 4 minutes even in datasets containing hundreds of thousands of sequences and thousands of items. We could say that this method is the second best on solving our specific problem and therefore fulfilling the constraints.

8.2 Markov Chain Extension First Order(MarkovM1)

While the trade off between accuracy in most cases is not big, it is apparent from the experiments that the extended version of first Order Markov Chain is the best choice considering the constraints, accuracy and building times. It remains as simple as the standard Markov Chain of first order in all aspects while it gets an accuracy boost (especially for the company’s dataset) with the introduction of the extension in Section 5.2.2. The extension

suggested proved to provide a little higher accuracy in all cases but the performance depends on how correlated are the elements between a range (which range needs to be determined by experimentation). The only problem that we could encounter considering the winning model and the requirements is that in case of too many individual items of a catalogue like FIFA dataset the model both loading time and memory consumption exceed the limits that the company set. However this is an extreme case where we have over than 3 thousands items.

8.3 Item Based

Item Based recommender approach was chosen for the scope of this paper, as an alternative of the standard sequence prediction approaches and it proved to be satisfactory in most cases. The Item Based issue is the building time, as it takes long enough to calculate the distances when a large number of sequences or unique items are contained in the dataset set. However Item Based is very close to the winner in all aspects of the constraints except loading time. It is possible to tackle the building time problem to a degree with some parallelization of the process. However, distance calculation times will still be high enough, while there are methods that can build a model faster with similar or better constraints fulfillments and better accuracy.

8.4 Markov Combined (Markov12)

The combined model of the first and second order Markov Chain proved to be the worse in most cases considering all the constraints while it remains on a steady good accuracy in different datasets. The problems that emerge with the combined model, are both long building times and memory consumption, required by the second Order transition matrix. The standard combined model proves to have a worse accuracy than the extended version of the first Order Markov Chain with *correlationStep* set to 4. The building time can be easily get reduced with simple parallelization but we cannot overcome the bad performance on constraints.

8.5 Markov Combined Extension (MarkovM12)

The combined first and second Order Markov chain when run with the extension, while it shares the same problems with the standard combined method, proved to have slightly better accuracy even from the winner model as it combines the both the extension proposed and the first and second order Markov Chain models.

8.6 CPT

CPT is a novel technique and from the aspect of accuracy it has satisfying results. Nevertheless it can have a low coverage as it is also mentioned in CPT paper [7]. In order to overcome this issue a method named recursive divider is introduced to by the authors which can slightly increase the prediction times without eliminating in total the coverage issue. The recursive divider also slightly boosts the accuracy, this is why our CPT performs worse in the BMS dataset than the implementation of the authors. We tested CPT without the recursive divider method in order to gain in prediction time. The resulted to a faster implementation but not within the limits of the requirements. The main issue with CPT is that,

as it was not designed to be efficient for our constraints, the data structures that need to be stored are proved to have the worst performance considering the constraints of memory consumption (especially in case of high number of items in catalogue), SSD storage size and loading time.

9 Future Work/Recommendations

The extended Markov Chain of First Order with *correlationStep* value 4 proved to be the model fulfilling the constraints to the higher degree. However, when the number of items in the catalogue is high, the limits of memory consumption and loading time are exceeded. More research is needed for finding better ways to store the matrix in both memory and space. For transition matrices of orders higher than 1 a sparse matrix representation could be used in order to reduce the size of the matrix in both RAM and SSD.

The Compact Prediction Tree methodology was interesting to apply to the problem. The data structures were not optimized to our problem which led to high loading times and model size. Further research is needed for applying the whole sequence similarity computation offline and store the similar sequences for each similar sequence in a matrix like we did for items in item-based.

Though cross validation can give us an indication of the performance of the model in an independent dataset, analyzing feedback from users can prove more effective. Additionally, it should be investigated which is the optimum period to rebuild the model considering the changes that can happen in the dataset (additions of items, many new sequences to consider for building). This is an important decision because, if a model is trained just after a new item is introduced a very small percentage, if any, of the sequences would contain the new item. On the other hand if we do not take into account a new item soon it will not get promoted as much as the others as it can not get predicted yet. If the method proves to be used by the users a lot (high percentage of users that clicked on the prediction), it could make sense to create a framework which would automatically rebuilds the model when it is supposed to be rebuild and reapply it to the application's system.

10 Conclusion

The task of this thesis was to deliver a sequence prediction algorithm that works with an offline mobile application named Amikasa developed by INDG company. The smartphone application system limits the way we treat such problem considering prediction time, RAM consumption and model loading time as important constraints that need to be fulfilled. In order to find the best method among others we followed some variety different directions of solving the problem and compared them on four different dataset (including the company's dataset). The comparison was based on accuracy and the performance of each algorithm on the system requirements. For the evaluation of the accuracy we used 10-Fold Cross Validation technique written in Python 2.7 and the constraints performance tests were written in C# as it is the language that is used from the application's system. An extension of the standard first order Markov Chain was introduced that proved to have a higher accuracy while satisfying the requirements to the fullest. Additionally, we briefly presented a way to remove noise from a contaminated dataset based on common sense and understanding over the data and proved the effectiveness of such method on the company's dataset.

References

- [1] Rakesh Agrawal, Tomasz Imielinski, Arun N. Swami, *Mining Association Rules between Sets of Items in Large Databases*, Proceedings of the 1993, International Conference on Management of Data, Washington, D.C., 1993.
- [2] Mohammed J. Zaki, Ching-Jui Hsiao, *CHARM: An Efficient Algorithm for Closed Association Rule Mining*, Computer Science Department, Rensselaer Polytechnic Institute, 2004.
- [3] Jiawei Han, Micheline Kamber, Jian Pei, *Data Mining Concepts and Techniques 3rd Edition*, Morgan Kaufmann, University of Illinois at UrbanaChampaign, Simon Fraser University, 2012.
- [4] Liqiang Geng, Howard J. Hamilton, *Interestingness Measures for Data Mining: A Survey*, University of Regina, ACM Computing Surveys, Vol. 38, No. 3, Article 9, 2006.
- [5] Ramakrishnan Srikant, Rakesh Agrawal, *Mining Generalized Association Rules*, IBM Almaden Research Center, San Jose, CA 95120, Proceedings of the 2first VLDB Conference Zurich, Swizerland, 1995.
- [6] Greg Linden, Brent Smith, Jeremy York, *Amazon.com Recommendations Item-to-Item Collaborative Filtering*, Amazon.com, IEEE Computer Society, 2003.
- [7] Ted Gueniche, Philippe Fournier-Viger, Vincent S. Tseng, *Compact Prediction Tree: A Lossless Model for Accurate Sequence Prediction*, Dept. of Computer Science, University of Moncton, Canada and Dept. of Computer Science and Inf. Eng., National Cheng Kung University, Taiwan, 2013.
- [8] Ian H. Witten, Eibe Frank, *Data Mining: Practical Machine Learning Tools and Techniques Second Edition*, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2005.
- [9] Jitender Deogun, Liying Jiang, *Prediction Mining An Approach to Mining Association Rules for Prediction*, Department of Computer Science and Engineering, University of Nebraska, Lincoln, 2005.
- [10] Cynthia Rudin, Benjamin Letham, Ansaf Salleb-Aouissi, Eugene Kogan, David Madigan, *Sequential Event Prediction with Association Rules*, JMLR: Workshop and Conference Proceedings 19, p 615634, 24th Annual Conference on Learning Theory, 2011.
- [11] Iuliana Teodorescu, *Maximum Likelihood Estimation For Markov Chains*, Department of Statistics, University of New Mexico, Albuquerque, 2009.
- [12] Mukund Deshpande, George Karypis, *Selective Markov Models for Predicting Web-Page Accesses*, University of Minnesota, Department of Computer Science/Army HPC Research Center Minneapolis, 2004.
- [13] DWARF Debugging Information Format Workgroup *DWARF Debugging Information Format Version 3*, Free Standards Group, 2005

- [14] Ron Begleiter, Ran El-Yaniv, Golan Yona, *On Prediction Using Variable Order Markov Models*, Journal of Artificial Intelligence Research 22, p 385-421 ,2004
- [15] Jun Chen, Chaokun Wang, Jianmin Wang, *A Personalized Interest-Forgetting Markov Model for Recommendations*, Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015
- [16] Tie Liu, *Application of Markov Chains to Analyze and Predict the Time Series*, Specialized Scientific Research Program on Scientific Research of high-level talents in Ankang University (Program No. AYQDZR200705), Vol. 4, No. 5, 2010
- [17] Badrul Sarwar, George Karypis, Joseph Konstan, John Riedl, *Item-Based Collaborative Filtering Recommendation Algorithms*, WWW10, Hong Kong, ACM 1-58113-348-0/01/0005, 2001
- [18] Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao, *Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach*, Data Mining and Knowledge Discovery, p 53-87, 2004
- [19] KDD Cup 2000: Online retailer website clickstream analysis, retrieved by <http://www.kdd.org/kdd-cup/view/kdd-cup-2000>, ,08/06/2016
- [20] An Open-Source Data Mining Library, retrieved by <http://www.philippe-fournier-viger.com/spmf/>, ,08/06/2016