Recommending Games Based on Implicit Feedback

Master thesis

Wednesday, 02 May 2012
Business Information Systems
Eindhoven University of Technology
Faculty of Mathematics & Computer Science
Abstract

Nowadays recommender systems are often used in e-commerce. They are vital to increase customer loyalty and sales by offering the products the customer is most likely interested in. In this report we apply the notion of a recommender system to the business model of a games website. We implement a recommender system based on historical data of previously played games without the visitor expressing their explicit feedback; instead we use the database log for implicit feedback.
Contents

1. Introduction................................................................................................................. 1
   1.1. Context .................................................................................................................. 1
   1.2. Problem Description .............................................................................................. 2
   1.3. Project Goals ......................................................................................................... 2
   1.4. Approach ............................................................................................................... 2
   1.5. Overview ............................................................................................................... 3
2. Related Work .............................................................................................................. 4
3. Data .......................................................................................................................... 6
   3.1. Available Data ....................................................................................................... 6
   3.2. Data Quality ......................................................................................................... 7
4. Potential of newly added games .............................................................................. 12
   4.1. Current popularity measure .................................................................................. 12
   4.2. Effect of popularity measure ............................................................................... 14
   4.3. Model .................................................................................................................... 15
      4.3.1. Formal approach ......................................................................................... 17
   4.4. Applications ......................................................................................................... 17
   4.5. Evaluation of the estimation ............................................................................... 19
   4.6. Relation between new list and categorical popularity lists .................................. 21
   4.7. Future Work ......................................................................................................... 25
   4.8. Conclusion on the popularity estimation ............................................................... 26
5. Broken game classification ...................................................................................... 27
   5.1. Attributes ............................................................................................................. 27
   5.2. Comparison of methods ....................................................................................... 28
   5.3. Classification Model ............................................................................................ 30
   5.4. Description of the model ...................................................................................... 30
   5.5. Evaluation ............................................................................................................ 31
   5.6. Cost Sensitive classification .................................................................................. 32
   5.7. Implementation ..................................................................................................... 32
   5.8. Future work .......................................................................................................... 33
   5.9. Conclusion on the classification .......................................................................... 33
6. Recommender System .............................................................................................. 35
   6.1. Hybrid ................................................................................................................. 35
   6.2. Notion of preference ......................................................................................... 35
6.3. General Popularity Model: Exploration-Exploitation ................................................. 36
6.4. Personalized model: Collaborative filtering ................................................................. 38
   6.4.1. Optimizations ........................................................................................................ 40
   6.4.2. Dataset .................................................................................................................. 41
6.5. Integration Details .......................................................................................................... 43
6.6. Evaluation of the personalized model ............................................................................ 45
   6.6.1. Top-N evaluation .................................................................................................... 47
   6.6.2. Empirical Evaluation .............................................................................................. 49
7. Conclusion and Future Work .............................................................................................. 50
Bibliography .......................................................................................................................... 52
Appendix A. Popularity Lists on Index .................................................................................. 54
Appendix B. Target groups ...................................................................................................... 55
Appendix C. Preferred model threshold .................................................................................. 56
Appendix D. Decision Tree model .......................................................................................... 57
Appendix E. Cost Sensitive Model ......................................................................................... 58
Appendix F. Used Queries ...................................................................................................... 59
Appendix G. Average playtimes in heartbeat intervals ............................................................ 60
Appendix H. Top-N Evaluation Details ................................................................................... 61
1. Introduction
Recommender systems are often used in e-commerce to offer suitable products to visitors of the website. By offering suitable products the goal is to optimize sales and reduce the visitor’s effort in finding what they are looking for. Without the recommender system, people might have not have seen the product and therefore could also not have bought it. In this paper we take the notion of product consumption to the world of online games. Although it is not possible for visitors to buy the online games, we might consider playing a game as some sort of consumption and apply existing recommender system techniques to optimize the visitor experience.

1.1. Context
Tibaco Internet Media B.V. is a company that offers online games on their websites. Examples of their collection of websites are Kindertent, Playtime, Jogo and the largest branch Funnygames. In this report we will focus on the Dutch domain of Funnygames, since this is the largest of the Funnygames domains with around 2 million unique visitors each month.

Most of the games they offer are not developed by Tibaco itself, but are acquired from other parties. All games are manually assigned to a category. Categories are defined recursively such that a parent category consists of several subcategories. The most popular games of a top level category will appear on the index. There are 8 top level categories and four special index lists (See Appendix A), of which we will discuss the exact composition in Chapter 4. Games that are not included in any of these lists will only be accessible by a search function or by cycling through the collection of unpopular games on the archive pages. The image below depicts the appearance of the website.

Figure 1. Appearance of the website
The website is hosted in a cloud [1]. The main implication of cloud computing for the recommender system is the cost of extra computation. In our approach we have to take this in consideration and either find an efficient method or compute the recommendations in an offline fashion.

Tibaco receives income from displaying advertisements on their website. The advertisements are either provided by the Google AdSense program or by companies through the sales department of Tibaco. The revenue is either generated by a page view that includes the advertisement or by a click on the advertisement. It is therefore important to offer many advertisements to visitors, which is directly related to the amount of games a visitor plays.

### 1.2. Problem Description

One way to increase the amount of advertisements offered, is to increasing gameplays. We attempt to increase these gameplays by offering suitable games. The website offers many games, of which only a few can be actively promoted by listing them in one of the twelve index lists. In the current format of the Funnygames websites, there are 192 slots available on the index page and 4 slots to offer games to visitors on each game page. We will elaborate on the exact content of the slots in Chapter 4, but it is important to know that these slots are the same for each visitor and not personalized.

A visitor however, might only be interested in a certain type of games, which means that in the current situation many games are offered to the visitor that are not of interest and also many games might not be shown while they are in fact of interest to the visitor. Note that we cannot offer all games, because the amount of games exceeds 15.000. In order to fully utilize the available slots, it is vital to offer those games that are suited for the individual visitor rather than offering a generic list of games of which only a few will be of interest to the specific visitor.

### 1.3. Project Goals

On our approach to implement the recommender system, we introduce some other interesting topics. These topics, a classification for detecting broken games and a popularity estimation technique for new games, are not directly related to the recommender system, but offer interesting applications on the data. We define the following research questions:

- **Q1**: Based on partial daily data, can we estimate the popularity of a game?
- **Q2**: Can we use the estimation of Q1 to improve the constitution of the index lists?
- **Q3**: Can we use game statistics to detect broken games?

Once we have answered the above questions we focus on the main research questions of this thesis, related to the actual recommender system.

- **Q4**: How can we recommend games to visitors that have no gameplay history?
- **Q5**: How can we recommend games to returning visitors, based on their gameplay history?

### 1.4. Approach

We start off by studying scientific articles in order to find suitable methods for our context. As a starting point we look at several approaches on explicit feedback datasets and see how these relate
to the implicit feedback datasets. As described before, the website of Tibaco runs in the cloud which leads to extra restrictions regarding resource usage, as computations regarding recommender systems are often time consuming.

During our search in literature we match related work to our context. A graphical overview of our approach is given in Figure 2. We answer the research questions in three separate sections, which are not tightly related. Since late 2010 all webpage requests by visitors have been recorded in a database. We use this data as a basis for the three sections.

![Figure 2. Overview of the approach](image)

For the popularity estimation, we look at the constitution of the index page to analyze the effect of the current popularity measure and relate the current user behavior to this popularity measure. During the broken game classification we compare several approaches and use the best performing approach to classify the games. Regarding the recommender system we first look at several options for defining preference, because we do not have explicit ratings. Then we combine the knowledge from literature and our data to design a recommender system for our context and perform several evaluation techniques on the models.

1.5. Overview

In the upcoming sections we start by positioning our context and approach in the current scientific literature in chapter 2. Then we take a look at the data that we have available in chapter 3. During the analysis of the data we encounter several interesting aspects of the data that we will elaborate on as a subsection of chapter 3.

Once we have identified the environment and data properties we will discuss the three main subjects. First we describe a method for estimating the popularity potential of a game in chapter 4. Chapter 5 describes a classification for detecting broken game on the website. As the final topic we develop a recommender system that will suggest interesting games to the visitors in chapter 6.

We end this report with a conclusion where we put the results of our approach in perspective to the project goals and discuss possible improvements.
2. Related Work

In this section we will describe several papers and approaches that are related to the approach we take. We start off with a user based collaborative filtering approach and then switch to the closely related item based collaborative filtering. These approaches share the same model consisting of users, items and ratings of items by these users. Finally we include matrix decomposition techniques by first discussing the default approach and shift towards approaches in literature regarding implicit feedback.

The first technique we discuss is the user to user nearest neighbor algorithm [2]. The idea is that if users rated items similar in the past, they will share opinions on other items as well and we can derive a rating of this user for the item based on the similar user’s rating. User similarity is central in this approach, where we compare ratings of different users of the same items. There exist different measures to calculate similarity between users. Two common measures are the cosine similarity [3] \( s_C \) and Pearson correlation [4] \( s_P \) as indicated below for two points \( x \) and \( y \) in an n-dimensional space, where \( n \) is the amount of items that are rated by both user \( x \) and \( y \).

\[
s_C(x, y) = \frac{x \cdot y}{\|x\|\|y\|} \quad s_P(x, y) = \frac{\sum_{i=1}^{n}(R_{x,i} - \bar{R}_x)(R_{y,i} - \bar{R}_y)}{\sqrt{\sum_{i=1}^{n}(R_{x,i} - \bar{R}_x)^2} \sqrt{\sum_{i=1}^{n}(R_{y,i} - \bar{R}_y)^2}}
\]

Breese et al. [2] showed that the Pearson correlation often performs better than the cosine measure. Using the Pearson correlation, the estimation of the rating for an item \( i \) by user \( u \) can be computed by the weighted average of the ratings of its k-nearest neighbors [5].

\[
\hat{R}(u, i) = \frac{\sum_{a=1}^{k} s_P(u, a)(R_{a,i} - \bar{R}_a)}{\sum_{a=1}^{k}(s_P(u, a))}
\]

Although the above approach of defining user similarity is intuitive, Sarwar et al. [3] showed that item based collaborative filtering outperforms user based collaborative filtering. Item based collaborative filtering is similar to user based collaborative filtering, with a modification that we search for item similarity instead of user similarity. The underlying logic is that a user will most likely also like items similar to the ones of which this user has indicated that he or she likes.

An alternative to the neighborhood approach are matrix factorization techniques. In matrix factorization all users are associated with a user-factor vector \( x_u \in \mathbb{R}^f \) and each item with an item-factor vector \( y_i \in \mathbb{R}^f \), where typically the amount of factors \( f \) is much smaller than the amount of users and items. This allows for a simplified representation for both the users and the items in terms of these factors. The vectors \( x_u \) and \( y_i \) are not explicitly given, but are obtained by decomposing the user-item rating matrix as a product of two \( f \) dimensional matrices. This is a numerical optimization problem where we try to find the two matrices of lower dimensions such that their product is close to the original rating matrix. Examples of techniques for this optimization are stochastic gradient descent [6] and the method of alternating least squares [7].
The inner product of the vectors of the lower dimensional matrices results in the estimated ratings \( \hat{r}_{ui} \) for the user-item combination.

\[
\hat{r}_{ui} = x^T_u y_i
\]

An approach for explicit feedback datasets is given in *Matrix Factorization Techniques for Recommender Systems* [6] and an approach for implicit feedback datasets in *Collaborative Filtering for Implicit Feedback Datasets* [8]. Koren [9] showed that a combination of both a neighborhood model and a factorization model can improve performance.
3. Data
Since late 2010 Tibaco has been collecting data about their visitors. Examples of this data are navigational paths and data about the user obtained through questionnaires. This is the information we use for our broken game classification, the popularity estimation and the recommender system. In this section we give an overview of the available data and take a look at several interesting properties of the data.

3.1. Available Data
We will give an overview of the most important data that is used throughout this document and supplement the notion of available data in the corresponding sections where needed. The data that is central throughout this document is the set of user transactions. Each webpage request by a visitor is logged in the database. This log consists of the following data.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uuid</td>
<td>A unique identifier for the visitor.</td>
</tr>
<tr>
<td>counter</td>
<td>The page identifier of the requested page.</td>
</tr>
<tr>
<td>count</td>
<td>A value indicating how often the visitor visited the page in the corresponding session.</td>
</tr>
<tr>
<td>timestamp</td>
<td>Time of the request in seconds after 00:00 on 1-1-1970.</td>
</tr>
<tr>
<td>seconds</td>
<td>(An estimation of) the time the visitor remained on the requested page. See section 3.2 for more information and implications.</td>
</tr>
</tbody>
</table>

The uuid is a universally unique identifier generated according to the specifications of RFC4122 [10] and stored in a cookie for visitors that have not supplied a uuid from their current cookie. Users that have no cookies enabled or remove their cookies will regenerate a new uuid and appear in the database as a distinct visitor each session. For readability we will reference to all visitors without a uuid in their cookie as new visitors. In addition to these individual transactions, a number is stored under uuid global that is said to be a materialized view of the daily total; the sum over all visitors. However, in section 3.2 we will see that there is a discrepancy.

There is also a database that contains age, gender and country information in relation to a session and tables with a sessionid in relation to a uuid.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_id</td>
<td>A unique identifier for the session.</td>
</tr>
<tr>
<td>type</td>
<td>{start, end} indicating whether it is the start or the end of a session</td>
</tr>
<tr>
<td>name</td>
<td>{CountryCode, age, gender}.</td>
</tr>
<tr>
<td>value</td>
<td>The value corresponding to the name attribute.</td>
</tr>
</tbody>
</table>

We can link this information to a uuid by joining the two tables on their session_id and therefore we can also link a page request to an age and/or gender using the first table. Note that the age and
gender information is not available for all visitors. This information is obtained by a questionnaire that occurs below a game and is optional to fill out.

Each game has a certain amount of properties. The most important ones are the identifier, the name and the category the game belongs to.

<table>
<thead>
<tr>
<th>id</th>
<th>Game identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Game name</td>
</tr>
<tr>
<td>categoryid</td>
<td>Category identifier</td>
</tr>
</tbody>
</table>

Table 4. Game information

Finally we have a table that defines the hierarchical structure of the categories, by mapping each category to its parent. The highest level categories have category 0 as parent, which is non-existent.

<table>
<thead>
<tr>
<th>id</th>
<th>Category identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>The parent category identifier.</td>
</tr>
<tr>
<td>category</td>
<td>Name</td>
</tr>
</tbody>
</table>

Table 5. Hierarchical structure of categories

3.2. Data Quality

This section describes several aspects of the data that have been discovered during the process. Because these aspects do not directly relate to any of the upcoming sections or will distract the reader from the subject we will discuss it in this section first and reference it in the upcoming sections. The aspects of the data that we will discuss are 1) the difference between the intended value and the true value related to the userid \textit{global}, 2) unlikely playtime durations and 3) irregularities in the playtime distribution.

\textbf{UUID global}

In the previous section we introduced the userid \textit{global}, which was designed to represent the materialized view of the sum over all users. We mentioned that there was a discrepancy between the intended value and the true value. This section examines this discrepancy. If we accumulate the individual database transactions per game page counter, and plot this magnitude in relation to the stored amount of requests for uuid 'global' in Figure 3, we notice that these values are not equal. All points should lie on the red diagonal line.

![Figure 3: Materialized view in relation to accumulated total.](image-url)
Apart from several outliers, we notice a linear relation between the materialized view and the accumulated total. Using simple linear regression we find a factor that expresses the difference in relation to the accumulated total, which is expressed in $sumCount$, the sum over all individual page requests.

$$factor = 0.8416 - 4.07E^{-6} sumCount$$

On a domain of $sumCount$ ranging from 1 to 2000, we see that the factor is almost constant over $sumCount$ and conclude that either 15-16% of the individual records are missing or that the materialized view considers an incorrect amount of transactions of the same proportion, either by duplication or unrelated data. We conclude that the materialized view might once have represented the accumulated total, but at the current point in time it represents an incorrect value. The materialized view should therefore not be used for statistical analysis.

**Unlikely long playtimes**

According to the data in the database, several users have been playing a game for over 24 hours on a single day. This is caused by the technique to write playtimes to the database after the visitor stops playing the game. Browsers that have been displaying a page for over a day and are then closed, result in an unrealistically long playtime. The visitor has most likely not been playing the game for that long. There is currently no registration for user activity inside the game window. This would cause too much interaction with the session server, causing an excessive server load.

Although we cannot make a practical distinction between valid and invalid playtimes, we make a statistical distinction. We make a box plot to determine the maximum value and remove the outliers. We make this distinction on the daily playtime aggregated by user by game.

![Figure 4. Box plot of playtimes](image-url)
We consider a value to be an outlier if it exceeds 1.5 IQR (inter quartile range) above the third quartile, a method often described in literature [11,12]. Note that in our case it does not make sense to detect outliers below 1.5 IQR below the first quartile. From this box plot we determine Q3 + 1.5 IQR to equal 1330 + 1.5 (1330-26) = 3286. All values above are considered outliers and removed from the data. The same phenomenon possibly also occurs for shorter playtimes (e.g. coffee or toilet break), but it is harder perhaps even impossible to distinguish these small effects from the true playtimes.

**Irregularities in playtime distribution**

Playtimes are based on timestamps between two pages. The final page of a session does not have a consecutive page to determine the playtime of the final page. To determine the playtime of the final page, the website includes code that is executed on the visitor's computer which pings a session server every five minutes while the user stays on the page. The timestamp of the end of the playtime is set to the time of the final heartbeat. Note that this leads to a playtime of zero seconds for actual playtimes up to five minutes or more generally, it leads to a recorded playtime rounded down to the closest multiple of the heartbeat interval. This method is inaccurate, but absolutely necessary to accommodate for the playtimes of the final game, since experiments by the development department of Tibaco to introduce a kill signal upon navigating away from the website were unsuccessful. The average playtime of the final game is about fifteen minutes. An inaccuracy of five minutes is quite significant.

The effect of this inaccuracy is expressed in the graph below. In this graph the blue line shows the cumulative distribution of recorded playtimes and the red line indicates the expected distribution. Every five minutes (including 0 minutes) the recorded playtime shows a strong increase. These playtimes are determined by the heartbeat interval method.

![Figure 5. Irregularities in playtime distribution](image)

If we are able to increase the accuracy of the recorded playtimes, we expect to achieve better models. An important factor in the accuracy of our data is the large interval of the heartbeat. Lowering this interval increases the amount of messages sent to the server and thus increases the server load.
In practical experiments by the development department of Tibaco an interval of one minute leads to server overload. An interval of two minutes did not lead to any problems and to ensure extra safety an interval of three minutes is currently used instead of the initial five minute interval. Note that we will sketch an alternative approach at the end of this subsection.

In Figure 6 we see the effect of the heartbeat interval reduction on the distribution of playtimes. The green line indicates the interval on March 25th in 2011 when the interval was set at five minutes and the purple line indicates an interval of three minutes as observed on October 26th. Because it concerns different days the frequencies cannot be compared, but it can clearly be seen that the burst in the playtime distribution has shifted from multiples of five minutes, to multiples of three minutes. This means the interval has been reduced in practice and we have limited the possible range of the true value of the playtime to an interval of three minutes and therefore also reduced the inaccuracy in our recorded playtime.

As a practical note, we mention that when changing the heartbeat interval, a combination of new and old heartbeat intervals was seen in the database during the day after the change. This may be explained by the client caching the script that sends out the heartbeat and results in a day of recorded data consisting of both intervals that cannot be explicitly discriminated from genuine playtimes of the same length.

We could also use more advanced techniques to determine a lower limit for the heartbeat interval, by setting the heartbeat interval to different values and for each of these values we record the highest server load over several days. This results in samples of maximum server load for each of the heartbeat settings. Using these samples we can derive a distribution and use statistical analysis to determine confidence intervals with respect to the distribution of server overload probability.

Playtimes can be determined by the two different techniques we have discussed earlier, based on time between pageloads or by the heartbeat method. We may want to treat, modify or interpret values resulting from these methods in a different manner. In particular, in section 6.4.2 we want to
modify only the values of the heartbeat method. It is currently not explicitly stored which technique determined the playtime. It is however possible to deduct navigation paths from the database and determine the last page that was requested by the visitor. For efficiency reasons it would be most practical to store the type of the method upon determination of the playtime.

In this section we have discussed three possible problems in the data quality with respect to the applications we will discuss in the upcoming sections. At the same time we mention that the effect of these issues is expected to be minimal because it only affects a small portion of the data. Note that we can simply recompute the sum to avoid the incorrect value of the materialized view, less than 1% of the playtimes exceed the outlier boundary and less than 8% of the playtimes are determined by the heartbeat method.
4. Potential of newly added games

Tibaco is interested in adding games at a higher daily rate than in the current situation. Three games are added each day, which is currently being increased to five. Since the amount of slots on the index page remains the same, this means that a game will cycle through the new list in a shorter time and it has less time to gain popularity. In this section we analyze the effect of the popularity measure that is currently used and search for a method to estimate popularity potential of games while they are still in the new list. We are particularly interested in examining if we can use this estimation to early remove games from this new list in case we either see that the game has no potential of being popular, or move them to the categorical popularity list in case the game is estimated to reach a popularity threshold. In this section we answer the first two research questions we defined in the introduction.

Q1: Based on partial daily data, can we estimate the popularity of a game?

Q2: Can we use the estimation of Q1 to improve the constitution of the index lists?

4.1. Current popularity measure

Before we dive into the actual construction of the popularity lists, we first need to define popularity. In the Tibaco case, popularity is measured in gameplays, weighted over the days as an exponentially weighted moving average (EWMA or EMA). An EMA \( z_t \) is defined in Applied Statistics and Probability for Engineers [11] as follows.

\[
\text{Equation 1: Exponentially Weighted Moving Average}
\]

\[
z_t = \sum_{k=0}^{t} \lambda(1-\lambda)^k x_{t-k} + (1-\lambda)^t \mu_0,
\]

which can be written recursively as

\[
\text{Equation 2: Exponentially Weighted Moving Average (recursive)}
\]

\[
z_t = \lambda x_t + (1-\lambda)z_{t-1}
\]

\[
z_0 = x_0
\]

In our case, \( x_t \) is the amount of gameplays of a game on day \( t \), which is weighted exponentially by a parameter \( \lambda \). For historical reasons, this weighting parameter \( \lambda \) is set to \( \frac{2}{29} \) by Tibaco, resulting in the following weight distribution.
Tibaco does not only compute these EMA scores as an overall statistic, but has defined several target groups and for each of these groups an EMA score is calculated. To determine to which target group a visitor belongs, we combine the transactional information with the age and gender information in the database using the uuid and sessionid. Based upon this information we can calculate the gameplays that belong to a certain target group. Using the same definition as above, we may calculate EMA scores for each target group. Both the global and the targeted EMAs are inputted as XML files into a module to generate the site, including the index page.

We will now discuss the construction of the categorical popularity lists. Each category has its own popularity list. For each of the target groups we order games in the current category on their targeted EMA score in decreasing order, so each game occurs once in each of the target group’s list. Then for each of the targeted popularity lists we take the most popular game to be added in the list. Tibaco distinguishes between nine different target groups, but only eight are included in the index calculation (age 0-5 is not included). Since we now have eight different age groups and two different genders, this results in sixteen popular games, corresponding to the sixteen slots per category on the index.

The special popularity list popular follows the same principle as the categorical popularity list, but has no categorical restrictions. To prevent duplication with the categorical popularity lists, any game that is in both the popular list and a categorical list will only occur in the popular list. The games in the categorical popularity list are supplemented by the next categorically most popular game.
The other special index page list is the list of new games. Tibaco launches a new game at three different times of the day, i.e. at 9AM, 12 noon and 3PM. A new game is added at the top of the new list; the oldest game in the list is removed.

4.2. Effect of popularity measure

The purpose of the new list is to give a chance to new games to accumulate enough gameplays to be included in the categorical popularity list after being removed from the new list. In practice however, we see that very few games actually shift from the new list to its categorical popularity list. This could either be an indication of perfect popularity lists or an imperfection in the method that is used. To understand why this happens, we take a closer look at the effect of the popularity measure on newly added games using the recursive definition of the EMA as given in Equation 2. Since five games are added each day and the new list consist of sixteen games, a game is in the new list on four different days. We find a relation between the gameplays on the first day ($x_0$) and the EMA score after four days ($z_3$) using the following equation.

Equation 3. Effect of EMA over a short period

$$z_3 = \lambda x_3 + (1 - \lambda) z_2$$
$$= \lambda x_3 + (1 - \lambda)(\lambda x_2 + (1 - \lambda)z_1)$$
$$= \lambda x_3 + (1 - \lambda) \lambda x_2 + (1 - \lambda)^2 z_1$$
$$= \lambda x_3 + (1 - \lambda) \lambda x_2 + (1 - \lambda)^2 (\lambda x_1 + (1 - \lambda)z_0)$$
$$= \lambda x_3 + (1 - \lambda) \lambda x_2 + (1 - \lambda)^2 \lambda x_1 + (1 - \lambda)^3 z_0$$
$$= \lambda x_3 + (1 - \lambda) \lambda x_2 + (1 - \lambda)^2 \lambda x_1 + (1 - \lambda)^3 x_0$$

Recall that in the Tibaco setting $\lambda = \frac{2}{29}$, so $(1 - \lambda)^3 x_0 = 0.807 x_0$. This means that the weighted EMA score after the fourth day consists of the gameplays on the first day for 80.7%. Since the gameplays of a new game of the first day are not over the same period as an existing game these are incomparable and if the first (and last) day’s gameplays are included in the popularity score, it is estimated too low, resulting in the game not being added to the categorical popularity list.

We illustrate this by a concrete example. Suppose a game has a steady popularity of 1000 gameplays every day and it is published on the first day at 3PM. On the first day 430 gameplays occurred between the time of publishing and midnight. Using Equation 3 we calculate the popularity score on the moment the game has cycled through the entire list of new games and will possibly be moved to the categorical popularity list.

Equation 4. Example of EMA calculation after publishing

$$z_3 = \lambda x_3 + (1 - \lambda) \lambda x_2 + (1 - \lambda)^2 \lambda x_1 + (1 - \lambda)^3 x_0$$
$$= 1000 \lambda + 1000(1 - \lambda) \lambda + 1000(1 - \lambda)^2 \lambda + 430(1 - \lambda)^3$$
$$= 1000 \cdot \frac{2}{29} + 1000 \cdot \frac{27}{29} \cdot \frac{2}{29} + 1000 \cdot \left(\frac{27}{29}\right)^2 \cdot \frac{2}{29} + 430 \cdot \left(\frac{27}{29}\right)^3$$
$$= \frac{2}{29} \cdot 1000 + \frac{27}{29} \cdot \frac{2}{29} \cdot 1000 + \left(\frac{27}{29}\right)^2 \cdot \frac{2}{29} \cdot 1000 + \left(\frac{27}{29}\right)^3 \cdot 430$$
$$= 539.98$$
From this example we see that the popularity score by calculation of the EMA is 539.98, which is much lower than the true popularity of 1000.

Now that we have identified the problem, the remainder of this section will attempt to find a solution. The approach we take is to scale the gameplays of the partial day to gameplays of an entire day to give equal chances to games regardless of their time of publishing.

For both the first and the last day of a game’s period in the new list we are interested if we can scale the gameplays of a limited period to estimate the amount of gameplays the game would have achieved on the scale of a day. To generalize the idea, we set up a model that estimates the gameplays for any target period, based on an arbitrary source period. In the approach we take, we will first set up a model to determine the distribution of gameplays based on historical data and then use this distribution to scale the observed gameplays in a limited period to an estimate of the daily gameplays. Finally we evaluate this approach by determining the average error in the estimation using a variable period.

4.3. Model
We set up a model that comprises the cumulative distribution of gameplays over the day. This distribution is based on the timestamps of the gameplays that are stored in the database. We accumulate the amount of gameplays grouped by their timestamp to get the distribution required for our model. The model describes the amount of gameplays up to a certain point in time. We derive a distribution for each day of the week, because we see from the graph below that there is a clear difference in the distributions. The most obvious distinction is between the weekend (purple and pink line) in relation to the rest. Furthermore we see that the gameplays on Wednesday deviate from about halfway the day.

![Figure 9. Cumulative distribution of gameplays](image)

Figure 9 visualizes the model as a function of the fraction of gameplays up to a specific time and indirectly represents the fraction of gameplays in a certain time interval by taking the difference between two points. Based on this model, one can estimate how many visitors will play a game during the entire day by scaling the gameplays in the period by the same factor that is needed to scale the fraction of gameplays in the model to 1.

As an illustrative example we see in the figure below that based on the model, a game achieves 8% of its daily gameplays on Sunday between 9AM and 10AM. The actual amount of visitors that is...
observed in this time period (say 100) is assumed to be equal to the 8% of the daily gameplays as observed in the past. This means that 100% of the daily visitor is estimated at $100 \times \frac{1}{0.08} = 1250$.

![Figure 10. Fraction of total gameplays](image)

This far the model does not include target groups. This means that the projected gameplays may neglect the influence of a changing visitor distribution of visitors that have different interests. We modify our model to accommodate for this change in distribution during the day. We derive different distributions for each of the target groups as differentiated by Tibaco in the past (see Appendix B). To illustrate that there is indeed a significant difference between distributions of the target groups, we plot the overall gameplays of the target Age10-13 (pink line) and Age41-50 (green line) on a Wednesday.

![Figure 11. Illustration of difference between target groups](image)

From these distributions we can clearly see that the younger visitors make a rapid increase after 12 hours which can be explained by the time at which the school ends on a Wednesday, while the older visitors are more equally distributed over the day and achieve a significant amount of gameplays in the late hours. To accommodate for this difference in behavior, we compute the distribution for each target group separately, allowing for adding up the partial estimations of the different target groups.

Note that visitors are grouped in this manner based on Tibaco’s earlier findings involving the size of each group. Although it is important to have a significant amount of people in each group, the exact splits might be suboptimal for this type of application.
A large group of people have never filled out the questionnaire, which means they cannot be assigned to one of the target groups. The model allows for different options how to handle this group. It can be included in the model as a separate target group, allowing this unknown group to be handled as another target group, but it also allows redistributing the observed gameplays of this unknown class over the known classes. If we assume that this unknown target group has the same proportions as the known target groups, we can redistribute these gameplays over the known target group and estimate according to the respective gameplay distribution of that known target group.

4.3.1. Formal approach
In this section we give a more formal approach to the model introduced in the previous section. Let \( p_t \) be the fraction of gameplays with timestamp \( t \), where \( t \) represents the daytime in seconds. Then the model represents a cumulative function which is defined on the domain \([0..86400]\) as follows.

\[
f_s = \sum_{0 \leq t < s} p_t
\]

If we add the distinction between days we get the following function, where \( d \in [0..6] \) indicates the day.

\[
g_{d,s} = \sum_{0 \leq t < s} p_{d,t}
\]

Adding target groups \( a \in [0..8] \) to this function yields

\[
h_{d,a,s} = \sum_{0 \leq t < s} p_{d,a,t}
\]

Note that \( h_{d,a,t_1} - h_{d,a,t_0} \), where \( t_0, t_1 \in [0..86400] \) and \( t_1 > t_0 \) represents the fraction of daily gameplays of the target group in the time interval \([t_0..t_1]\).

This means we derive 63 different distributions; one for each of the nine age groups (Appendix B) for each day of the week. To use these functions to estimate the gameplays \( \bar{g}_{ss,se,ts,te,d} \) in a (possibly inclusive) target period \([t_0..t_1]\) on day \( d \) based on a source period \([s_0..s_e]\) we use the following formula to include targeting and treating the unknown visitors as a separate target group. Let \( g_{p,a,ss,se} \) be the observed gameplays in the source period \([s_0..s_e]\) by target group \( a \).

\[
\bar{g}_{ss,se,ts,te,d} = \sum_{a \in \text{Target}} \left( g_{p,a,ss,se} \cdot \frac{h_{d,a,te} - h_{d,a,ts}}{h_{d,a,se} - h_{d,a,ss}} \right)
\]

To include the redistribution of the gameplays of the unknown target class we scale the gameplays of each known target class to include the fraction that this known target class represents under all known target classes. Let \( g_{p,a,ss,se} \) be the amount of gameplays by target group \( a \) in the source period \([s_0..s_e]\) and \( u \) be the unknown target group.

\[
\bar{g}_{ss,se,ts,te,d} = \sum_{a \in \text{Target}} \left( g_{p,a,ss,se} + g_{p,a,ss,se} \cdot \frac{g_{p,a,ss,se}}{\sum_{a \in \text{Target}} g_{p,a,ss,se}} \cdot \frac{h_{d,a,te} - h_{d,a,ts}}{h_{d,a,se} - h_{d,a,ss}} \right)
\]

4.4. Applications
How to use this technique is open for multiple options, but its main application is in minimizing the loss of data due to the granularity of the current popularity metric algorithms. This section will suggest two concrete applications and shortly mention a third. These applications aim for projecting
gameplays of new games on the same scale as existing games to make them comparable. Once the

gameplays of new games can be compared with the existing games using the current popularity
score, it is possible to compare the projected value to the score of the lowest ranked game in the

current popularity list to see if the new game is expected to take over the position of this lowest
ranked game, i.e. whether or not the new game should appear in a popularity list.

**Day of publishing**

On the day on which a game is being published, the game has only had a limited amount of time to

gain gameplays, i.e. the time after which it was published. This means that compared to other games
the newly added game has little chance of obtaining a comparable amount of gameplays on this day.
Based on the findings in Equation 3 and Equation 4, we can state that it is important to scale the
gameplays of the new game to represent a projected daily total. For this type of application we can
use the formula in the following way.

\[
\hat{GP}_{t,86399,0,\Delta t-1,d} = \sum_{a \in \text{Target}} \left( \frac{gP_{a,t,86399} + gP_{t,86399}}{\sum_{a \in \text{Target}} gP_{a,t,86399}} \right) \frac{h_{d,a,\Delta t-1} - h_{d,a,0}}{h_{d,a,86399} - h_{d,a,0}}
\]

**Day of evaluation**

The same approach can be used to find the estimated total gameplays on the day on which the game
will leave the new list and the choice will be made whether or not the game has achieved enough
gameplays. These games have data available from midnight up to the point of analysis. For this type
of estimations we can use the formula as indicated below.

\[
\hat{GP}_{0,t,t+1,86399,d} = \sum_{a \in \text{Target}} \left( \frac{gP_{a,0,t} + gP_{a,0,t}}{\sum_{a \in \text{Target}} gP_{a,0,t}} \right) \frac{h_{d,a,86399} - h_{d,a,t+1}}{h_{d,a,t} - h_{d,a,0}}
\]

**Generalized approximation**

The method to scale gameplays has no restriction on the length of the period or the start and end of
the period. It is therefore also possible to apply this technique on an arbitrary source period and an
arbitrary target period. It is for example possible to use data of a three hour time span starting at
9AM and ending at 12 noon to predict the gameplays between 3PM and 8PM. This type of
application has not been evaluated, because that would involve an evaluation with many possible
combinations of starting and end points, both for the source period as for the target period.

**Index potential estimation**

Due to its general approach this method can provide estimations under targeting restrictions to scale
the target groups’ observed gameplays to a daily estimation of gameplays for that target group. Daily
gameplays are weighted exponentially over time to be used to determine which games are displayed
on the index. Using the estimated target group’s gameplays we can estimate the popularity score
which may be merged into the current game popularity rankings for each target group to determine
if this game would end up in one of the categorical popularity lists after leaving the list of newly added games. For this application we use a form of the formula that estimates for a single target group, without redistributing the unknown users, since we are only interested in the gameplays that count as a target group.

\[
\bar{GP}_{a,ss,se,ts,86000,dt} = GP_{a,ss,se} \cdot \frac{d_{ate} - d_{ats}}{d_{ase} - d_{ass}}
\]

As an example we estimate the gameplays of target group F3 (gender: Female, age group: 3) for game 14809 on different time intervals. The actual gameplays over the entire day is 10.

The amount of gameplays between 12 noon and 12 midnight equals 8. We use this to approximate the gameplays between 12 midnight and 12 noon.

\[
GP_{F3,12,24} \cdot \frac{h_{2,F3,12} - h_{2,F3,0}}{h_{2,F3,24} - h_{2,F3,12}} = 8 \cdot \frac{0.197}{0.803} = 1.96
\]

\text{Daily: } 8 + 1.96 = 9.96

The amount of gameplays between 3PM and 12 midnight equals 6. We use this to approximate the gameplays between 0AM-3PM.

\[
GP_{F3,15,24} \cdot \frac{h_{2,F3,15} - h_{2,F3,0}}{h_{2,F3,24} - h_{2,F3,15}} = 6 \cdot \frac{0.394}{0.606} = 3.9
\]

\text{Daily: } 6 + 3.9 = 9.9

The amount of gameplays between 9AM and 12 midnight equals 10. We use this to approximate the gameplays between 12 midnight and 9AM.

\[
GP_{F3,9,24} \cdot \frac{h_{2,F3,9} - h_{2,F3,0}}{h_{2,F3,24} - h_{2,F3,9}} = 10 \cdot \frac{0.0769}{0.923} = 0.83
\]

\text{Daily: } 10 + 0.83 = 10.83

4.5. Evaluation of the estimation

A program for defining the model has been implemented. Without it, the evaluation cannot be completed. Two types of evaluation have been performed. The first is the forward approximation based on a source period that starts at midnight and is used to predict the daily total. The second is backward approximation that uses a source period that ends at midnight to estimate the daily total.

Forward approximation

The forward approximation of the gameplays is suited for obtaining an estimate of the daily gameplays where gameplays are known up to a certain time. This can be used to estimate the gameplays on the remainder of the current day. The error of the estimation is shown in Figure 12. The horizontal axis represents the period after midnight up to which is included in the source data. The vertical axis is the average error relatively to the true value. We see that for the different game publishing times (9AM, 12 noon, 3PM) the variant with redistribution of the unknown target group has a smaller average error than the estimation without redistribution.
We also see that as long as we do not take a short source period, we can estimate the popularity of a game with sufficient accuracy. On average, at 9AM the daily gameplays can be estimated with an error of 20.5%, which is not extremely accurate. However, taken into consideration that this estimation is based only on 13% of the daily gameplays we are not displeased about the result.

**Backward approximation**

This type of approximation is suited for estimating the daily gameplays when a game is published during the day. Data is missing from the period before the time of publishing and needs to be estimated to approximate the daily gameplays required for the popularity calculation. The evaluation of this type of application is measured by estimating the daily total, based on the data from the time of publishing to midnight. The horizontal axis represents the time of publishing and the vertical axis again represents the average error relatively to the true value. Again we see that for the different game publishing times the variant with redistribution of the unknown target group has a smaller average error than the estimation without redistribution.
Just like with the forward approximation we see that for the backward approximation it also holds that once we take a sufficiently large source period of data, we can estimate the daily gameplays quite accurate. In this case the accuracy is extremely high for the time spans related to the publication times at 9AM, 12 noon and 3PM with an average error of 2%, 3% and 5% respectively.

Recall the first research question Q1:

Based on partial daily data, can we estimate the popularity of a game?

Considering the low relative error we conclude that the described method for estimation can indeed be used to estimate the popularity of a game of which we only have data of a part of a day; especially since we are primarily interested in using the backward approximation for the estimation of the daily gameplays on the day of publishing, which has an extremely low relative error. In the next subsection we answer research question Q2:

Can we use the estimation of Q1 to improve the constitution of the index lists?

4.6. Relation between new list and categorical popularity lists

In the previous section a method is given to estimate gameplays to help determine whether or not a game will enter the categorical popularity list. Although this initial placement may be corresponding to the observed gameplays, we may see that the game does not achieve enough gameplays once it is in the categorical list. This does not mean that the estimation was wrong, but this is caused by the different boosts in gameplays that games get from the popularity lists. The newly added games are clearly visible on the index, while other popularity lists require more effort to see. This gives rise to defining a relation between gameplays while the game is in the new list and gameplays while it is in the categorical popularity list. We want to find functions $f_0 \ldots f_{10}$ for each of the categories that
capture the decrease of gameplays once a game is moved from the new list to the categorical popularity list.

More formal we are looking for functions $f_0, ..., f_{10}$ that estimate the gameplays a game would obtain in the categorical popularity list based on its gameplays in the new list, which is depicted in the Equation 7. To compensate for the difference in the total amount of visitors between days, we also state a function based on the fraction of total gameplays in Equation 8.

\[
\hat{g}_i = f_i(g_{new})
\]

\[
\hat{g}_{frac} i = f_i(g_{frac new})
\]

Because Tibaco is increasing the amount of daily added games, it is vital to make a quick elimination of games without potential. Because Tibaco is primarily interested in integrating this technique on data of the first day’s gameplays, we limit our approach to finding a relation to gameplays on the first day in the new list, in relation to the gameplays on the first day of the categorical popularity list. No data is recorded that explicitly states the publishing date of a game, so we stored the index page over a period of six months in order to extract the game positions on the website. In this period, over 700 new games have been added, but we see that only 42 of these games have actually made the step to the categorical popularity lists. These 42 games belong to 12 different categories and therefore we only have a small amount of samples per category. The category with the most samples is category racing. We will illustrate the approach using this category, but first we will state that the exact values used in this approach are not conclusive.

To compensate for the difference in the amount of visitors on the day the game was in the new list and the day the game was in the categorical popularity list, we use the fraction of total gameplays to
determine the relation between the lists. If we plot the estimated fraction of gameplays in the new list (vertical axis) against the fraction of gameplays in the categorical popularity list (horizontal axis), we obtain Figure 15.

![Figure 15. Linear regression to obtain a relation between the lists.](image)

The blue line indicates the result of simple linear regression. In this case, the relation obtained by regression is given in

**Equation 9. Relation between new list and categorical popularity list 2**

\[
g_{\text{frac}}^2 = f_2(g_{\text{frac}}^\text{new}) = -0.0001 + 0.3744g_{\text{frac}}^\text{new}
\]

Once we obtain more samples we might see that the offset in the above equation is no longer applicable and the equation can be reduced to a single scaling. We can use these scale factors as samples to set up a confidence interval on our scalar value. Even if we take a confidence level as low as 50% on the data of category *racing* we obtain a confidence interval of \([0.22, 0.37]\).

![Figure 16. Confidence interval on scale factor. 50% [0.22; 0.37], 70% [0.18; 0.41], 90% [0.11; 0.48]](image)

Once we are able to obtain more samples, the interval is expected to narrow. We can only obtain this information by obtaining more historical index pages. In an attempt to acquire more of these index pages, we attempt to recalculate the popularity scores and rebuild the index pages using the earlier discussed site generation module (Figure 8). The input consists of site specific overall and site non-
specific targeted exponentially weighted moving averages. Note that in the site specific overall EMA we do not use the sum of the individual transactions, but instead we use the materialized view (Section 3.2) in order to do the same calculation as was done by Tibaco when the website was generated. For the recalculation of the popularity scores this does not lead to any problems, because the materialized view is available for historical dates and can be used in the recalculation.

The problem is with the target EMA scores. Visitors are either male, female or unknown. We expect that the sum over all male age groups (M0...M8) add up to the male total (M). However, if we look at the values stored in the current database, we see that this is not the case. We take an arbitrary game (11444) and retrieve the result of the last EMA calculation from the database, which is 59.97 for the male EMA score. Now if we look at the table below, we see that the left column is directly retrieved from the database as information used for the generation of the website and adds up to 19.4 over all age groups of males. This difference is remarkable, even more if we recalculate the target EMA scores using transactional information; we see that our sum adds up to 59.8 which is much closer to the total of the male group (59.97).

<table>
<thead>
<tr>
<th>Targeting group</th>
<th>EMA score (database)</th>
<th>EMA score (recalculated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M0</td>
<td>1.8</td>
<td>1.9</td>
</tr>
<tr>
<td>M1</td>
<td>3.9</td>
<td>21.9</td>
</tr>
<tr>
<td>M2</td>
<td>2.4</td>
<td>19.8</td>
</tr>
<tr>
<td>M3</td>
<td>1.7</td>
<td>2.6</td>
</tr>
<tr>
<td>M4</td>
<td>2.0</td>
<td>2.1</td>
</tr>
<tr>
<td>M5</td>
<td>1.8</td>
<td>4.4</td>
</tr>
<tr>
<td>M6</td>
<td>2.0</td>
<td>3.8</td>
</tr>
<tr>
<td>M7</td>
<td>1.8</td>
<td>1.3</td>
</tr>
<tr>
<td>M8</td>
<td>1.9</td>
<td>2.0</td>
</tr>
<tr>
<td>Sum</td>
<td>19.4</td>
<td>59.8</td>
</tr>
<tr>
<td>M</td>
<td>59.97</td>
<td>59.97</td>
</tr>
<tr>
<td>Error</td>
<td>40.57</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Table 6. Database EMA (left) recalculated EMA (right)

Using the definition of the EMA given in Equation 2 we show that if the age-gender gameplays add up to the gender total, this property is maintained under the EMA. We start off by stating that on a single day $t$ all gameplays grouped by their age-gender target group $i$, sum up to the total gender target group gameplays recorded in the database. We use $x_{i,t}$ to denote the gameplays of age-gender target group $i$ and $x_{t}$ to denote the gender target group’s total gameplays.

\[
x_{t} = \sum_{i=0}^{8} x_{i,t}
\]

Equation 10. Sum over age groups add up to gender total gameplays.

Next we use the definition of the initial value of the EMA score (Equation 2) to prove that the age-gender gameplays should add up to the gender total on the first day.
Next we use an iterative proof to show that the same holds for each day after the first day.

$$z_{t+1} = \lambda x_{t+1} + (1 - \lambda)z_t$$

$$= \lambda \sum_{l=0}^{8} x_{l,t+1} + (1 - \lambda) \sum_{l=0}^{8} z_{l,t}$$

$$= \sum_{l=0}^{8} (\lambda x_{l,t+1} + (1 - \lambda)z_{l,t})$$

$$= \sum_{l=0}^{8} z_{l,t+1}$$

From the above proof we conclude that the summation property should hold, but it does not hold in the Tibaco data and therefore the age-gender EMA used by Tibaco can only be correct if there are many people (68%) that have filled out a gender, but not their age, which is highly unlikely. After all, our recalculated sum does add up to the male total. It is more likely that there is an inconsistency with the calculation of Tibaco and since we cannot recompute this difference it is not possible to generate old index pages to increase the amount of samples for determining the effect on the new list in relation to the categorical popularity list.

### 4.7. Future Work

Since we currently do not have enough samples, the topic of the popularity estimation has not been fully completed. The amount of samples will increase over time, because there will be more and more games that move from the new list to the categorical popularity list. Note that at the current rate, it might take several months or even several years before enough samples have been acquired.

Besides finding the relation between the new list and the categorical popularity lists once there are more samples, we will now suggest other subject of future work that may improve performance. As stated before, the age borders that are used in defining the different target groups are the groups that are predefined by Tibaco. This division is based on the quantity of visitors in each of the groups. It is most likely suboptimal for this type of application, but in principle forms the basis of the division of visitors according to the available playtime caused by their environments in the sense that it accommodates for school times for the younger groups and activity in the evening hours for visitors with a higher age. Fine tuning these boundaries may provide slightly better accuracy, however, at the same time we can see that the current age groups differ in a moderate way and there is no reason to expect a major increase in performance by redefining these boundaries.

It is currently being assumed that the unknown group of visitors is distributed the same way as the known group of users, while in fact it might be that for example teenagers are more likely to fill out their information than adults. Although we see that this assumption leads to slightly better results,
we can examine how we can explain the unknown target group’s gameplays using regression techniques with the known target groups’ gameplays as regressor variables.

4.8. Conclusion on the popularity estimation
In this section we have given an approach to estimate whether or not a newly added game is popular enough to survive in its categorical popularity list. We took the approach of first scaling the gameplays of a partial day to the amount of gameplays on the entire day and then relate this value to the expected gameplays once it would be placed in the categorical popularity list. The model that estimates the gameplays to the time span of an entire day are sufficiently accurate, especially on the source periods related to the current times on which games are being published. Therefore we answered research question Q1 affirmatively.

For the final step of determining the change in gameplays resulting from switching lists, we have to conclude that we have too few data available to determine the effect. Without this final step, the described method should not be integrated in the decision making regarding the popularity of a game. This would compare new games to the games in the categorical popularity list, which profit from different boosts from these lists, resulting in many new games being moved to the categorical popularity list. Although this will provide many samples for determining the boost effect between lists, we would not suggest doing so because adding games to the index page that do not belong there directly affects the user experience. Reconsider research question Q2:

Can we use the estimation of Q1 to improve the constitution of the index lists?

At the current moment in time, we have to conclude that we have not fully succeeded in doing this. However, we have sketched an approach on how this can be done in the future.
5. Broken game classification

While searching for a relation between the current popularity measure (gameplays) and the average playtime of a game, there seemed to be a clear relation between these two types of data. Exceptions to this relation appeared often to be broken, which lead to the hypothesis that games with a low average playtime and a high number of visitors have a higher chance to be broken. Intuitively this can be explained by the fact that users will not be playing a broken game for an extensive period and are likely to press the refresh button, hoping for the game to appear correctly.

This chapter will focus on investigating whether the above relation indeed exists. It will lead to an exploratory search for the best approach to construct a model for classification. We aim to answer research question Q3:

Can we use game statistics to detect broken games?

The discussed choices are Logistic Regression, a Decision Tree and a Naïve Bayes classifier. The objective of the classification is to present truly broken games in a certain ratio (or better) to games falsely classified as being broken. Rationale behind this approach is to control a certain benefit per manually checked game.

5.1. Attributes

Each game has a certain amount of metrics/statistics available. The data is based on the earlier introduced tables in Section 3.1: Available Data. We select the following game statistics as attributes.

- Game ID (identifier)
- Game Name (identifier)
- Count: Amount of gameplays
- Seconds: Total seconds played

From this data we deduct two more attributes. However, before stating these two attributes, we need to look into the data characteristics with more detail. The heartbeat timeout method of the session server causes several playtimes to be equal to zero or any multiple of the timeout interval. This influences the average playtime and although the influence is expected to be uniformly distributed over all playtimes and therefore proportional to the amount of gameplays, the variation in this distribution will cause some games to have a relatively large amount of these playtimes. The ratio of broken games vs. non broken games will magnify this phenomenon, leading to a suboptimal split. Therefore we deduce slightly modified attributes count and average playtime, the latter of which represents the average of all playtimes that are unequal to any multiple of the heartbeat interval. Let \( g \) be a game with attributes and \( h \) be the heartbeat interval.

Count: \( \sum_{(g.id, g.seconds \mod h \neq 0)} 1 \)

Average Playtime: \( \frac{\sum_{(g.id, g.seconds \mod h \neq 0)} g.seconds}{\text{Count}} \)

Ratio: \( \frac{\text{Average Playtime}}{\text{Count}} \)

Furthermore, there exists a button on the website where users may indicate a broken game. It is expected that the clicks on this button is highly correlated with a game being broken, even more so.
for the fraction of visitors that reports the game as being broken (ReportsRatio) as to eliminate the bias towards popular games.

Reports: Amount of broken game reports

\[
\text{ReportsRatio} = \frac{\text{Reports}}{\text{Count}}
\]

The Average Playtime and Ratio are strictly dependent on other attributes. It may therefore seem that they do not add anything new. The reason that these attributes are included is that a decision tree performs badly on linear dependencies. Because we expect a relation involving the average playtime and possibly a linear relation involving ratio, we add these metrics and see later on whether these attributes are indeed important. If the linear relation is significant, the decision tree model will include a split on this attribute.

In Figure 17 the yellow line shows the ideal division between green and red points. Because a decision tree can only make horizontal and vertical splits, the resulting tree will use splits like the blue lines. Using the ratio, the decision tree gives a simple classification using a horizontal split. Average Playtime and Ratio are added as attributes because it may be important attributes, but cannot easily be used by a decision tree based on the initial attributes.

5.2. Comparison of methods

We will compare three different methods for classification: decision tree classification, logistic regression and naïve bayes. These models are compared using their ROC-curves. The curves are displayed in Figure 18. ROC curves are constructed by first ordering the games by their positive class probability in decreasing order. Consequently, for each game it is determined whether the game is truly broken or not. Upon encountering a true positive the line goes up one step; on encountering a false positive the line goes one step to the right. The y-axis therefore represents the true positive rate and the x-axis represents the false positive rate. The ideal graph would go close to (or on) the top left corner. One property that follows is that a high Area Under Curve (AUC) describes a good model. The grey diagonal represents a random classifier, the worst possible model. Curves below the diagonal can be mirrored, so they end up above the diagonal. Based on the value of the AUC metric, logistic regression would be the best model, followed by the naïve bayes classifier. The decision tree model performs the worst on the AUC metric.
Remarkable is the early increase of true positives on the decision tree model. The choice of which model to choose will not be based on the AUC, but rather on the ratio of true positives related to false positives to ensure certain efficiency on manually verifying the suggested broken games. This is an important factor for trust in the classification. An ROC curve is normalized on both axes. In our case, this means that there is a scale difference of a factor 60, where FPR 1 correlates to 60 times as many games as TPR 1. First we deduct a coefficient for the desired ratio of TP/FP. This ratio is based on the portal manager’s opinion on what is an acceptable efficiency. The recall of broken games is considered less important. The classification will be a supporting mechanism that should not take too much time of human resources.

The ratio used is at least 1 true positive on every 4 suggested broken games. This corresponds to a coefficient of 1/3 as displayed in Figure 19. To normalize this coefficient on the ROC curve scale we use the following equation:

\[ c_{norm} = \frac{c \times \#FP}{\#TP} \]

\[ = \frac{c \times \frac{\text{amount of non defects}}{\text{amount of defects}}}{1} \]

\[ = \frac{1}{3} \times \frac{6784}{111} = 20.4 \]

The tangent line of the ROC curve indicates the point where the model has the desired precision. A tangent line with a higher Y-axis intersection indicates a better model, because it has a higher recall.
for the same tangent line. In this case, the blue line, the decision tree has the best properties. A threshold can be estimated optically after which another model becomes preferred. Appendix C indicates that this threshold is around a $c_{\text{norm}}$ of 3.15, which corresponds to $c=0.052$. This means that on average, 20.4 games have to be checked before a truly broken game is found. This leads to an unacceptable amount of overhead work. For this reason, the decision tree is chosen as most preferred method.

5.3. Classification Model

Based on the previous section, the method to be used is the decision tree. Splits will be determined using the GINI-index [13]. MDL pruning [14] is applied to prevent overfitting. Overfitting causes the model to take too many details into account, while the objective is to construct a generally applicable model. This complicates the model and causes it to be less accurate, as these details are often unstructured and are not reproduced during evaluation or in practice. Figure 20 shows the effect of overfitting when using a decision tree. The left model is often preferred.

![Figure 20: Overfitting (right)](image)

5.4. Description of the model

The resulting model is depicted in Appendix A. The decision tree contains no split on the derived attribute Ratio. Therefore is seems of less significance than the original attributes. Furthermore, it does not make a split on attributes Reports or ReportsRatio, which is counter intuitive. One would expect that people tend to click this button upon seeing a broken game. The lack of correspondence can possibly be explained by the appearance of the button (flag in Figure 21). It may not be noted by the visitor as a button to report a broken game and may therefore also be clicked when a game is not broken.

![Figure 21. Button to report broken game (flag)](image)
The first split in the model is on average playtime. An average playtime of more than 47.1 seconds is an indication that the game is not defect in 99.5% of the cases. This split causes 31 games to be wrongly classified as not broken. The next split in the model is count. This primarily filters out the games that share the low average playtime, but only because it’s a sample mean based on few samples. This branch classifies correctly in 96.7% of the cases. The remaining part of the model may be explained by the same principle as above, but for more gameplays, which further restricts the average playtime. This part of the model will be the subject of refinement in the next section. Since the decision tree only consists of two dimensions, it can be displayed in a two dimensional space.

5.5. Evaluation

The decision tree method is evaluated under cross validation, which causes a higher reliability and is a better representation of the performance in practice as it separates the training data from the test data and still classifies each game.

The results under cross validation are a precision of 67.3% and a recall of 29.7%. This performance is based on data from the same day as the training data. The training and test dataset are made independent by the partitioning of the cross validation, but still share certain characteristics. It is interesting to see how the model performs on data that has other daily characteristics.

The model performs extremely well on this data with a precision of 65% and a recall of 23%, although the recall is still quite low. By default, games are classified as broken when \( P(\text{true}) \geq 0.5 \) because then \( P(\text{true}) \geq P(\text{false}) \). This threshold can be changed which modifies the precision and recall. A lower threshold for \( P(\text{true}) \) will increase the recall, but will also classify more non-broken games as broken.

The ROC-graph shown earlier is a graph involving the cross validation results. Therefore it contains points that are not in the final model. The ROC curve of the final decision tree model on the new independent dataset is shown below. Upon given the choice between the two highlighted points, the portal manager preferred the higher point of the two, leading to a precision of 35% and a recall of 37%.
5.6. Cost Sensitive classification

The main discrepancy in the previous model is caused by the difference in the goal of learning the model and using the model to predict defects. The usage of the GINI index to determine splits causes the learner to optimize overall accuracy (including non-defect predictions), while the evaluation focuses on maximizing recall of the positive class while under a precision constraint of that same positive class. Using cost sensitive classification [15] may lead to splits that better fit the goals.

Based on the same required efficiency as before (at least 1 TP out of every 3 FP), we state that every false positive has a cost of 1 as opposed to every true positive having a cost of -3. This leads to the following cost matrix.

<table>
<thead>
<tr>
<th>Broken\Prediction</th>
<th>False</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>True</td>
<td>0</td>
<td>-3</td>
</tr>
</tbody>
</table>

Table 7. Cost matrix

This cost matrix defines a cost function

\[ c(fp, tp) = fp - 3 \, tp \]

which is minimized by the cost sensitive classifier. The resulting decision tree is depicted in Appendix E. This model can be explained analog to the previous model, but with other split values and has one remarkable addition: the inclusion of the reports ratio. This model has a precision of 53% and a recall of 38% under cross validation and a precision of 61% and a recall of 40% on the validation set; a slight improvement in recall, but a significant improvement in precision w.r.t. to the overall accuracy model.

5.7. Implementation

To ensure a fast and simple implementation, it is possible not to recompute the model over and over again. The implementation can be based on this classification as it is expected that properties of broken games do not change over time and most splits in the model are proportional to the amount
of gameplays. Therefore the splits are insensitive to a future increase of gameplays, except for the gameplays and seconds attribute. If changes are detected in the future on this attributes, it may be necessary to recompute the model, because the optimal borders of the decision tree may have changed as well.

The implementation of a program or script that predicts which games are broken is quite easy once the model has been set up. Note: See Section 5.1 for an interpretation of the attributes and Appendix F for a MySQL query representation.

```java
for (i=0; i<g.length; i++) {
    path1 = g[i].reportsRatio <= 0.0046 &&
            g[i].avgPlaytime <= 27.95 &&
            g[i].count > 12
    path2 = g[i].reportsRatio > 0.0046 &&
            g[i].avgPlaytime <= 74.80 &&
            g[i].count > 12
    path3 = g[i].reportsRatio > 0.0046 &&
            g[i].avgPlaytime <= 74.80 &&
            g[i].count <= 12 &&
            g[i].seconds <= 51 &&
            g[i].seconds > 21
    g[i].broken = path1 || path2 || path3
}
```

To prevent the same games from being falsely detected each day, it may be appropriate to supplement the above mechanism with a technique that gives detected games a time-out in which they will not be suggested as broken again.

### 5.8. Future work

There are some grounds for improving this classification. No distinction is made between different types of defects. Therefore the splits are influenced by defects that possess the unwanted properties, but also by defects that do not share these properties. This causes a game that has a defect in level 102 to be weighted in the same manner as a game that does not even load. The split Average Playtime is influenced by this behavior and therefore suboptimal.

Furthermore some games are not in the native language. Those games are close to impossible to play for a group of people. Introducing a new attribute that states that x% of the people played the game at least y seconds, may also be able to distinct between these games and broken games. However, this involves the fine tuning of a new parameter, which is left for future research.

### 5.9. Conclusion on the classification

The model confirms that a certain group of broken games has properties that include a low average playtime and a relative high amount of visitors. Although the latter is not necessarily caused by visitors reloading the game, but is rather an indication that games with few gameplays happen to share the same average playtime as broken games. Furthermore we see that in the cost sensitive classification the reports by visitors is the most important attribute and combined with splits on the above attributes leads to the best model.

The classification gives a list of suggested broken games (prediction: true, broken: false). A property of the data which constructs the model is that from a large group of games it was unknown whether the game was broken or not. A value false for broken in the training set does not mean the game is not broken, but only that we do not know if the game is broken. The model is therefore constituted
on partially unknown data. The model assumes the game is not defect, while in fact it might very well be broken.

The classification performs acceptably well on both precision 61% and recall 40%. This means that if the classification suggests a game as being broken, this is indeed the case in 61% of the cases. The recall measure indicates that 40% of the defects are expected to be found using this classification. Based on these statistics we can positively answer research question Q3:

Can we use game statistics to detect broken games?

As a final note we mention that the model is suited for detecting games with certain properties, which causes more and more broken games to be found in this region. Broken games which do not have these properties will not be found. Relying solely on this classification will cause broken games to remain in the safe zone. This will cause a new classification in the future to specifically classify according to the current classification. It is therefore recommended to manually scan all games before doing a reclassification, in case this does not cause a disproportionate amount of effort.
6. Recommender System

Although much of the available time has been shifted to the two previously discussed subjects, the main objective of this thesis was to develop a recommender system for the website Funnygames.nl. This website contains almost 16,000 games of which only a small amount of games can be offered to the visitor in a clear manner through game slots on the index page and several small groups of slots on game pages. By using visitor interests to match games to visitors we may use a fraction of the available slots to provide our visitors with the most appropriate games. In this chapter we will answer the following research questions.

Q4: How can we recommend games to visitors that have no gameplay history?

Q5: How can we recommend games to returning visitors, based on their gameplay history?

6.1. Hybrid

In general, collaborative filtering approaches are superior to classical nearest-neighbor approaches [6]. One of its downsides is that it suffers from the cold start problem, which means that it does not accommodate for new games or visitors which are not included in any previous observation. This means that we have to complement our collaborative filtering approach with another model that does not use user specific information and gives a more general recommendation to new visitors.

The general model is for new visitors. Recall from section 3.1 that in some cases, we see returning visitors as new visitors. In some contexts, we might use explicit profile information to still be able to make personalized (grouped) recommendations based on for example age, gender or geographical data. By the nature of which they are acquired, we make a distinction between these types of data by distinguishing between age and gender on the one side and geographical information on the other. The age and gender information is obtained by a questionnaire that never occurs on the first visit and with a chance of 1 out of every 3 after. So besides the fact that only a small portion of the visitors actually fills out this information, we never have this information on new visitors; unlike a registration profile would offer. The other option of basing recommendations on geographical information is expected to offer little distinction between a large group of visitors as most of the visitors origin from the country corresponding to the domain suffix.

A remaining option is to create an aggregated preference model that suggests the most preferred games over all users. We will first discuss what we exactly interpret as preference in the next section and then discuss the models that will use this notion of preference to suggest the most preferred games to the visitors.

6.2. Notion of preference

In the classical approach of a recommender system, users are asked to explicitly rate items. These ratings are then used to propose other interesting items. In our case we have no such information available. Visitors have never given their explicit opinion on any of the games. We do however have navigational information available that we can use as a visitor’s appreciation of a game. An essential question to answer is: what do we take as a degree of preference. In this subsection we will discuss our options.

The most straightforward options are either based on gameplays or playtime. Before discussing the actual model, we look at these two metrics and we derive a third, the average playtime of a game. In
the table below we name the pros and cons of the metrics. Then we will attempt to exploit the pros and limit the cons, to achieve a better notion of preference.

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Disadvantage</th>
<th>Gameplays</th>
<th>Playtime</th>
<th>Average Playtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct relation with income</td>
<td>Preference before playing</td>
<td>Preference while/after playing</td>
<td>Preference while/after playing</td>
<td></td>
</tr>
<tr>
<td>Preference before playing</td>
<td>Inaccurate wrt retry button¹</td>
<td>Inaccurate wrt heartbeat interval³</td>
<td>Inaccurate wrt heartbeat interval³</td>
<td></td>
</tr>
<tr>
<td>Heavily biased towards current popularity rank²</td>
<td>Inaccurate wrt external games⁴</td>
<td>Inaccurate wrt external games⁴</td>
<td>Sensitive value for low amount of gameplays</td>
<td></td>
</tr>
</tbody>
</table>

¹ Games that can be restarted inside the flash application will not count as a new gameplay, while a refresh of the page will count as such.
² Self-fulfilling prophecy
³ We reduced this inaccuracy in section 3.2.
⁴ The visitor may close the browser at funnygames.nl and continue to play on the external website.

Table 8. Pros and cons of available preference metrics

When making a choice for the preference metric, we find a balance between practical problems and theoretical fitness. The most important factor in the theoretical fitness is the point in time when the metric is achieved. The gameplay metric is achieved after viewing the name of the game and perhaps viewing a preview screenshots, but before playing the actual game. The gameplay metric is therefore rather a metric for an indication of liking a game’s first impression, rather than appreciating the gameplay experience itself. On the other hand, the playtime metric is a metric that is achieved after the game has been played and therefore represents a degree of liking the game itself, rather than solely its title or screenshots. Therefore we use the playtime and average playtime as notion of preference rather than gameplays.

6.3 General Popularity Model: Exploration-Exploitation

In this subsection we search for an answer to research question Q4:

How can we recommend games to visitors that have no gameplay history?

To accommodate for visitors with no gameplay history, we complement the collaborative filtering model with a general popularity model in the form of an exploration-exploitation scheme. This scheme will contain two principles: exploration and exploitation. Exploitation will take the currently known optimal values. Exploration tries to obtain more information to further optimize the exploitation process. We will discuss the exploitation approach first and then the exploration approach.
Exploitation

The average playtime is used as the most intuitive notion of game appreciation. The exploitation part contains the items with the highest average playtime. To prevent a few amount of visitors to cause a high average playtime, we set a minimum amount of visitors $\beta$ as a requirement for a game to be included in the exploitation part of our model. Formally we define an exploit score as follows. We define an observation as a combination of visitor $v$, game $g$, playtime $t$ that occurs in the transactional database. Let $S$ be the subset of observations containing game $g$, i.e. $S(g) = (v, g, t) \in Observations$.

$$\text{exploitation}(g) = \begin{cases} \frac{\sum_{(v, g, t) \in S(g)} t}{|S(g)|} & \text{if } |S(g)| \geq \beta \\ 0 & \text{otherwise} \end{cases}$$

Equation 14. Exploitation score

Exploration

A game is added in the new list once. If the game does not achieve enough gameplays in this short time span, it will be moved to the archive pages from which it is close to impossible to emerge back on the index page. In the past this comparison has been based on different time spans after publishing (section 4) and therefore required modifications. Recall that for this reason, we designed the method in section 4 to scale the gameplays to a time span of a day and then attempted to relate the amount of gameplays in the new list to the amount of gameplays in the categorical popularity list. The games that were added under these circumstances are still in the large collection of games on the archive pages. Note that not all statistics of these games have been recorded around the publishing date as they date from before the introduction of the data warehouse. Therefore the method of section 4 is not applicable and we supplement the exploitation model with an exploration model that attempts to find games that have the best potential of becoming popular, but are not (yet) included in the exploitation part.

The exploration will be based on Active Discovery of Black Boxes [16], but modified to be included in an exploration/exploitation setting based on a static environment using daily feedback rather than immediate feedback. For each game $g$ we determine the average playtime and standard deviation based on the currently known data.

$$\mu_g = \text{the sample mean of observed playtimes of game } g$$

$$s_g = \text{the standard deviation of observed playtimes of game } g$$

Next we define the threshold $\theta$, which intuitively equals the lowest value of the exploitation model, in order to represent the threshold needed to exceed the lowest exploited game.

$$\theta = \text{threshold, set to the lowest value included in the exploitation}$$

The measure used in Active Discovery of Black Boxes is the following:
Here, $|\mu_g - \theta|$ is the distance of the average playtime of a game with respect to the threshold. For games that have an average playtime lower than the threshold and equal variance, games closer to the threshold have a higher chance to end up above the threshold once we have more samples. Using the same logic on the variance, we expect that for games with equal distance, the games with the highest variance are the games that may exceed the threshold the most. So using this measure, we explore for games that are close to the threshold and/or have a high variance, i.e. games that have most potential to rise above threshold $\theta$. In order to prevent duplication of games in the list, we perform the exploration on all games excluding the ones represented in the exploitation step.

However, we need to make several modifications to the exploration score $a_g$, since we included a minimum amount of gameplays in the exploitation model. The current exploration score would treat games above the threshold equal to games below the threshold in the sense that games above the threshold but for which $|S(g)| < \beta$, we have an indication that our threshold $\theta$ would shift once this game would be played by more visitors, since this game would be included in the exploitation and has a higher $\mu_g$ than the current lowest $\mu_g$ above $\theta$. From the games above the threshold, but with too few gameplays to occur in the exploitation, the ones that have low variance and/or a high average playtime are more likely to stay above threshold $\theta$ when more gameplays occur. Therefore, we redefine our exploration score $a_g$ as follows.

\[
exploration(g) = \begin{cases} 
  a_g & \text{if } \mu_g < \theta \\
  \infty & \text{if } \mu_g = \theta \\
  a_g^{-1} & \text{if } \mu_g > \theta 
\end{cases}
\]

**Equation 16. Exploration score of a game**

The exploitation and exploration both generate an ordering of games that we need to combine into a single list of recommendations. Since we have only four available slots there is not much choice and we take a proportion of 0.25 to be explored and a portion of 0.75 to be exploited.

### 6.4. Personalized model: Collaborative filtering

Once we can identify a user by using the uuid in the cookie, we can relate previously played games to the current user. We will use this information to offer personalized recommendations to the user and answer research question Q5:

**Q5:** How can we recommend games to returning visitors, based on their gameplay history?

Currently matrix factorization is the most promising method of collaborative filtering when there are sufficient users and items included [6]. Note that in the classical collaborative filtering approach as used in the often referred Netflix contest, it is common to obtain explicit ratings from the user by using a feedback mechanism. In our case, there is no such feedback available and we will use the navigational log in the database to derive the notion of preference as stated in 6.2.
By using matrix factorization we model our large amount of visitors and games in a smaller common space of factors. To set up this common space of factors we decompose the observation matrix $P$ in two matrices.

- Matrix $X$ describing the factors of a visitor
- Matrix $Y$ describing the factors of a game

Combining these factors on unobserved combinations of visitors and games allows us to use the correspondence of factors to estimate the preference of the visitor for these games. We estimate the preference matrix $P$ by multiplying the user-factor matrix $X$ by the transposed item-factor matrix $Y$.

Equation 17: Preference matrix $\bar{P}$

$$\bar{P} = X^T \cdot Y$$

To compute matrix $X$ and $Y$ we follow the approach described in Collaborative Filtering for Implicit Feedback Datasets [8], which proposes a matrix factorization technique for implicit feedback contexts, using an alternating least squares approach [7]. Using this technique, we want to find a minimum for the following function.

Equation 18: cost function

$$cost = \min \sum_{ui} c_{ui} (p_{ui} - x_u^T y_i)^2 + \lambda \left( \sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

This function represents the sum of the confidence weighted squared error in preference estimation, supplemented by a regularization function to prevent overfitting. Here, $p_{ui}$ is a binary value that equals 1 if the visitor played the game and zero otherwise.

Equation 19: Definition of preference

$$p_{ui} = \begin{cases} 1 & \text{if } r_{ui} > 0 \\ 0 & \text{otherwise} \end{cases}$$

c_{ui} indicates the confidence we have in the preference. We take the definition of confidence from the paper [8] and relate this confidence to the playtime $r_{ui}$ as defined by the following equation.

Equation 20: Definition of confidence

$$c_{ui} = 1 + \alpha r_{ui}$$

To calculate the matrices $X$ and $Y$ we use the technique of alternating least squares [7] in order to find a minimum for the cost function of Equation 18. Using this technique we consider matrix $Y$ static when calculating matrix $X$ and vice versa, using the following equations for each row of $X$ and $Y$.

Equation 21: Minimizing the cost function

$$x_u = (Y^T C_u Y + \lambda I)^{-1} Y^T C_u p(u)$$

$$y_i = (X^T C_i X + \lambda I)^{-1} X^T C_i p(i)$$
In this equation \( p(u) \) is a vector containing the Boolean values indicating an observed preference and \( C \) is a diagonal item-item matrix containing the confidences for this user (analog for \( Y \)).

### 6.4.1. Optimizations

Any multiplication by zero is a useless operation that costs time. Typically people play only a small amount of games. We will introduce several data structures to use this knowledge to our advantage in order to minimize the waste of time on the computation of zero values. The basic approach will be to use dense matrix representations to accommodate for the user-factor matrices \( X \) and \( X^T \) and item-factor matrices \( Y \) and \( Y^T \), since these matrices are completely filled and to introduce special data structures for the other intermediate results.

The first optimization is proposed by the paper [8] by transforming \( Y^T C_u Y \) into \( Y^T Y + Y^T (C_u - I) Y \), where \( Y^T Y \) is user independent and can be pre-computed for each alternation. We used a confidence function that transforms all zero observations to 1. This results in a diagonal matrix \( C_u \) where games not played by this user are represented by a confidence of 1. By subtracting the identity matrix these entries in \( C_u \) are reduced to zero, resulting in a diagonal matrix with only \( n_u \) non-zero elements, where \( n_u \) equals the number of games user \( u \) played. When multiplying a matrix \( Y^T \) by such a sparse diagonal matrix, non-zero elements will only occur in the result matrix in the non-zero columns of the diagonal matrix. By storing the diagonal matrix in a sparse format, we can directly access the non-zero elements and multiply only those elements with the values in the corresponding column in \( Y^T \).

\[
Y^T (C_u - I) = \begin{bmatrix}
Y^T_{0,0} & \cdots & Y^T_{0,N} \\
\vdots & \ddots & \vdots \\
Y^T_{f,0} & \cdots & Y^T_{f,N}
\end{bmatrix}
\begin{bmatrix}
c_i^1

c_i^2
\end{bmatrix}
\begin{bmatrix}
Y^T_{0,11} * c_i^1 \\
\vdots \\
Y^T_{f,11} * c_i^1
\end{bmatrix}
= \begin{bmatrix}
Y^T_{0,11} * c_i^2 \\
\vdots \\
Y^T_{f,11} * c_i^2
\end{bmatrix}
\]

We take this optimization even further by noting that \( Y^T (C_u - I) \) contains only \( n_u \) non-zero columns. We use this to construct \( Y^T (C_u - I) Y \) by cycling only through all non-zero elements of \( Y^T (C_u - I) \) to use in the matrix multiplication. For this purpose we use a column sparse matrix format defined by a list of column indices and their column data, allowing us to skip columns consisting only of zero data.

\[
Y^T (C_u - I) Y = \begin{bmatrix}
Y^T_{0,11} * c_i^1 & Y^T_{0,12} * c_i^2 \\
\vdots & \vdots \\
Y^T_{N,11} * c_i^1 & Y^T_{N,12} * c_i^2
\end{bmatrix}
\begin{bmatrix}
Y_{0,0} & \cdots & Y_{0,f} \\
\vdots & \ddots & \vdots \\
Y_{N,0} & \cdots & Y_{N,f}
\end{bmatrix}
\]

Another optimization is acquired by using the associativity of matrix multiplication.

\[
((Y^T Y + Y^T (C_u - I) Y + \lambda I)^{-1} Y^T C_u p(u)) = ((Y^T Y + Y^T (C_u - I) Y + \lambda I)^{-1} Y^T) (C_u p(u))
\]

Note that \( p(u) \) contains at most \( n_u \) non-zero elements, since \( p(u) \) is defined as \( r_{ui}^* > \theta \) (In our case \( \theta = 0 \)). Since we defined our confidence function to be equal to 1 for all non-played games, \( C_u \) itself is not sparse (as opposed to the raw observations \( r \)). However, if we multiply \( C_u \) by \( p(u) \) before multiplying \( (Y^T Y + Y^T (C_u - I) Y + \lambda I)^{-1} Y^T \) with \( C_u p(u) \), we do not only optimize the multiplication of \( C_u p(u) \) by introducing a sparse vector multiplication, but also the multiplication by \( (Y^T Y + Y^T (C_u - I) Y + \lambda I)^{-1} Y^T \), since \( C_u p(u) \) has become sparse.
We gain another runtime reduction by considering the types of activity of the general and the personalized model. We note that both models consist of a dataset construction period and a computation period. During the dataset construction period the CPU is mostly idle and vice versa. We use this to our advantage by combining the dataset construction of the general model with the computation period of the personalized model, as depicted in Figure 24.

![Figure 24. Gain from parallel execution](image)

### 6.4.2. Dataset
The system will recommend games to the visitor based on historical data. We will construct a dataset of this historical data on which we train a model for the recommender system. The size of the dataset is open for discussion. A dataset based on too little data will not result in accurate predictions, since the user interests cannot be fully extracted from this data. Large datasets will result in extra computation time; not only during dataset construction, but primarily while computing the model. In our case a dataset of sixty days leads to satisfying results. Since a dataset of this time span is close to the computation time limit, we will take this as our dataset size.

Visitors may play the same game on different days. On our training data playtimes of observations are aggregated over all days of the dataset for each user. Therefore, extra observations of already contained visitor-game pairs do not lead to extra computations in the model. This makes the dataset size scale less than linear over the dataset time span, in the sense that doubling the amount of days in the training set causes a less than double increase in aggregated observations.

The recommender system has to finish overnight and therefore it is vital to limit the computations to the recommendations that likely changed most since the last computation. Concrete this means that we compute recommendations every night, but only for the users for which new data is available. In our situation that means that we compute recommendations for visitors that played on the last day.
Early in the morning we compute new recommendations for day $t$. We collect a list of all visitors that played on day $t - 1$ and retrieve all observations of these users over a period of the preset length of 60 days, up to day $t - 60$. Note that this greatly reduces the amount of observations. The total dataset of two months would contain approximately 16.4 million observations, where the reduced dataset only contains approximately 1.7 million observations.

To illustrate the size of the dataset we use the daily aggregated data between the time span starting at February 12th 2012 up to and including April 11th 2012. A dataset of this time span has the following properties.

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique visitors</td>
<td>120,952</td>
</tr>
<tr>
<td>Unique games</td>
<td>9,197</td>
</tr>
<tr>
<td>Observations</td>
<td>1,675,363</td>
</tr>
</tbody>
</table>

Table 9. Dataset properties (reduced)

However, if we would not reduce the users in our dataset to a single day and recompute the recommendations for all users in this time span, the dataset would have the following properties.

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique visitors</td>
<td>3,802,626</td>
</tr>
<tr>
<td>Unique games</td>
<td>9,320</td>
</tr>
<tr>
<td>Observations</td>
<td>16,386,917</td>
</tr>
</tbody>
</table>

Table 10. Dataset properties (before reduction)

This method has a small downside as it causes sub optimality in our recommendations. Visitors will only be recommended games that were present the last time they played. All new games added after that date could not have been included in the calculations for these users. At the rate of adding three new games every day, we can say that the effect on the recommendations is fairly limited. Taken into consideration that only a fraction of these games will be suited for the individual visitor and the computed list nonetheless contains suited games, we argue that it will only have a minimal effect on the quality of our recommendations.

If we look at the distribution of playtimes with respect to the heartbeat interval (section 3.2), we expect that the collaborative filtering model of the recommender system will perform better if we are able to estimate the playtime closer to the true value. The expected error will be lower if we modify the currently registered playtime to be equal to the average value of the playtimes inside a
heartbeat interval. This means we modify each registered playtime of 0 seconds to 24 seconds, each registered playtime of 180 seconds to 220 seconds, etc. See Appendix G for the full list.

Note that we do not apply this technique to the exploration-exploitation model, because it will lower the variance, which is an important factor in the exploration step of the model.

### 6.5. Integration Details

This section specifies in which format the recommender system will supply the recommendations to be included by the site builder module. The recommender system has been designed to work on the Funnygames.nl website. However, for the site builder module to function, recommendations have to be supplied for all Funnygames domains. Although the recommender system will not be evaluated outside the Funnygames.nl domain, it will generate recommendations for these sites nonetheless.

In the current situation site specific settings are set by using parameters. The (semi)static settings are stored in a configuration file. The current program needs the following parameters.

<table>
<thead>
<tr>
<th>Position</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Path</td>
<td>Path to output recommendations. Will be created if it does not exist and will overwrite existing files.</td>
</tr>
<tr>
<td>2</td>
<td>N</td>
<td>Amount of recommendation slots (set to 4)</td>
</tr>
<tr>
<td>3</td>
<td>Site</td>
<td>Site identifier (use 101 for Funnygames.nl)</td>
</tr>
<tr>
<td>4</td>
<td>Date</td>
<td>[Optional] Date [YYYYMMDD format] for which the system acts as if it were that date. If omitted Date is assumed to be today.</td>
</tr>
</tbody>
</table>

Table 11. Parameters

A proper call of the recommender system for Funnygames.nl would be

```bash
recsys.jar ./rec/101 4 101
```

It is possible to execute the recommender system as if it were an historical date, e.g. when there was a database problem at the usual automated runtime one may be interested in recomputing the recommendations once the system has recovered. Note that this recomputation should be done in a conservative way with respect to recommendations generated between the historical date and now to prevent an extra introduction of new-item exclusion and loss of observation data for those visitors. This conservation can be achieved by either recomputing recommendations for all the days between the historical date and now or (preferably) by simply overwriting regenerated recommendations with the newer files.

```bash
recsys.jar ./rec/101 4 101 20120213
```

The recommender system has to be executed once for each specified site (it does not detect the sites by itself). Paths and tables will be created in case they do not exist. Both models will write their result to XML files. Filenames of personalized recommendations are [uuid].xml, where uuid is the uuid as used in the database. Filenames of the generalized model are cat[#].xml, where ## represents the category number. Catgeneric.xml contains overall recommendations in case a category and its ancestors do not contain enough games for the amount of recommendations. This generic recommendations can also be used in contextless environments such as the “my games” page. The format of the XML files is defined in the following recom.xsd.
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- define single recommendation data -->
<xs:complexType name="rectype">
  <xs:attribute name="rank" type="xs:integer" />
  <xs:attribute name="type" type="xs:string" />
  <xs:attribute name="score" type="xs:decimal" />
  <xs:attribute name="gameid" type="xs:integer" use="required" />
</xs:complexType>

<!-- define list of recommendations -->
<xs:element name="recomlist">
  <xs:complexType>
    <xs:attribute name="target" type="xs:string" />
    <xs:sequence>
      <xs:element name="recommendation" type="rectype" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

Figure 26. XML Schema

An example xml file 0001ecbb-41e2-4c8a-96bb-b977ed854a2a.xml is given below.

<?xml version="1.0" encoding="ISO-8859-1" ?>
<recomlist target="0001ecbb-41e2-4c8a-96bb-b977ed854a2a"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="recom.xsd">
<recommendation rank="0" type="personal" score="0.999675371097893" gameid="12178" />
<recommendation rank="1" type="personal" score="0.978850395531043" gameid="1895" />
<recommendation rank="2" type="personal" score="0.925780613328597" gameid="121" />
<recommendation rank="3" type="personal" score="0.892558651302574" gameid="4228" />
</recomlist>

Figure 27. XML file containing personalized recommendations

The type of the recommendation is irrelevant for the actual behavior of the recommender system, but rather an indication of the basis of the recommendation. Currently type can consist of

- Personal, for personal recommendations
- Explore, for categorical recommendations, explore part
- Exploit, for categorical recommendations, exploit part
- Exploit-parent, for categorical recommendations of which the subcategory has less games than the amount of requested recommendations
- Exploit-Generic for contextless recommendations or when there are insufficient exploit, explore and exploit-parents in the category.
The configuration file can be used to set several (semi-)static properties. The recommender system reads this file by line. An example configuration file is written below.

```
#statistics database server
statsdb.tibaco.net
#statistics database
patrick
#statistics database user
patrick
#statistics database password
Passwd
#game-category database server
datawarehouse.tibaco.net
#game-category database user
patrick
#game-category database password
Passwd2
#game-category database
cms
#game-category table
Game
#personalized model: amount of factors (f)
25
#personalized model: overfitting constant (lambda)
900
#personalized model: confidence scaling parameter (alpha)
0.1
#personalized model: preference threshold (theta)
0
#personalized model: fold (set to -1 for operating mode)
-1
#categorical model: Exploration fraction (epsilon)
0.25
#categorical model: gameplay threshold
1000
#number of CPU cores to use for personalized model
4
#number of historical days general model
28
#number of historical days for uuid set for personalized model
1
#number of historical days for observations
59
```

**Figure 28. Configuration file example**

### 6.6. Evaluation of the personalized model

We shall evaluate both on historical data and empirically in practice. First we will discuss the theoretical evaluation that focuses on how well our model fits the visitor’s observed preferences and its potential to recommend games to visitors. After that we make an experimental setup to evaluate the model in practice.
Our approach on the theoretical evaluation is suggested by Hu, Koren & Volinsky [8]. We do not have feedback on which games a visitor dislikes, as not playing a game can have multiple reasons and is not necessarily caused by disliking a game. However, a visitor does express a kind of preference by playing a game, so we can use a recall oriented approach.

Let $\text{rank}_{u,i}$ be the predicted percentile ranking of game $i$ for visitor $u$ and $r^t_{u,i}$ the observation of that combination in the test set. Hu, Koren and Volinsky [8] propose to use the following evaluation metric:

$$\overline{\text{rank}} = \frac{\sum_u r^t_{u,i} \text{rank}_{u,i}}{\sum_u r^t_{u,i}}$$

We use this metric to determine the amount of factors needed to sufficiently describe the visitors and games. We expect $\overline{\text{rank}}$ to decrease for an increasing amount of factors, because the model will be able to describe visitors and games in more detail. For a sufficiently large amount of factors we expect $\overline{\text{rank}}$ to converge to its best performance.

We compare the performance metric of the collaborative filtering approach with a popularity performance metric. To calculate this popularity performance metric, we order the games by their popularity score, the exponential moving average of the overall gameplays of a game. The weighted percentile ranking of the popularity measure $\overline{\text{rank}}_{\text{pop}}$ equals 0.035808. This means that the popularity metric is a very powerful way to predict which game a visitor will play. This is influenced by the fact that the popularity measure is correlated to the decision which games are displayed on the index and therefore clearly visible to the visitor.

The above graph shows $\overline{\text{rank}}$ for different settings for the number of factors and regularization constant $\lambda$. We use the curves in this graph to optimize these settings. A decreasing line segment expresses underfitting which should be compensated by increasing the amount of factors. An increasing line segment represents overfitting, upon which we increase the regularization constant $\lambda$. From this graph we conclude that an amount of 25 factors and $\lambda=900$ performs well on our data. Increasing the amount of factors still improves the performance of the recommender system, but cannot be considered effective usage of computing time, since the $\overline{\text{rank}}$ only decreases slowly.
Depending on the situation, we may be interested in the performance only on games that the visitor has never played before. For this graph we define a similar base line as before, based upon the rank according to the popularity ranking of previously unplayed games. If we apply the same parameter estimation technique as before, we see that satisfying results can be achieved by using 25 factors with $\lambda=900$, just like the recommendations that include games the visitor previously played.

![Collaborative vs Popularity ranking](image)

**Figure 30. rank for different factors and $\lambda$ settings.**

One of the advantages of this approach is that it is an evaluation strategy that omits the easy estimation of preferences of previously played games. The disadvantage is that it will not remind the visitor of forgotten preferred games. Tibaco prefers to include previously played games in the recommendations. In both cases the collaborative filtering method fits the behavior of the visitor better in comparison to the overall most popular games.

### 6.6.1. Top-N evaluation

During the design of the recommender system it was not yet known where the recommendations would be placed and how many slots would be available. After we evaluated the recommender system for different settings, a decision was made to place the recommendations on a game page, which implies that we have only four slots for our recommendations. This knowledge gives rise to a new kind of evaluation. We can now evaluate on our top-4 recommendations. For this purpose we introduce a function which shows the fraction of observations in the test set for which the game is in the top-N of the recommendations. If we look at the top-1% recommendations, we see that the collaborative filtering performs superior to the popularity ranking.
The above graph gives insight in the quality of the top-1% recommendations. On a scale of around fifteen thousand games, the one percent equals around 150 games. What we are particularly interested in, is seeing how our model performs on the top-4 recommendations, as these will be the ones offered to the visitor. We also include a top-16 comparison, because the index pages consist of lists of sixteen games and perhaps in the future Tibaco might be interested in shifting the recommendations to the index. The graphs below show the top-4 and top-16 performance. The exact numbers are included in Appendix H.

The collaborative filtering model leads to better results on both the top-4 (fraction 0.129% vs 0.077%) and the top-16 (fraction 0.259% vs 0.165%) recommendations. See Appendix G for the exact numbers.
Recall research question Q5:

Q5: How can we recommend games to returning visitors, based on their gameplay history?

In this section we have described an approach based on collaborative filtering that outperforms the current overall popularity measure in a top-4 and top-16 evaluation. We can therefore conclude that we have successfully implemented a recommender system that recommends games to returning visitors based on their gameplay history. In the remainder of this section we sketch an approach for the empirical evaluation by using an ABCD-test. This test has not been executed yet, so we cannot conclude on actual better performance in practice.

### 6.6.2. Empirical Evaluation

The third kind of evaluation will take place on the live website. This way we will be able to compare the performance of our system to the old situation under equal environmental circumstances for both the situation with and without a recommender system. The situation without a recommender system will show a list of the most popular games inside the category.

We are not only interested in changes between the new and old situation, but we further distinguish between partial integrations to measure the influence of the developed models separately. We divide visitors into the following groups. Using this experiment setup we have a double option for observing the effect of the individual models, e.g. comparing B to C and A to D both indicate the influence of the personalized model.

<table>
<thead>
<tr>
<th>First time visitors</th>
<th>Returning visitors</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Current</td>
</tr>
<tr>
<td>B</td>
<td>General Model</td>
</tr>
<tr>
<td>C</td>
<td>General Model</td>
</tr>
<tr>
<td>D</td>
<td>Current</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Personalized Model</th>
</tr>
</thead>
</table>

Table 12. ABCD test

In addition to the currently available metrics, we are interested in distinguishing clicks on games in the recommendation slots. The metrics we are interested in are the following:

- Average playtime of games on a page visited through a recommended slot link.
- The click-through-rate on a game in a recommendation slot (CTR).
- Page views per user (PPU)

At the time of writing this report, the empirical evaluation has not started to run. Results can therefore not be included in this report.
7. Conclusion and Future Work

In the first chapter we have identified five research questions. In this final chapter we will look back to these research questions and conclude whether or not we have answered these questions in a satisfactory manner.

Q1: Based on partial daily data, can we estimate the popularity of a game?

We have set up a model that can estimate the daily gameplays based on data of only a part of the day. The accuracy depends on the size of the period on which the estimation is based, but we conclude that the estimation is sufficiently accurate for the application it is designed for, i.e. the estimation based on periods related to the publishing times of games. The accuracy of the given method may be further improved by studying the constitution of the large group of people of which we have no information on. Another subject of future work are the boundaries of the current target groups. These boundaries are based on the quantity of visitors that belong to the group, rather than their joint characteristics w.r.t. playing times. Redefining the target groups’ boundaries may improve the estimation of each separate target group and therefore also improve the total estimation.

Q2: Can we use the estimation of Q1 to improve the constitution of the index lists?

We have given an approach on how this can be done. In the current situation however, we have too few data samples to be conclusive on this question. Once more data is available the given approach can be used to improve the transition of a game from the new list to the categorical popularity list.

Q3: Can we use game statistics to detect broken games?

Using a cost-sensitive decision tree classifier on the game statistics we achieve a precision of 61% and recall of 40%. We are satisfied with these metrics, especially since the false positives are often games of less quality, in the sense that it concerns foreign languages or bad controls which degrades overall visitor experience.

Q4: How can we recommend games to visitors that have no gameplay history?

We have taken an approach based on an exploration-exploitation scheme. Central in this approach are the playtimes of a game, in particular the average playtime and the variance in the playtimes. The exploitation part takes the N most popular games in a balanced policy consisting of both playtimes and amount of gameplays. The exploration is based on an uncertainty measure with respect to the popularity of the current games in the exploitation. We have not been able to evaluate our approach, but have set up an experiment based on empirical analysis that might show a difference in gameplay or playtime statistics in practice.

We make a note on the currently used exploration measure. Currently we do not prioritize games with an average playtime above the threshold over games with an average playtime below the threshold. This choice is arbitrary and we may also be interested in determining the effect of prioritizing games above this threshold over games below the threshold.

The recommender system context was unknown for a long time. During the process Tibaco decided to insert the recommendations on a game page, which yields a game context: the game that is currently being played. An alternative approach to recommending games to visitors without
gameplay history is to determine item similarity in the classical item based collaborative filtering approach (See Chapter 2). Using this item similarity we can then recommended the games that are most similar to the game the user is currently playing, since the user has expressed a preference by playing the current game.

Q5: How can we recommend games to returning visitors, based on their gameplay history?

We have used a collaborative filtering technique for implicit feedback datasets to recommend games to returning visitors. By using the results of a top-4 and top-16 evaluation we concluded that our recommendations are expected to outperform the ranking based on the current popularity measure. A scheduled empirical test has not yet been executed, so we are inconclusive on the actual performance in practice.

Although we are satisfied by the result of the recommender system, there are several interesting aspects that may lead to improvements. We start off by the confidence measure we use for our collaborative filtering as given in Equation 20. We have used a confidence measure that linearly relates to the playtime. Using a different confidence measure might yield better results. As a side note we mention that we have also experimented with the sine function given below to better represent the intuitive notion of confidence for very low or very high playtimes, but it did not lead to better results.

\[ c_{ui} = 1 + \frac{\alpha}{2} \left( 1 + \sin \left( \frac{r_{ui} \pi}{3268} - 0.5\pi \right) \right) \]

![Figure 34. Alternative confidence function](image)

Any click oriented measure (including profit) focuses on the click attractiveness of the game and not the actual visitor satisfaction. Any playtime oriented measure is influenced by the heartbeat interval when determining the final page playtime. Since we expect visitors to be more satisfied by the recommendations we expect them to visit fewer pages per time unit and therefore (if people have limited time available) have less non-final pages, resulting in more final-pages per page view. This means that relatively more of the page views are rounded downwards to the closest heartbeat interval and the registered average playtime may appear lower than in the current situation, while the actual average playtime is in fact higher. This might lead to unexpected results in the empirical evaluation.

As a conclusive note we mention that our recommender system is designed for optimizing the visitor’s satisfaction while the company’s objectives are primarily based on financial profit. The ultimate visitor satisfaction is achieved when the recommender system suggests a game that will keep the visitor on the website forever playing this one game. This will result in very low page views and therefore very low financial profit. In fact, if we want to optimize financial profit we should keep the visitor unhappy to some extent to stimulate the visitor to try another game instead.
Bibliography


Appendix A. Popularity Lists on Index

Top level categories:

- Adventure
- Girls
- Skill
- Brain
- Racing
- Sports
- Fighting
- Casino

Attribute categories; besides their main category from the list above, games may also belong to one of the attribute categories below. Note that only a small portion of the games have this property.

- Highscore
- Multiplayer

Other lists on index

- Popular, contains the most popular games over all top level categories
- New, contains new games; disregards the popularity measure.
Appendix B. Target groups

<table>
<thead>
<tr>
<th>Group</th>
<th>Gender</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>M0</td>
<td>Male</td>
<td>0-5</td>
</tr>
<tr>
<td>M1</td>
<td>Male</td>
<td>6-9</td>
</tr>
<tr>
<td>M2</td>
<td>Male</td>
<td>10-13</td>
</tr>
<tr>
<td>M3</td>
<td>Male</td>
<td>14-17</td>
</tr>
<tr>
<td>M4</td>
<td>Male</td>
<td>18-24</td>
</tr>
<tr>
<td>M5</td>
<td>Male</td>
<td>25-40</td>
</tr>
<tr>
<td>M6</td>
<td>Male</td>
<td>41-50</td>
</tr>
<tr>
<td>M7</td>
<td>Male</td>
<td>51-90</td>
</tr>
<tr>
<td>M8</td>
<td>Male</td>
<td>90-150</td>
</tr>
<tr>
<td>V0</td>
<td>Female</td>
<td>0-5</td>
</tr>
<tr>
<td>V1</td>
<td>Female</td>
<td>6-9</td>
</tr>
<tr>
<td>V2</td>
<td>Female</td>
<td>10-13</td>
</tr>
<tr>
<td>V3</td>
<td>Female</td>
<td>14-17</td>
</tr>
<tr>
<td>V4</td>
<td>Female</td>
<td>18-24</td>
</tr>
<tr>
<td>V5</td>
<td>Female</td>
<td>25-40</td>
</tr>
<tr>
<td>V6</td>
<td>Female</td>
<td>41-50</td>
</tr>
<tr>
<td>V7</td>
<td>Female</td>
<td>51-90</td>
</tr>
<tr>
<td>V8</td>
<td>Female</td>
<td>90-150</td>
</tr>
</tbody>
</table>
Appendix C. Preferred model threshold
Appendix E. Cost Sensitive Model
Appendix F. Used Queries

Replace YYYY by the current year, MM_DD by the current month and day (DB format) and by h by the current heartbeat interval.

Retrieving stats related to playtimes and gameplays:

```sql
SELECT counter, sum(count) as count, sum(seconds)  as secondsTot
FROM statistics_YYYY_101.MM_DD
WHERE uuid != 'global' AND counter LIKE 'Game%' AND seconds % h > 0
GROUP BY counter
```

Retrieving Reports:

```sql
SELECT counter, sum(count) as count FROM statistics_YYYY_101.MM_DD
WHERE uuid = 'global' AND counter LIKE 'reportBroken%'
GROUP BY counter
```
### Appendix G. Average playtimes in heartbeat intervals

<table>
<thead>
<tr>
<th>Interval</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-179</td>
<td>24</td>
</tr>
<tr>
<td>181-359</td>
<td>220</td>
</tr>
<tr>
<td>361-539</td>
<td>403</td>
</tr>
<tr>
<td>541-719</td>
<td>587</td>
</tr>
<tr>
<td>721-899</td>
<td>767</td>
</tr>
<tr>
<td>901-1079</td>
<td>948</td>
</tr>
<tr>
<td>1080-1259</td>
<td>1129</td>
</tr>
<tr>
<td>1261-1439</td>
<td>1308</td>
</tr>
<tr>
<td>1441-1619</td>
<td>1489</td>
</tr>
<tr>
<td>1621-1799</td>
<td>1670</td>
</tr>
<tr>
<td>1801-1979</td>
<td>1851</td>
</tr>
<tr>
<td>1981-2159</td>
<td>2032</td>
</tr>
<tr>
<td>2161-2339</td>
<td>2211</td>
</tr>
<tr>
<td>2341-2519</td>
<td>2389</td>
</tr>
<tr>
<td>2521-2699</td>
<td>2571</td>
</tr>
<tr>
<td>2701-2879</td>
<td>2748</td>
</tr>
<tr>
<td>2881-3059</td>
<td>2930</td>
</tr>
<tr>
<td>3061-3239</td>
<td>3113</td>
</tr>
<tr>
<td>3241-3286</td>
<td>3249</td>
</tr>
</tbody>
</table>
### Appendix H. Top-N Evaluation Details

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0,0%</td>
<td>0,030272205</td>
<td>0,034737629</td>
<td>0,005817824</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0,0%</td>
<td>0,044145321</td>
<td>0,074124658</td>
<td>0,011153509</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0,0%</td>
<td>0,057049886</td>
<td>0,109253686</td>
<td>0,016434091</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0,0%</td>
<td>0,077444721</td>
<td>0,128677342</td>
<td>0,021466716</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0,1%</td>
<td>0,090692522</td>
<td>0,144844151</td>
<td>0,026352403</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0,1%</td>
<td>0,097520019</td>
<td>0,160629863</td>
<td>0,03080646</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0,1%</td>
<td>0,108142722</td>
<td>0,173191688</td>
<td>0,035301843</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0,1%</td>
<td>0,113142722</td>
<td>0,185254996</td>
<td>0,039691614</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0,1%</td>
<td>0,12175669</td>
<td>0,196448987</td>
<td>0,044191589</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0,1%</td>
<td>0,132993751</td>
<td>0,206821042</td>
<td>0,048838501</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0,1%</td>
<td>0,142561267</td>
<td>0,21665544</td>
<td>0,052906846</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0,1%</td>
<td>0,14948279</td>
<td>0,226046939</td>
<td>0,057062435</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0,1%</td>
<td>0,154442498</td>
<td>0,234501142</td>
<td>0,0610068</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0,1%</td>
<td>0,160808016</td>
<td>0,242922386</td>
<td>0,065121063</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0,2%</td>
<td>0,162808016</td>
<td>0,25116029</td>
<td>0,068955225</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0,2%</td>
<td>0,16519643</td>
<td>0,258691617</td>
<td>0,072885815</td>
<td></td>
</tr>
</tbody>
</table>