Realtime recommendation system for online games

Tautvydas Misiunas

Master of Science
School of Informatics
University of Edinburgh
2014

Abstract

Online retailers are offering ever more items to choose from, but this makes it difficult for users to find the right product. Algorithm driven recommendation systems address this problem by helping to explore the catalogue more efficiently, while at the same time increasing the revenues for the shop. These systems are widely applied by online retailers and service providers, examples include: Amazon and Netflix. We extend this concept and focus on the online game in-game shops.

We tackle two major problems. The first one is cold start which is common for recommender systems. Cold start is present for new users or items that do not have any previous history hence making the predictions hard. However, we cannot rely on the usual solution of content-based filtering because our dataset is completely anonymous – without the content. The second issue is that we do not have item rating which are mandatory for vast majority of collaborative filtering methods. We engineer several rating alternatives from the in-game event stream.

For comparison purposes, we define two baselines that are both computationally fast and are fairly accurate. They give us an idea for most popular items for either all users or a single user.

We investigate the performance of two algorithm groups: collaborative filtering (CF) methods and logistic regression classifier. First, we apply memory and model based CF algorithms to a game dataset. We selected 'Slope One', user clustering and matrix factorization models because they are the best performing models in terms of speed and prediction accuracy for movie recommendation datasets (Lee et al., 2012). Our results demonstrate that 'Slope One' is not suitable for the rating-less dataset. Furthermore, our experiments show that matrix factorization models perform on the same level as the proposed baselines. Then, we show that simple user clustering exhibits best performance of all CF methods.

Next we consider the logistic regression for the purchase predictions. We use a richer feature set to show that this standard machine learning method can be successfully applied for the recommendation purposes. We then combine the CF and the logistic regression together by extending the feature set with the user clustering results. This made better predictions than either of the methods separately. Through experiments we find that the logistic regression combined with clustering results outperforms our baselines and also makes successful predictions for the new users – solving the cold start problem.

Acknowledgements

I would like to thank my supervisor, Iain Murray. His continued support and patience has helped me to stay motivated and focused. We also shared many long discussions where Iain game me many insights into my work and the field. I have learned many things thought this project and I appreciate the help. Thank you.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Tautvydas Misiunas*)

Table of Contents

1 Introduction **1**

1.1 Online games 2

1.2 Goals 2

1.3 Project outline 3

2 Background **5**

2.1 Literature review 6

2.1.1 Memory-based methods 6

2.1.2 Model-based methods 7

2.1.3 Logistic regression 8

3 Dataset **9**

3.1 User and item distributions 10

4 Models **13**

4.1 Baselines 13

4.2 Feature engineering 14

4.3 Slope One 14

4.4 User clustering 16

4.5 Matrix factorization 18

4.6 Logistic regression 19

5 Evaluation **22**

5.1 Evaluation setup 22

5.2 Evaluation metrics 23

5.3 Baselines 24

5.4 Slope One 25

5.5 User clustering 26

5.6 Matrix factorization 28
5.7 Logistic regression 30
5.7.1 Filtering items 30
5.7.2 Data split 31
5.7.3 Extra features 32
5.7.4 Conclusion 34

6 Conclusion **36**
6.1 Other remarks 37
6.2 Future work 38

A Event attributes and categories **39**

B Raw data filtering **41**

Bibliography **42**

Chapter 1

Introduction

A recommender recommends. From the consumer point-of-view that is all it does — suggests products, friends, movies or activities that may be interesting to the user. We can identify four main benefits of such system. Firstly, the systems learns customer's patterns and helps to find desired items *quicker*. Secondly, by suggesting new and trendy items it helps the customer to *explore* the larger product collection and also reminds users about forgotten items. Thirdly, the personalized item list *boosts user experience and retention rate*. Finally, listing items that are likely to be purchased first, increases the chance of purchase, and thus *increases revenues*.

Product are suggestions are wide spread in online retailing. From specialized companies like Wiggle[1] selling bicycles, to national grocery retailers like Ocado[2], to global leaders in retail and distribution like Amazon[3]. Although recommendations systems are wide spread, in many cases they are remarkably simple – just showing the popular products.

Online grocery shops are selling consumable product hence they tend to give a list of previously bought items by the same user. Such suggestions in fact are helpful when the customer does a weekly shopping to resupply stock. However, they do not help the user to explore the catalog. Another simple method to provide suggestions is to recommend the most popular items. They can be shown as overall or temporal (i.e. in the last month) popularity. While the popular item list exposes the user to new items it is poorly tailored to a specific customer as the same list is shown to everybody.

To address this some retailers are using sophisticated approaches alongside the previously mentioned methods. This allows to tailor suggestions to every user and

[1]www.wiggle.co.uk
[2]www.ocado.com
[3]www.amazon.com

maximise previously discussed goals. We will come back to these methods in chapter 2.

1.1 Online games

Internet user base is expanding at about 8% annually (International Telecommunications Union, 2013). The wide availability allowed spread of new online multiplayer games. Notably, a new type of games emerged – a massively multiplayer online free-to-play games. Examples include League of Legends[4] or Dota[5] with over 5 million and 1.5 million users respectively (Nair, 2009; Riot Games, 2013).

Such large user base is possible due to the free-to-play business model. Developers make their revenue by selling optional game items in their in-game shops. These are virtual items with almost no production costs and with wide selections to capture the wide range of players' tastes. They are luxury and advanced items that are somewhat optional to the game. Therefore, the user experience for the shop has to be well polished. Smart suggestions should increase the number of items purchased and also increase the user satisfaction.

Furthermore, there is an additional aspect to the in-game retail. Users can also get a pop-up like message with a special offer tailored for them. Since notifications have limited space on the screen the number of items in it is also very limited. It is a considerable constraint. In later chapter we provide an estimate for optimal list length that results in most successful purchases.

1.2 Goals

The dataset was provided by deltaDNA company and their main request is to develop a recommender system. The system has to be flexible enough to work in any game and not to rely on the game specifics. As the in-game recommendation is still rather unexplored field we identified three goals.

First of all, a typical collaborative filtering (CF) technique infers item rating that the user would give. The method uses previously rated items to make the predictions. However, our dataset does not have ratings. The data comes as a stream of various in-game events. Therefore, we *engineer the item ratings* from the purchase history which

[4]www.leagueoflegends.com
[5]www.dota2.com

is assembled from the event stream. Afterwards, CF algorithms are modeled with the artificial ratings. Additionally, we explore a standard machine learning method, logistic regression, which does not require ratings matrix.

Secondly, a free-to-play nature of the games attracts more players. However, they are less committed to the game and are more likely to switch to other games after just a trial play. As a result the dataset has a higher number of beginner players with a limited or no history of previous purchases. Predicting for such users is a challenging problem known as **cold start** because the CF algorithms rely on identifying like-minded players. Typically, this problem is approached by mixing in content-based filtering. However, in order to protect sensitive user information, our dataset was anonymised and does not contain any descriptions. Our goal is to identify which methods work well with the newcomers.

Lastly, even though the recommendation algorithms are widely applied to movie[6,7], music[8], books[9], food[10], wine[11] and dating[12] databases, we failed to find any public application to the video games. Therefore, our third goal is to *explore and adapt recommendation models* from other fields. We will apply best performing models from different methodology categories to the game dataset.

1.3 Project outline

The structure of the thesis is divided into chapters in the following way:

Chapter 2 In the next chapter we introduce the different approaches to the user demand prediction. We also overview the current state-of-the-art recommender models of the recommender systems. Some of the background overview is shared with the research proposal paper for this project.

Chapter 3 The provided real-life dataset is described in terms of size and content. We also discuss the data preparation and introduce the created dataset which is used in the remaining work.

[6]MovieLens: http://grouplens.org/datasets/movielens/
[7]Netflix: http://www.netflixprize.com
[8]LastFM: http://www.dtic.upf.edu/ ocelma/MusicRecommendationDataset/index.html
[9]Institut fur Informatik: http://www2.informatik.uni-freiburg.de/ cziegler/BX/
[10]Chicago Entree: http://archive.ics.uci.edu/ml/datasets/Entree+Chicago+Recommendation+Data
[11]CellarTracker: https://snap.stanford.edu/data/web-CellarTracker.html
[12]http://www.occamslab.com/petricek/data/

Chapter 4 The Models chapter introduces the selected recommendation models giving the details of how they are modified to suit our specific dataset.

Chapter 5 Model evaluation is provided along the comparison and discussion on performance.

Chapter 6 In the final chapter we summarize achievements of this work.

Chapter 2

Background

The recommendations are usually based on either preferences of similar users using *collaborative filtering* (CF) or similar items derived by *content-based filtering*. The content based models make use of the product characteristics, such as the author and genre of the song or word frequencies in a book. The method is similar to search query routines and is sometimes referred to as a search-based approach (Linden et al., 2003). The items, that are defined by frequency-type features, are typically matched using a similarity function such as cosine similarity or term frequency – inverse document frequency (TF-IDF) (Adomavicius and Tuzhilin, 2005).

The other approach makes the recommendations (filters) data for a specific user by analyzing preferences of many other users (collaboration). Unlike content-based approaches, CF recommendations are not limited in scope by simple item similarity. It can make suggestions on completely different items if other users prefer those item combinations.

Collaborative filtering does not require any prior information about the items and their similarity. Therefore, the proposal will focus on collaborative filtering rather than a content-based approach or a mixture of them. Collaborative filtering can be further split into two types: **memory-based** and **model-based** (Adomavicius and Tuzhilin, 2005; Lee et al., 2012). The memory-based methods aim to find similarities between users or items. The system can similar users (neighbors) and recommend their preferred items which is known as nearest-neighbor or user-based CF. It can also try to identify items that are related to the currently bought ones. In both cases the trained data is stored in a matrix.

On the other hand, model-based methods train a model on the data which is used to make future recommendations. Typically they are faster at query time and take less space for representation, however might take longer to train and update (Lemire and Maclachlan, 2005).

2.1 Literature review

Lee, Sun and Lebanon performed a comparative study of collaborative filtering algorithms (Lee et al., 2012) including memory and model based approaches. They evaluated 12 different methods as well as three baselines on the Netflix competition dataset. On top of that, they have re-sampled the dataset in order to measure model performance dependency on item and user counts as well as user-item matrix sparsity. Moreover, the study provides the time required to train and test each algorithm.

The authors conclude that non-negative matrix factorization (NMF) and regularised SVD (reg-SVD) perform best on sparse datasets. However, reg-SVD requires significantly more time. For a time constrained setup, the study shows that the Slope-one method (Lemire and Maclachlan, 2005) out performs NMF on denser matrices.

2.1.1 Memory-based methods

A user-based method such as nearest neighbor finds a close neighbor based on the purchase rating or history. However, with a big item range and especially new users the system may be unable to find similar customers. In other words, sparse user-item matrix decreases the performance of the recommender (Sarwar et al., 2001). Additionally, once queried, the user has to be compared to all other users. Each comparison involves measuring the distance between vectors that represent the users, which are the ratings for all the items. Therefore, the prediction time of this approach scales linearly with the number of users and items which leads to a slow query time performance.

Finding item to item similarities and using them for suggestions has higher performance then user-based methods (Sarwar et al., 2001). Moreover, these similarities are relatively static and can be pre-computed offline in $O(\text{users} \times \text{items})$ time (Linden et al., 2003). Query time performance only depends on the number of items the user has bought before which is only a fraction of a large item set (Sarwar et al., 2001). In conclusion, the item-based methods provide high query throughput thus we will only apply this method from memory-based category. Furthermore, the item to item do not require the item ratings as they are generated purely from the user sets that bought the items.

2.1.2 Model-based methods

Slope one (Lemire and Maclachlan, 2005) is a simple and intuitive predictor. It is based on pairwise rating difference between items. The model estimates unknown item ratings in the form of the slope $f(x) = x + b$, where x is the rating for a purchased item and b is a constant. The constant is determined by taking an average over all users who bought the two items.

The Slope one model shows that a simple, easily interpretable, efficient at query time and updatable on the fly method at the same time has competitive performance. The model has similar MAE rates to memory-based techniques on two movie rating databases: EachMovie[1] and Movielens[2] (Lemire and Maclachlan, 2005).

A new approach to model evolution of users' preferences and tastes explicitly was suggested by McAuley and Leskovec (McAuley and Leskovec, 2013). In their paper they argue that a novice user's tastes differ from an expert with a refined taste. Therefore, they model different user levels with separately trained recommender systems. Each user level change is modeled individually and may have varying level intervals which allows for some users to start as experts while others may stay beginners for a long periods of time.

This model is applied on top of a standard recommender system which can be any of the model-based methods. In the report authors report a significant improvement in mean squared error (MSE) over five dataset tested. The highest performance boost is on Amazon movie dataset with 23% error decrease.

The experience modeling model not only improves prediction accuracy, but also groups users into segments and knows when they level up. Moreover, even though this approach splits the data into as many subsets as there are user segments, it does not lower the user-item matrix density or severs the cold start problem.

Another set of recommender systems are based on matrix factorization. Regularized singular value decomposition (reg SVD) splits the user-item matrix into a product of two separate matrices: user and item profiles. These profile matrices have lower dimensionality then the original representation. Alternatively, non-negative matrix factorization (NMF) splits the matrix with the constraint that elements have to be non-negative. Other matrix factorization method were suggested, however reg SVD and NMF performed best on the large and sparse datasets in terms of accuracy and time (Lee et al., 2012).

[1]http://grouplens.org/datasets/eachmovie/

[2]http://grouplens.org/datasets/movielens/

2.1.3 Logistic regression

Logistic regression is a well known textbook model (Bishop et al., 2006) with wide range of applications. However, it is hardly used for the ranking and recommending purposes.

Logistic regression has been applied to news recommendation (Lau, 2011) which can be restated as news filtering. There are two properties specific to the news filtering. First of all, the documents have content which describes the item itself. As seen in the section 3, our items are content-less. However, to some extent we know the environment and the state of the user when the purchase was made.

Secondly, filtering task works with unlimited set of documents and provides binary classification whether the article is interesting or not.

Furthermore, logistic regression has been applied to predict online purchases based on the clickstream of the user (Van den Poel and Buckinx, 2005).

Chapter 3

Dataset

The company deltaDNA (formerly known as Games Analytics) provided recorded data of a real game. The dataset was fully anonymised to protect sensitive user information.

The dataset is a record of 40 million in-game events covering 44 days (just over six weeks), starting on the first of January 2013. Throughout the monitoring period no recommendation system was in action. As a result, any bias towards an item will be present for all users. However, at the beginning of the recording period a 50% discount was offered for a day. Since the discount was applied to all items, the offer increased the net purchases while the sold item proportions stayed the same. Therefore, we kept the transactions in the dataset.

Every event has 76 attributes, such as user id, however only a fraction of the attributes are related to a specific event category. Furthermore, to allow flexible events every event has a main entry and subentries which allows to have unbounded event, for example involving multiple item trades. System has fifteen event categories ranging from notification of a started mission to an invitation from other player to form a party. Since we are interested in predicting what the user will buy next we will focus on the transaction category. The full lists of event attributes and categories are given in the appendix A.

Transaction category events can be further split into three groups – sale, trade and purchase. Sale and purchase events represent user interaction with the in-game shop using either virtual currency or real money. The trade events, on the other hand, are records of user to user item and money exchange. The dataset contains over 5 million transactions out of which 407 thousands are of purchase type.

The purchases from the shop were made by 40220 users and involve 982 distinct items. However, both user and item distributions have long tail with many items only being bought once and even more users only buying a single item.

The 40 million of event were filtered to contain only the informative transaction events using Unix command which are given in the appendix B. Afterwards, the data file was further trimmed by removing unused attribute columns.

3.1 User and item distributions

Firstly, to give a general understanding of how actively users buy items and which items get bought provide standard metrics in the table bellow:

	User	Item
unique	40143	982
mean	10.2	415.4
median	4	30
standard dev.	19.1	2049.5
max	1347	55748

Table 3.1: Basic metrics of the user and item transaction distributions. In total the dataset contains 40143 transactions.

From the user distribution (fig. 3.1) we observe that only 25% of the users have made more than 10 transactions. Furthermore, these users made over 70% of all transactions as seen in figure 3.2a.

In figure 3.2 we show the cumulative transaction distributions over the users and items. We can see that the transactions are dominated by the most popular 200 item that make up for more than 90% of the data.

To visualize the user retention rate, which is the number of returning users, we defined the overall and effective user lifetime. The overall lifetime is computed as a difference between the first and last purchases. On the other hand, the effective lifetime is the number of days a user was active — bought an item. Clearly the effective lifetimes will be either shorter or of the same length as the overall ones.

The lifetimes, generated from the `purchase` dataset, are shown in the fig. 3.3. It can be seen that both distribution have an exponential decay form. However, the overall lifetime has a wider tail indicating that the game has loyal players that are active throughout the monitored time period but not every day.

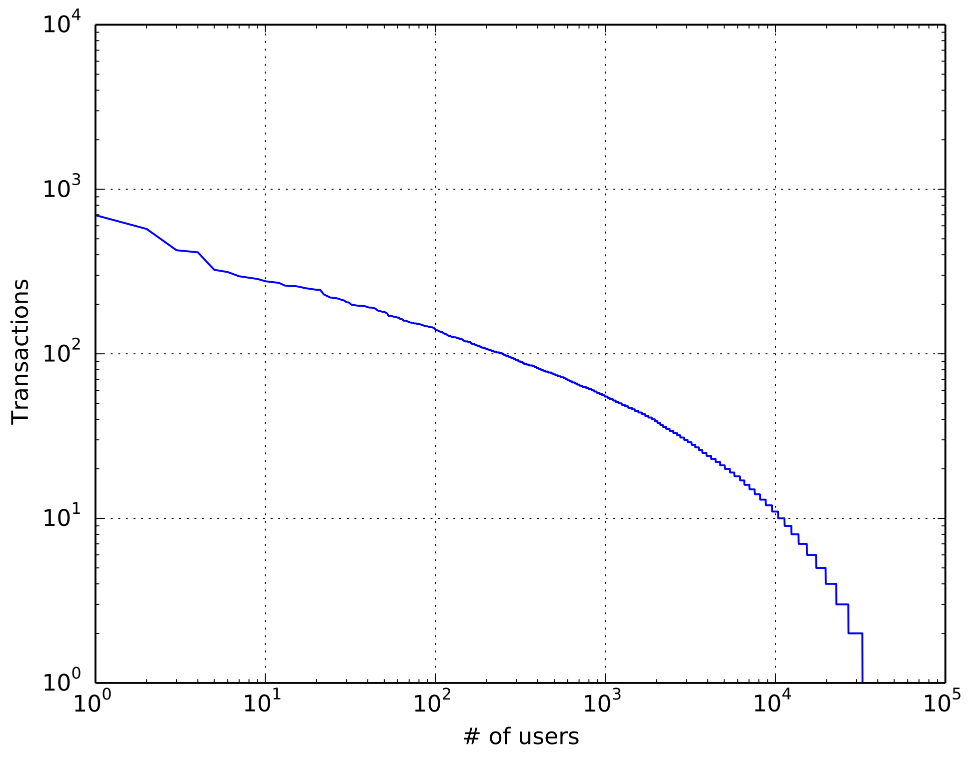
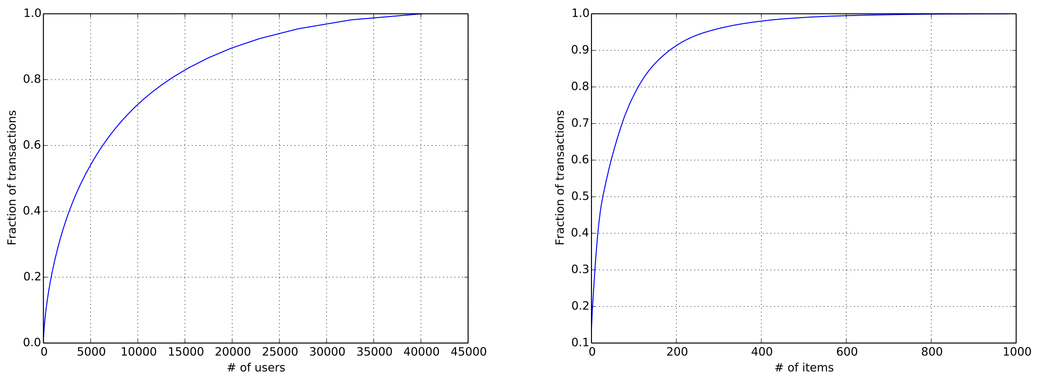


Figure 3.1: User distribution on a $\log$-$\log$ scale.



(a) User.

(b) Item.

Figure 3.2: Cumulative transaction distributions over users (fig. 3.2a) and items (fig. 3.2b). Both items and users were sorted by the popularity.

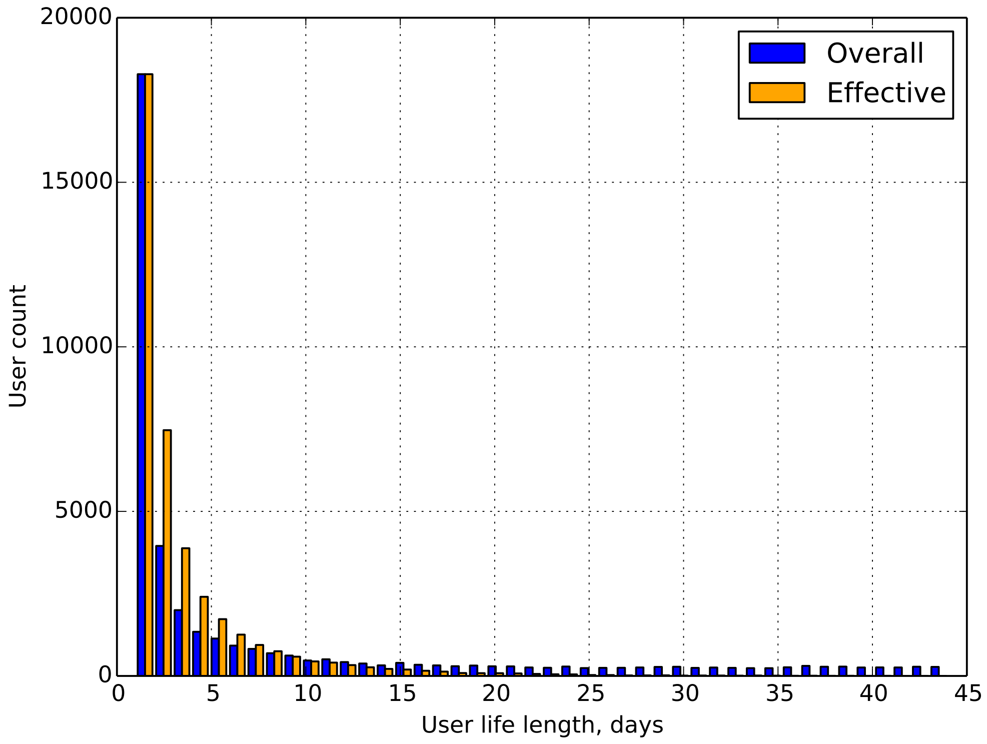


Figure 3.3: User lifetime. The overall lifetime is the period between the first and last activity whereas the effective lifetime is the number of days user was active.

Chapter 4

Models

This chapter introduces the selected recommendation models together with the baselines. The data analysis and the models were implemented using pandas (McKinney, 2010) as well as numpy and scipy (Jones et al., 01) Python libraries. Some of the more complex algorithm implementations were used from the sklearn library (Pedregosa et al., 2011).

The models are presented in the same order as they were implemented and evaluated. We start by defining our two baselines. Followed by the description of engineered features with were used as replacement to the item ratings. Afterwards, move on to the memory-based and model-based collaborative filtering algorithms. Finally, we finish with the application of multiclass logistic classifier. None of the content-based (CB) filtering methods were considered because the dataset is anonymised. Therefore, we do not have item or user related content which is critical in CB filtering.

The models were selected to cover many different techniques in limited time. As a result, we only consider one or two methods from each group.

4.1 Baselines

In order to evaluate trialed methods we will compare their results to the two baselines – a global popularity and a user average.

Global popularity, called H_0, is computed by counting how many times each item has been purchased in the training dataset and ranking them based on the counts. At the evaluation the same item suggestions are issues to every user regardless of any features.

The second baseline, user average, named H_1, is extended by computing the item popularity for every user. In other words, baseline suggest items that the user has frequently bought before. Unlike the global popularity, the user average discards all the data from the other users.

4.2 Feature engineering

Raw item counts are not good enough as we want new user with only few purchases and old user with a long history to have similar ratings if their purchase preferences match. Consequently, we explore a few transformation methods.

Firstly, we tried scaling the feature vectors to a unit length. This scaling process will put all the users on a unit length hyper-sphere and should bring closer users with similar item preferences but different quantities. As the item counts are all positive so are the scaled features. Therefore, we are using only 2^{-982} of the hyper-spheres surface where all numbers are non negative.

Another scaling approach was based on a hypothesis that a change in smaller item count is more important that in high counts. Therefore, we used logarithmic scale for the features bringing high counts closer.

User can also be defined by the unique item purchase profile. For this purpose features were converted to binary format – whether the user has ever bought the item or not. This transformation discards all the information about the repetitive purchases.

Features expressed as frequencies or counts are often encountered in information retrieval and text based information filtering. All documents and queries are treated as bags of words containing counts of present words or tokens. Term frequency-inverted document frequency (**tf-idf**)(Croft et al., 2010) is used in order to scale down the impact of tokens or items that appear frequently throughout the dataset and keep the low counts that are distinctive for particular user groups.

4.3 Slope One

The Slope One algorithm (Lemire and Maclachlan, 2005) was designed to capture global item similarities from the item scores which are numbers from a bounded scale. For example a movie dataset where each rating is from one to five and is given after the user watches the movie a single time. We will test how well Slope One performs when the item scores are expressed as purchase frequencies. Coming back to movie data set example, the new scores would be the number of times the user watched the movie. The more times it has been watched higher the score and will be recommended to be watched again.

The Slope One method encapsulates item relationships as linear dependencies and then uses them to predict scores for item previously unseen by user. For every item pair of x and y the score difference is modeled with a single number $\delta = y - x$. This is simplified version of linear equation $y = ax + \delta$.

The item suggestions are generated in two steps. Firstly, item similarities are calculated using all purchase information. Secondly, ranked list of item is produced by integrating item score similarities together with a set of items that the user has previously bought.

A simple example of user–item feature matrix is shown in the table 4.1. The only missing score is for the item J from the user B. Firstly, Slope One algorithm computes the difference between the items I and J using the user's A ratings: $\delta = 5 - 2$. Afterwards, the rating difference δ is added to the already rated item I of user B to predict rating for another item and the same user.

	Item I	Item J
User A	2	5
User B	1	$? = 1 + \overbrace{(5-2)}^{\delta}$

Table 4.1: A simple example of Slope One algorithm.

The item similarities are computed for every item pair resulting in $\binom{n}{2}$ numbers (where n is the number of items). Our main `purchase` dataset contains 982 items resulting in $\binom{982}{2} = 481,671$ item similarities.

On a bigger dataset the item pair similarity (δ) is averaged over all users (U) that bought both items:

$$\delta_{y,x} = \frac{1}{|U|} \sum_{u \in U} u_y - u_x \tag{4.1}$$

Similarly, the final prediction score is an average of all estimates for that item:

$$y = \frac{1}{|X|} \sum_{x \in X} x + \delta_{y,x} \tag{4.2}$$

where X is the set of ratings of currently bought items by the queried user. Even though the rating differences δ are computed using only the training data, the set X contains all the rating available at query time – from both training and testing sets. The second step is repeated for all unseen (without rating) items.

Once all predictions for items are generated, all item rating are sorted to make a recommendation list. Therefore, user's frequently bought items will appear high together with items that have high prediction scores.

The suggestion list contains all available items as long as the rating difference matrix Δ is dense enough and the user has at enough purchases. However, if the user has bought only a single item and that item happens to be a very unpopular then the recommendation list will be much shorter. In the extreme case a completely new user would get an empty list.

In order to make the query time execution faster we compute item similarities offline using training data set. Furthermore, for the purposes of our task we can precompute the item rank for every user as well. We can do this because we use the same recommendations for all transactions of a single user. We do not incorporate only of the transaction specific information other than user and item ids. However, on a real deployed application we would have incoming stream of new data which could help us generate better suggestions.

Updating the model with the new data can be done in two way – full recalculation and partial update. Full recalculation simply regenerates the item similarity matrix which is computationally expensive and therefore must be done offline. However, every purchase has a local impact on the user item set X. Since this set is user specific (eq. 4.2) we can update the set online after every transaction without updating the similarity matrix itself. This update is especially useful for new players with only few purchases as it reduces and eventually eliminates the cold start problem.

4.4 User clustering

In any game there are no identical two players, however they share some similar features. Some players may prefer to think and wage every decision while others can be more impulsive and make choices on the spot. Every user group has its own distinctive item preferences. We can exploit this assumption by recommending items based on a group of similar player purchases rather than all users. Once the player groups are identified we will use the baseline H_0 model (see chapter 4.1) for every group separately to generate recommendations.

We define every user using the item purchase history. User features are the frequencies of every item available in the shop. For example, the `purchase` dataset has 982 unique item (see chapter 3), thus the user feature vector is 982 feature long. Most of the users only buy a small fraction of items hence the user feature vector is mostly composed of zeros. Afterwards, we transform features using the techniques defined in section 4.2.

Having established the features we group users together into classes that have either hard or soft boundaries. Hard boundary clusters are determined using standard k-means algorithm using k-means++ initialisation (Arthur and Vassilvitskii, 2007). Hard boundaries mean that the user will be assigned to a single class only and receive suggestion as the most popular items from the group. The single class identification is a disadvantage which can be fixed used soft boundary multi-class assignments. The Mini-batch kmeans implementation (Sculley, 2010) can be used for faster clustering, however it provided worse result and because we were not tailoring for the fast times we chose the classical batch version.

Gaussian mixture model (GMM) allows probabilistic user classification to the groups. Users can be either described by a single class or exhibit properties of two or more classes. The probability of a user originating from a group can be seen as how well the group features define this user or what fraction of user behavior does it explain. For example say a player belongs to two categories with cluster A characterising 0.75 of the user's behavior and the rest 0.25 is explained by group B. As a result, the model shows the recommendations based on popular items in both groups.

Moreover, the generated item suggestion lists are merged taking into account the probabilities. Algorithm takes n unused items from group's A list and one from the B list. The number n is defined by the ratio of probabilities and in our example $n = \frac{0.75}{0.25} = 3$. Therefore, the resulting list will favor the best suited group but will have items from other groups as well.

Ideally, the feature vector would be extended with extra information provided in the dataset, such as user's level, however the features are not user specific and vary from transaction to transaction. They could be added if the clustering was executed independently for every transaction in the test set. However, this approach is computationally expensive and, considering the limited time span of this project, we did not pursue it any further.

4.5 Matrix factorization

Lee et al. (2012) demonstrated matrix factorization models as the most accurate predictors for the Netflix movie data. However, the improved prediction performance comes at the cost of training time. Taking into account the results of the mentioned study, we have explored two matrix factorization techniques – singular value decomposition (*SVD*) and non-negative matrix factorization (*NMF*).

Using SVD we decompose the user-item ratings matrix **R** ($n \times m$), with n users and m items, into two lower rank matrices. The first matrix **U** ($n \times k$) represents every user in k latent features while the second matrix **V** ($k \times m$) captures the item connection to the same k latent features. Because we use lower dimensional representation the SVD method is also called truncated-SVD.

The rating for an item j from a user a is then expressed as

$$R_{a,j} = \sum_{i}^{k} U_{a,i} \cdot V_{i,j} = \vec{U}_{a,:} \cdot \vec{V}_{:,j} \tag{4.3}$$

The regularised SVD minimizes the error between the original matrix and the dot product of the decomposed matrices

$$E = \sum_{i,j} ||\mathbf{R} - \mathbf{U} \cdot \mathbf{V}||^2_{i,j} \tag{4.4}$$

the original ratings matrix **R** is sparse and in the `purchase` dataset only 0.6% of the values are present. Therefore, the error term is computed iterating over the (i, j) index pairs that are present in the original matrix **R**. However, the reconstructed matrix $\mathbf{U} \cdot \mathbf{V}$ will have estimates for all of the entries.

Finally, after decomposing and reconstructing the ratings matrix we have a dense matrix with the rating estimates for all of the items. Every user row is converted into recommendations by sorting the row in a descending order. Such a list will contain both unseen and previously bought items.

The second factorization method, the non-negative matrix factorization (NMF), is similar to the SVD with additional constraint. The two decomposed matrices as well as the original matrix must be composed of non-negative entries. When the matrix **R** elements are expressed as counts the non-negative latent features in item matrix **V** can be interpreted as the amount of items demanded by the feature. As an illustration, say one of the latent features defines how cautious the player is. Then the item matrix slice of this feature will have higher values for the defensive items potions. In short, the features are only additive whereas in the SVD they could have negative impact as well.

We used sklearn implementation of both SVD and NMF methods.

4.6 Logistic regression

We experimented the usage of classical machine learning model such as logistic regression as it is well known and the feature set can be easily extended with additional information. Unlike previous models logistic classifier takes a list of features as an input thus requiring a different preprocessing of the data.

Every transaction was treated as a sample and the purchased item as a label. Since the item has multiple classes one-vs-rest training model was used. In this model a binary logistic regression classifier is trained for every item. In such classifier binary label is converted to a 1 if this item was purchased or 0 if any of the other item were bought. One-vs-all logistic regression has m times more weights than a simple logistic classifier. This model can be seen as an artificial neural network with no hidden layers as shown in figure 4.1. The input layer has n feature nodes and m output nodes give the probabilities of item being bought. The model can then be extended with a hidden layer.

In addition to item and user ids, every transaction has a list of player related attributes. Some of the 17 extracted features are player specific (they are constant for all purchases) while others change from transaction to transaction. The three transaction invariant features are player's age and gender and platform (whether it's Windows, Linux or Mac OS). The rest of the features are specific to playable character and define it's current state – attack, damage, armour, defence, dexterity, vitality, strength, focus, energy, health, gold, platinum currency, user level, experience.

Most of the features are numerical, however platform and gender are categorical thus they were converted to binary representation. To be more specific, platform has three categories (Windows, Linux and Mac OS) therefore three binary features were created to replace categorical feature. In the same manner gender was converted to two features – male and female.

Only two features, age and gender, are explicitly entered by the users. Other values are generated by the game system based on the user actions thus they are less susceptible to outliers. On the other hand, the player's birth year has two obvious outlier values – the year 2100, which we suspect is the highest permitted value, and the year 2013, which is the year when the game was played. In total 39,361 transactions (9.7%) have these values distributed among 20,045 users. The player's age was substituted with the

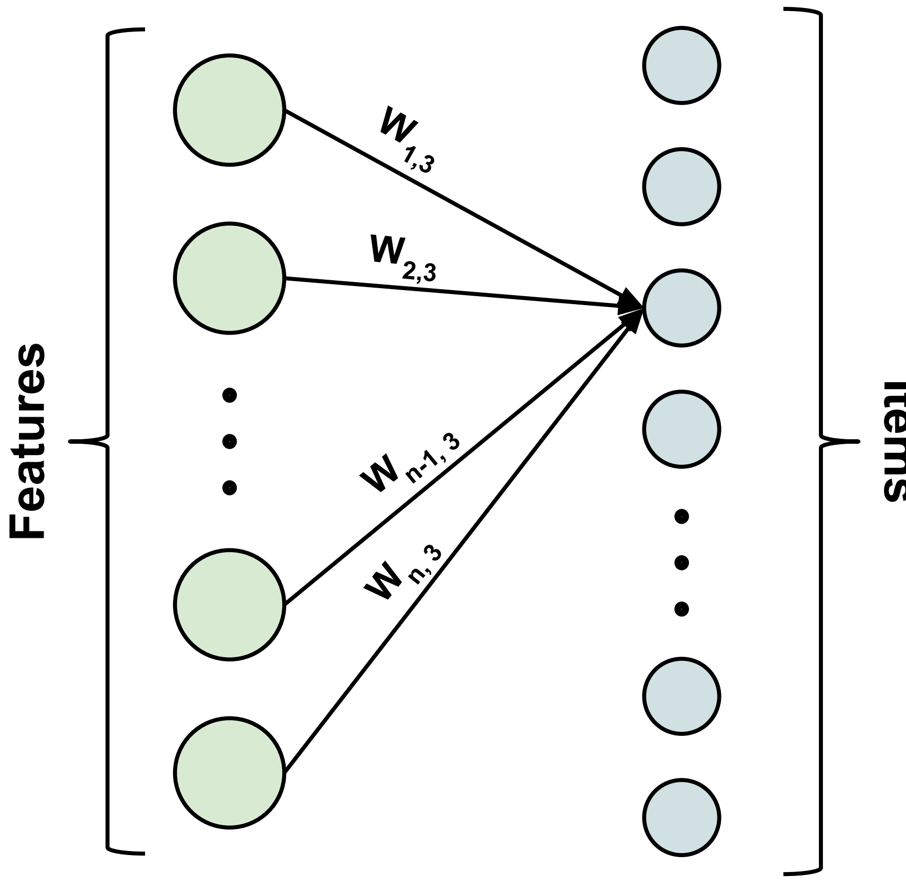


Figure 4.1: Multi-class logistic regression model representation. Weights for a single binary logistic classifier are shown.

average value of 23 years for all of the outliers.

The transaction feature vector $\vec{x}$ is then used in the classifier. A logistic binary classifier uses logistic function to estimate the probability of label

$$p(y|\vec{x}, \vec{w}_i) = \sigma(\vec{w}_i \cdot \vec{x}) = \frac{1}{1 + e^{-\vec{w}_i \cdot \vec{x}}} \tag{4.5}$$

where the $\vec{w}_i$ is the weight vector for the item i. The optimization problem is to find the weight vector that predict most of the purchases correctly. The overfitting is prevented adding regularization term to the loss function $E(Y,X)$. The most common choices for the logistic regression are L1 norm $\lambda||\vec{w}||$ and the L2 squared norm $\lambda||\vec{w}||^2$. Sometimes the regularization strength is expressed in the inverse form $C = {}^1\!/_\lambda$. A model regularised with L2 term exhibited better performance, thus it was used in most of the experiments.

The regularization penalizes all weights regardless of the feature mean and variance. Therefore, all features were scaled to be zero mean and unit variance. The sklearn implementation also normalizes the bias feature, however the bias was not scaled.

Every binary classifier in the one-vs-all model has an unbalanced positive and negative label ratio where negative labels indicate that some other item was purchased. Therefore, the transactions are re-sampled inversely proportional to the class training size. Without the resampling the classifier would maximize the accuracy by always predicting the most common label — that one of the other items will be purchased.

Chapter 5

Evaluation

In this chapter we provide the performance results of the evaluated models. We have evaluated the models on the `purchase` transaction dataset (see chapter 3) and used two different ways to split the data into training and testing subsets.

5.1 Evaluation setup

For the training and evaluation we use `purchase` dataset. The data set is splitting data samples into training and testing sets. As the classical recommender system collects item rating from users, the ratings are split (Lee et al., 2012). However, in our the data stream the atomic data sample is one transaction. Therefore, we sample transactions rather than the engineered ratings.

First we propose splitting the data by randomly sampling without replacement. The random transaction sampling provides a similar user distributions in both training and testing sets. For example a similar fraction of users will have only a few transactions and suffer from a cold start problem. In the latter sections we will refer to this dataset splitting as *random split* and the training set will have approximately 80% of transactions unless stated otherwise.

In order to evaluate a more realistic environment for our particular recommender we have split the data by time. In real-life application the system will try to predict the future transactions based on the previously observed data.

The provided dataset is over the period of 44 days which we split into first 35 days (5 weeks) for the training purposes and the remaining 9 days for the testing. Such a split will assigns 80.01% of transactions to the training set which is almost identical fraction as in the random split. Throughout this chapter we will refer to the split as *time split*.

The best performance would be observed when every single prediction is made using all of the observed data, however retraining the models for every transaction is very time consuming. As a result, we only train the models once using all of the training set. This can be seen as a batch update process with update period no shorter than 9 days.

The future data will contain more beginner players and therefore suffer from the cold start more than the randomly sampled data. In our setup the time split data contains twice as much player with a single purchase as in the random split dataset. Comparing model results on both splits gives indication of how well the model deals with the cold start problem.

Split type	Train size	Test size
time	326403 (80.01%)	81548 (19.99%)
random	327071 (80.17%)	80880 (19.82%)

Table 5.1: `Purchase` data set split sizes. The full `purchase` data set contains 407951 transactions.

5.2 Evaluation metrics

Classical recommender which infers item ratings is often evaluated by the error of the rank prediction (Gunawardana and Shani, 2009). The dataset is split into training and testing sets by taking a fraction of ratings from the dataset. Afterwards, the prediction error is computed which is the difference between the true ratings and the predicted values. The error is often expressed as root mean square error (RMSE) as it penalizes both positive and negative inaccuracies. However, our dataset is split into transaction datasets rendering the RMSE evaluation inapplicable.

As the main goal of the recommender is to produce a ranked list rather then a set of items we use mean reciprocal rank (MRR) as it captures the information of whether the item was suggested and how high in the list was it. MRR is expressed as

$$\mathrm{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\mathrm{rank}_i} \tag{5.1}$$

where $|Q|$ is the size of the test set. The rank_i is the purchased item number in the recommended item list of the i-th transaction. MRR is bounded metric ranging between zero and one. The best MRR is acquired when the system can always accurately predict which item will be bought. In other word, when user chooses the first item on the list. On the opposite side, MRR score of zero would either indicate that the item was not even on the list or at the end of the list when the score is close to zero.

Moreover, unlike mean rank MRR penalizes variation between high ranks (low in the list) less then variation between say first three places.

When evaluating the models suggestions ware generated for every transaction in the testing set and the rank of the purchased item was recorded. Therefore, MRR score is an average over many different users and items.

Some of the models, such as baselines, generate the same item list for all purchases by same user. This constant behavior enables us to compute theoretical upper limit. The best performing model with such behavior would rank items based on their occurrence in the test set. It would be the same as the user average baseline H_1 (see chapter 4.1) trained on the *test set*.

5.3 Baselines

We start by evaluating the baselines which will be used in the letter section for comparison.

The MRR score converges after about a thousand test samples, however we are using 20% of data for evaluation which is 80 thousand transactions. As a result the error is often negligible and too small to plot in later parts of the evaluation.

	Time split	Random split
H_0	0.1753 ± 0.0013	0.1864 ± 0.0014
H_1	0.1374 ± 0.0013	0.2643 ± 0.0014
$H_0 + H_1$	0.1905 ± 0.0014	0.2677 ± 0.0014

Table 5.2: Performance of baselines and their mix.

The different global popularity scores, presented in table 5.2, point out that item distributions are slightly different in the different data split methods. This is an expected consequence of the time split test having more inexperienced players, that prefer lower level items.

The $H_0 + H_1$ in the table 5.2 shows the performance of mixed baseline. Mixed baseline alternates between using H_0 and H_1 based on a threshold. When the player has less then 3 transactions in the history the globally popular items will be suggested — the baseline H_0. Otherwise, the previously bought items by user will be shown as defined by baseline H_1.

The baseline switching method indicates that randomly split dataset has less newcomers without the history as the H_0 was hardly used and the score matches the one of H_1 baseline. On the other hand, while using the time split the baseline H_0 was used for the 29.16% of test set transactions.

5.4 Slope One

Previous papers reported Slope One performance that was comparable with baselines, which were similar to ours, and slightly worse than matrix factorization methods such as reg-SVD and NFM (Lee et al., 2012). The study was performed on movie rating database where all present ratings were in the range from 1 to 5.

Slope One performance as a MRR score on the `purchase` dataset is given in the table bellow. We observe that the model underperformed compared to the baselines.

Preprocessing	Time split	Random split
none	0.049	0.092
scale	0.069	0.126

Table 5.3: Performance of Slope One model with time split and random sampling.

Slope one finds items with similar ranks and suggest them. When we express ratings as the purchase counts, the same score has different meaning for a new user and someone with a big buying history. Therefore, we scale the features to a constant maximum value per user. Afterwards, users' scores are bounded and use full range regardless of the player's activity.

We can see that the Slope One method shows poor results compared to the established baselines. Therefore, we do not pursue this model any further.

5.5 User clustering

We have grouped 40143 users together and suggested the most popular items within the groups. After experimenting with the feature transformations described int he section 4.2 we found out that the unit-vector representation works best for the clustering purposes.

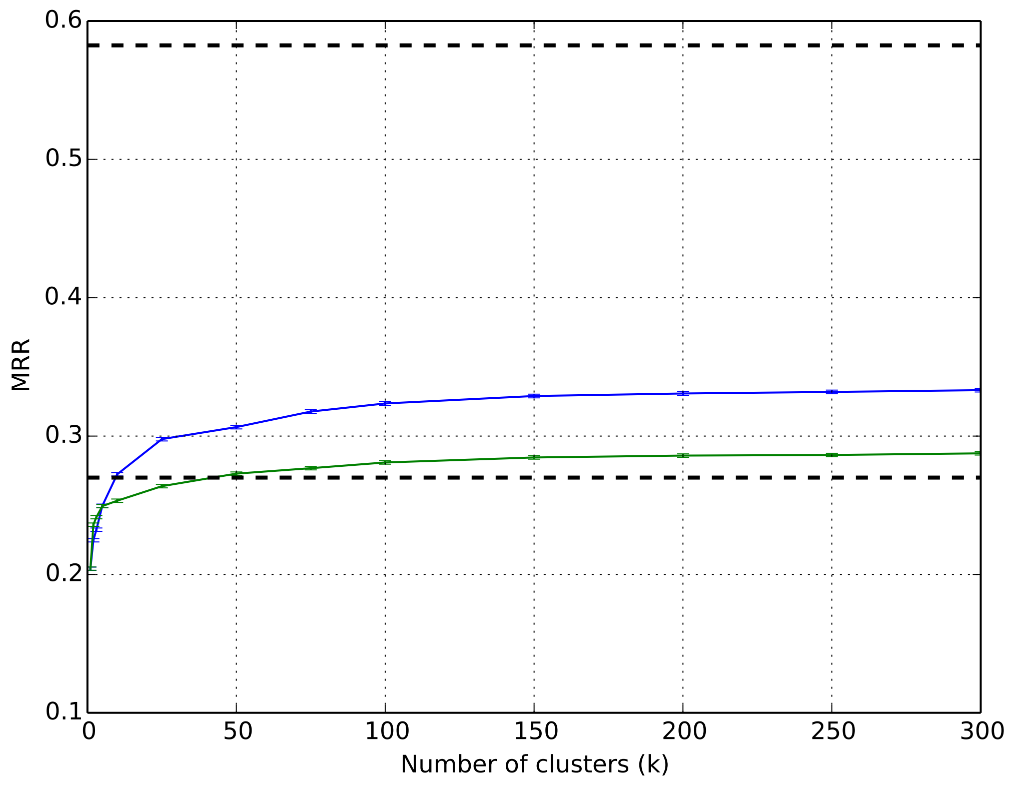


Figure 5.1: User grouping model performance. Blue and green lines represent hard and soft boundary clustering respectively. The upper dashed line shows theoretical upper score limit for the user test set. The lower dashed line shows the H_1 score which would be observed when number of cluster is equal the number of users. Training set contains randomly selected 80% of all transactions.

The only free parameter in both k-means and Gaussian mixture model (GMM) is the number of clusters/components – k. A single cluster will yield the same results as H_0 baseline as all users will receive the same recommendations. On the other hand, the maximum possible k value is the number of users. In such model every cluster will have a single user and that user will receive recommendations as a list of most often purchased items which is our H_1 baseline. From the figures 5.1 and 5.2, it can be seen that grouping users exhibits better performance than the both baselines. This confirms our hypothesis that players, in fact, share shopping patterns but not all of them.

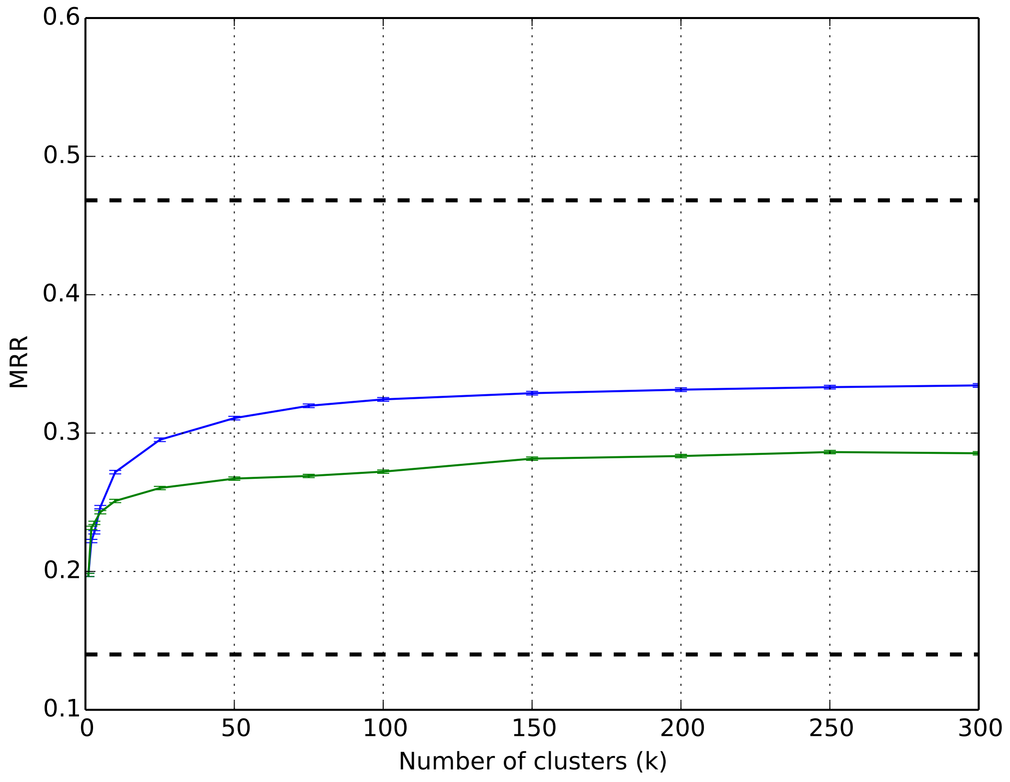


Figure 5.2: User grouping model performance. Blue and green lines represent hard and soft boundary clustering respectively. The upper dashed line shows theoretical upper score limit for the user test set. The lower dashed line shows the H_1 score which would be observed when number of cluster is equal the number of users. Training set contains the first 5 weeks of the data and the test set has the remaining 9 days.

The suggestions for all purchases of a single user are the same thus we can bound the MRR score from above if we were to suggested best possible list. Such a list is all the items in the test set ordered by quantity. In other words, it is H_1 baseline trained on the test set.

From both random (fig. 5.1) and time (fig. 5.2) data splits we can see that clustering users with hard-boundary k-means algorithm yields better results then the Gaussian mixture model. Furthermore, we can see that the performance on time and random split is similar which indicates that the model performs well under cold start. We speculate that the newcomers and experienced players are successfully segregated into separate groups. This allows to offer appropriate items to the users.

Additionally, the identified user groups can be reused for other purposes. For example, a custom promotions could target specific group or an in-game event tailored for a class of players.

5.6 Matrix factorization

In this section we present the evaluation of the singular value decomposition (SVD) and non-negative matrix factorization (NMF) algorithms on the `purchase` dataset.

Firstly, we determine the optimal number of latent features, f, which is then used is the latter experiments. The figure 5.3 shows the SVD predictive performance as MRR score using the random data split. The NMF is not shown as it underperformed with all tested f values.

As the number of the latent features f grows the MRR score should asymptotically reach the baseline H_1 score as the reconstruction error gets smaller. When $f = \min(\#items, \#users) = 982$ the score should be exactly as H_1 as the original ratings matrix can be decomposed as $\mathbf{R} = \mathbf{R} \cdot \mathbb{I}$ where the $\mathbb{I}$ is the identity matrix. Such solution would have a zero reconstruction error and therefore would be optimal solution. However, when we take into account the regularization term the total error would still be positive. As a result of regularization, the asymptotic MRR score for the SVD is 0.2022 ± 0.0012. The asymptote was computed by running experiment with the SVD and $f = n = 982$ values.

We present our results of the SVD and NMF models with the various rating matrix $\mathbf{R}$ transformations in the table 5.4. We have used $f = 10$ latent features for the experiments as it proved to give good results int he previous test (fig. 5.3). We note that the SVD is better under all rating transformations. This is likely to be caused by the negative latent feature impact which cannot be expressed in the NMF model.

It can also be observed that the rating transformation did not improve the predictions. However, surprisingly the binary representation performed only slightly worse even though it caries less information then the full counts.

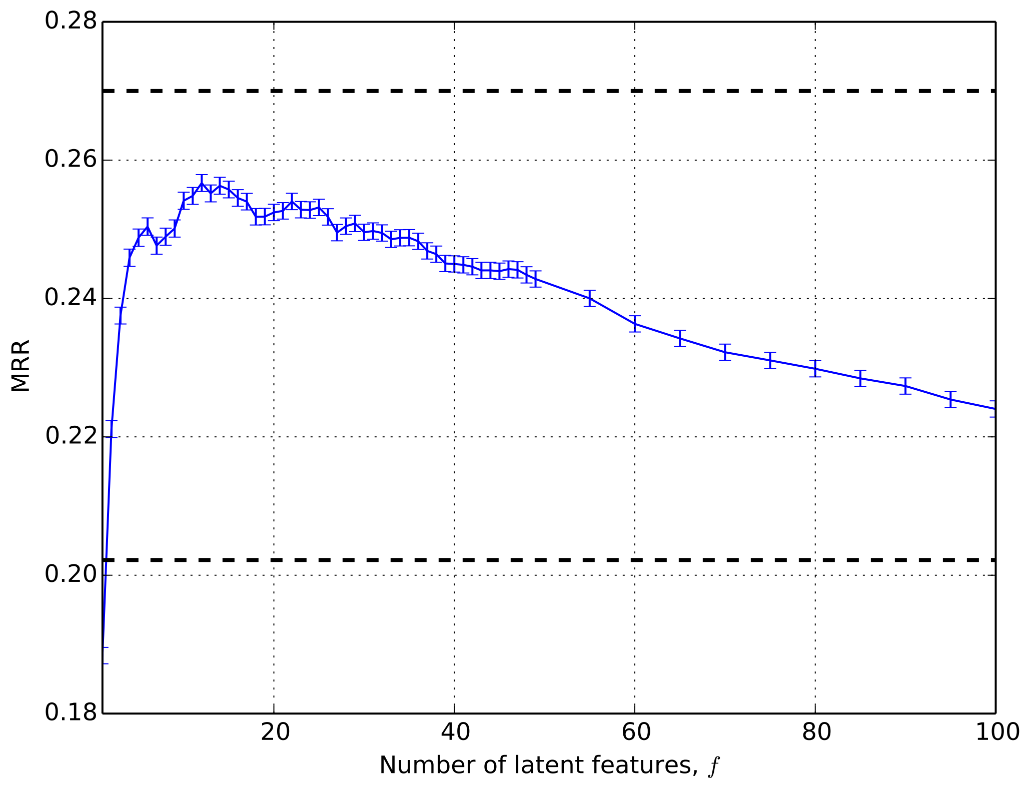


Figure 5.3: MRR score using a singular value decomposition with varying number of latent features f to estimate the item rankings. The lower dashed line shows the asymptotic score value when number of latent features is equal to the number of items ($f = n = 982$). The upper dashed line indicates the baseline H_1 – the user average.

	Time split		Random split	
Preprocessing	NMF	SVD	NMF	SVD
none	0.1257 ± 0.0008	0.1495 ± 0.0009	0.1924 ± 0.0012	0.2541 ± 0.0012
unit vector users	0.0771 ± 0.0005	0.1279 ± 0.0008	0.0793 ± 0.0005	0.2005 ± 0.0011
TF-IDF	0.0771 ± 0.0005	0.1186 ± 0.0007	0.0793 ± 0.0005	0.1844 ± 0.0011
log scale	0.0771 ± 0.0005	0.0942 ± 0.0006	0.0793 ± 0.0005	0.1071 ± 0.0007
binary	0.0376 ± 0.0003	0.1262 ± 0.0008	0.0062 ± 0.0001	0.2114 ± 0.0011
zero mean items	—	0.1378 ± 0.0009	—	0.2253 ± 0.0012

Table 5.4: Performance of SVD and NMF methods with $f = 10$ latent features. Zero mean item preprocessing does not work with NMF as the elements have to be non-negative.

Furthermore, by comparing the evaluation results from both the time and random data sets, we infer that matrix factorization methods indeed suffer from cold start problem.

Ultimately, SVD proved to be better suited than the NMF. However, neither of them generated better suggestions than the baselines. Therefore, we *do not recommend* to use it in a real-life application.

5.7 Logistic regression

Logistic regression one-vs-all model was trained and evaluated using the `purchase` dataset. As with the other models transactions were separated into training and testing set via random sampling and time threshold split (section 5.1). We provide results for both data splitting techniques.

We start by demonstrating that item filtering for the one-vs-all algorithm reduces noise and improves predicting performance as well as speeds up the training time. Followed by the evaluation of different dataset splits. Afterwards we discuss the impact of the additional features to the model. We conclude by stating best achieved performance together with a discussion and comparison to the other models.

The logistic regression classifier exhibited best performance under L2 regularization, however in some sections we provide same evaluation using L1 regularization for comparison purposes.

5.7.1 Filtering items

Some items are more popular than others and the unpopular items only introduce noise to the system. In the `purchase` dataset (see section 3) 59 items (6%) are only bought once. Moreover, one third of items are acquired ten times or less and half of the items are purchases less then 30 times. For comparison on average every item is bought 415 times. We have exploited this fact by filtered the item list to keep only the popular ones.

Evaluation is also done on the filtered test set. Thus the mean reciprocal rank (see section 5.2) will be bias towards the higher scores as the rare and hard to predict items were omitted. To account for all the data we assume that for every transaction in the test set involving the filtered out item the suggestion list did not contain that item. The reciprocal rank for such a item list would be zero as the item is not on the list (the rank is infinite).

As a consequence, the list of the test set reciprocal ranks will be extended with the amount of zeros equal to the number of transactions that were filtered out. Alternatively, the mean reciprocal rank computed on the filtered test set (MRR_{test}) can be adjusted by multiplying it by the filtered data set size ($N_{filtered}$) and dividing by the full data set size(N_{full}):

$$MRR_{scaled} = \frac{MRR_{test} \times N_{filtered}}{N_{full}} \tag{5.2}$$

We ran the experiment with varying item set sizes on the data set with random sampling. As expected in figure 5.4 we can see that with only few items left the bigger part of data is discarded. The MRR score is high as all items are at the top of the suggestions list when list contains only few items. On the other hand, the scaled MRR score drop low. In the most extreme scenario the system is always recommending to buy the most popular item.

However, filtering the hundred most popular items that account for the 70% of the dataset improves the performance. Each component of the one-vs-rest logistic regression model predicts whether a particular item will be purchases or any of the others. As explained in the section 4.6 the transactions were sampled with replacement inversely proportionally to the class labels. However, for the rare items this technique does not work properly as the only few available transactions become highly over-sampled. Without enough of data for the rare items, the binary classifier fails to distinguish the features that caused the purchase from the noise. Resulting in misleading prediction which pushes some rare items higher in the recommendation list then they should be. Such rare item behavior acts as a noise to the whole system.

Additionally, the filtered item count effectively reduces the number of one-vs-rest logistic regression models needed which speeds up the computation time. Training the model on top hundred items reduced computational speed in terms of magnitude. This performance boost could allow more frequent model updates for a real-time application. Faster updated can lead to better prediction performance in an early stage stage of the game or if the item demand is changing due to game balance, item or price changes.

5.7.2 Data split

We evaluated logistic regression model with exactly the same parameters but different data splits. The experiment was performed on the filtered dataset keeping 95 item that have more than 1000 transactions. Feature weights were regularized using L2 method with the inverse strength $C = 0.05$.

Split type	MRR
time	0.2276 ± 0.0015
random	0.2254 ± 0.0015

Table 5.5: Logistic regression scaled MRR performance. The test set was generated using two different techniques (section 5.1).

We can see that the performance is the same regardless of the data split method which indicates that the model does not suffer from cold start problem. For this reason, logistic classifier can be valuable and accurate for the early stages of the game or when the churn rate is high where the fresh player count with minimal transaction history is high.

5.7.3 Extra features

We applied k-means clustering with 25 clusters (section 4.4) on the dataset to generate new features for the classifier. For this experiment the user feature vector was scaled to a unit length vector. Afterwards, the categorical cluster id feature was binarised into $n = k = 25$ binary features $(c_0, c_1, c_2, \ldots, c_{k-1})$ indicating whether the user belongs to the cluster. Since the hard boundary k-means implementation was used, every used belongs to a single cluster:

$$\sum_{i=0}^{k-1} c_i = 1, \;\; c_i \in \{0, 1\} \tag{5.3}$$

Firstly, as a proof of concept we have trained logistic regression using only cluster ids as input features. We used 95 most popular items (section 5.7.1) and $k = 25$ clusters. The training set was generated using random transaction sampling.

The final scaled MRR score is 0.2747 ± 0.0013 and in the figure 5.5 we show the weights for all items. On the horizontal axis we have 25 weight ids – one for each cluster. Every line represents weights for a single item thus there are 95 lines and most of them overlap. As only a single cluster has a particular transaction all other will have zero and therefore will not contribute. As a result, every vertical weight slice will give the independent item ordering.

We can see that the cluster ids have some information which gives better results than H_0 baseline or featureless logistic regression. However, grouping user into the same number of cluster and then using H_0 item popularity (section 4.4) gives a better score of 0.2907 ± 0.0013. Therefore, we can conclude that logistic regression trained on cluster ids failed to learn the best item rankings for each of the cluster. Additionally, training the logistic regression takes more time than training the H_0 baselines. Equivalently to the user clustering in section 4.4 the MRR score increases with the cluster count.

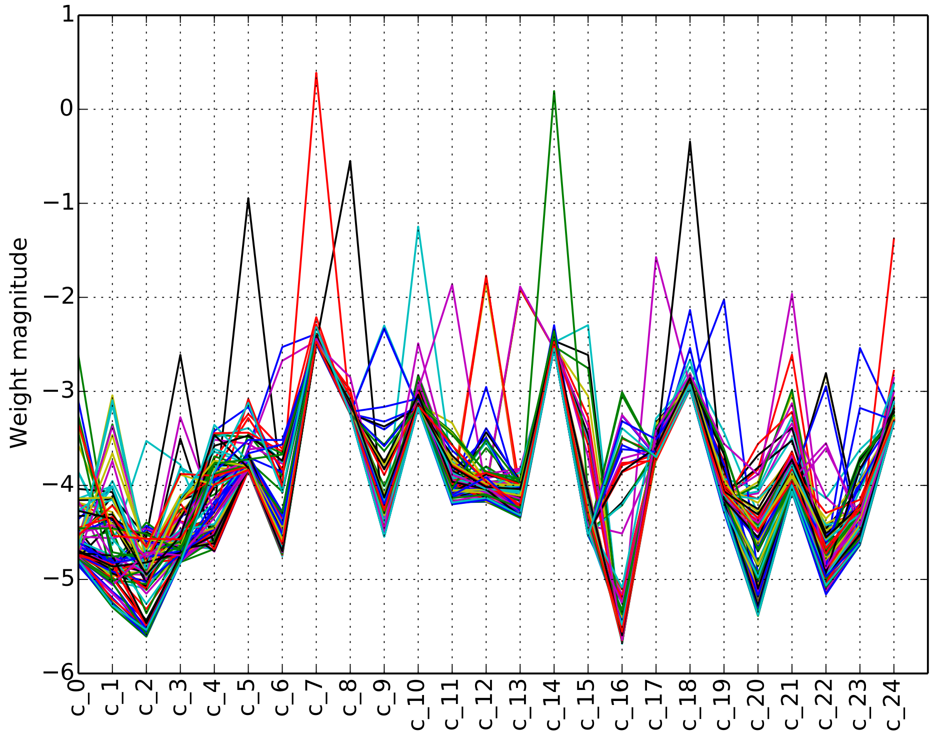


Figure 5.5: Weight matrix representation for the logistic classifier trained on the cluster ids alone. Cluster/weight ids are listed on the x-axis. Every line represents a weight vector for a single item.

Next we have combined the original 20 transaction specific features with the clustering features resulting in a list of 45 features. We used the same parameters, such regularization method and strength, so that the results would be comparable. However, we do not claim that this is the optimal parameter set. The results are given in the next section.

5.7.4 Conclusion

Logistic classifier trained with the bias feature only learns the overall item demand distribution which is exactly what the H_0, the global average baseline, does. Additional purchase related features provide more information to the model and increase the predictive capacity.

Furthermore, the model infers item purchases from the feature list rather than looking for similar items or users. As a result, cold start is not a problem for the model as long as there enough training samples with features like user level and experience indicating newcomers.

On the other hand, model does not have explicitly expressed user transaction history as the transactions are identified by the features only and the user id is not among the features. The other methods have this information via the user-item matrix.

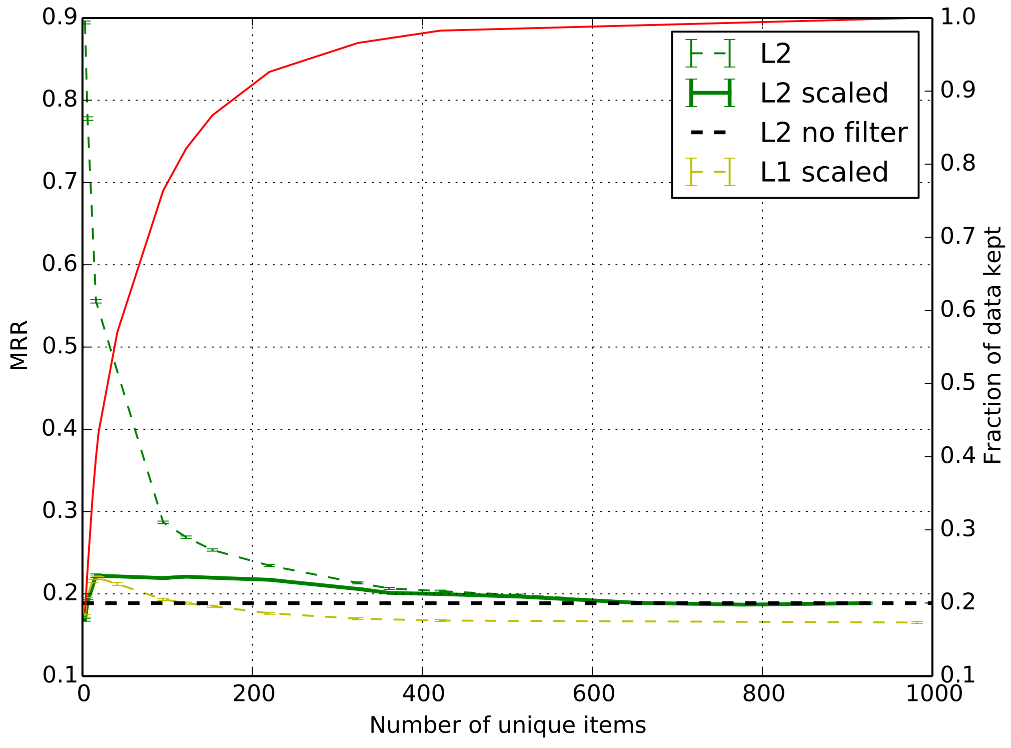
Features	MRR
bias	0.1961 ± 0.0012
transaction	0.2254 ± 0.0015
25 cluster ids	0.2747 ± 0.0013
combined	0.2806 ± 0.0015

Table 5.6: Logistic regression classifier performance. Transaction features in the second row are the purchase specific properties described in the section 4.6.

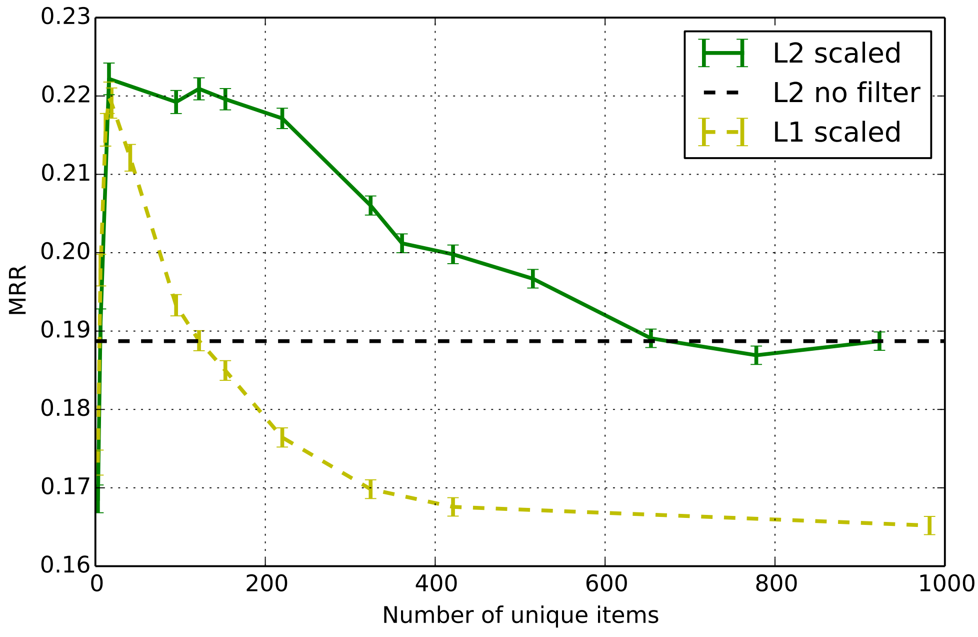
Finally, we demonstrate that by filtering the popular items we can increase systems predictive performance as a consequence of reduced noise generated by the undertrained rare item predictions. Additionally, the required training speed is reduced allowing more frequent model updates in an online system.

Future work would focus on further utilizing the flexibility of the feature based model by extending the list. Firstly, we suggest appending the user characterising features learned from the matrix decomposition such as SVD (section 4.5).

Secondly, in the data preprocessing we have discarded most of the data which contains game related events. The feature list could be further extended by the events happening in a short time window before the user has visited the shop. As an example, a binary feature indicating whether the last mission or a quest was successful could created as well as the time spent completing the quest.



(a) Full plot.



(b) Zoomed version.

Figure 5.4: Logistic regression performance when filtering items list. In figure 5.4a red line indicates the fraction of transactions that were used (right y-axis). The green dashed line shows the MRR score from the filtered test set while the solid green line shows the adjusted score. For comparison same experiment with L1 regularization is given in the yellow dashed line. Additionally, MRR score with L2 regularization and all items is shown as a horizontal black dashed line. Figure 5.4b shows the bottom part zoomed in. The experiments were run with the initial 20 features and the inverse of regularization strength C of 0.1. The data set was split by random sampling.

Chapter 6

Conclusion

In this chapter we summarize and asses the performance and applicability of the explored recommendation systems. We present our main findings in three bullet-points according to our initial goals. Then, we give our recommendation for the model that should be used in real-life application by the deltaDNA company. We finalize the thesis by sharing other interesting findings and suggesting possible future work.

Rating engineering We explored multiple artificial rating options, and found that the raw item purchase frequencies worked best when directly used in generating the item suggestions. We speculate that the results are caused by our evaluation method. Because of the static recommendations per user and uneven item distribution, as long as we rank the top item high the mean reciprocal rank (MRR) is high.

At the same time, when the rating were used to generate intermediate results, such as user groups, the unit-vector transformation proved to be the best option. The unit-vector transformation reduces the distance between players with similar shopping tastes regardless of the history length.

Cold start We have created a framework to check how different models are impacted by the cold star. For this we split the initial dataset using two different methods and tested the model on both of them. As a proof of concept, we tested the framework on the global item popularity baseline. It correctly indicated that the baseline does not suffer from the cold start problem.

Furthermore, we have showed that user clustering with k-means combined with the H_0 baseline and logistic regression perform equally well on both data splits. The user clustering solves the problem by segregating novice and adept players into separate groups. On the other hand, the logistic regression learns the item rankings from the transaction specific features. As a result, we claim that these two models are immune to the cold start problem.

Best model We have evaluated different recommendation strategies and found that the best performance is exhibited by the user clustering. The k-means algorithm is able to identify similar user groups better than the latent feature representation in the matrix factorization methods.

Having said that, we were unable to optimally exploit the logistic regression model because of extensive customization options. In our test it came second but it likely could be extended to achieve better results.

In addition, we also investigated the optimal number of items shown in a pop-up messages during the game. This was evaluated using the mean reciprocal rank metrics (MRR). The inverse of MRR score gives us the lower bound for the estimate. By using the score, of our best performing model – the k-means clustering, we found that the item list in the pop-up offers should be no shorter than three items long.

Finally, we recommend the user clustering method with k-means for the in-game shop recommendation systems. In our tests this system had the best performance and the universality of the method could be easily adapted in other areas. Examples include special events for small subset of players, or promotional offers for small user groups.

6.1 Other remarks

In this section we state our findings that are not directly related to the original project goals, yet we believe them to be of significance. The observations are listed in no particular order:

- The logistic regression model was found to be the most flexible model hence we haven't fully explored it in the limited time. The model can be easily extended by providing new information via features. As an example, we have successfully incorporated the results from the user clustering model into the logistic regression which increased the prediction performance.

- Some models do not work with the rating-less datasets. We found that the ratings are crucial for the Slope One model and without them the model performed a lot worse even then the simpler baseline model.

- The global item popularity and the user item popularity baselines proved to be a strong models with relatively high MRR scores. This was partly caused by the immutable recommendation list for all of the purchases made by a single user.

6.2 Future work

The recommendation field covers a wide range of models. In the limited time of this project we were only able to try a few of them and could not experiment with all of our ideas.

We provide a list of future directions that can be pursued:

- Logistic regression proven to be configurable with alternative cost functions, optimization methods and regularization. We suggest exploring the model further and more in-depth.
- In section 4.6 we showed that multinomial logistic regression can be seen as an artificial neural network without any hidden layers. A natural extension would add a hidden layer to the model.
- Extending the dataset with the sell events as the negative counts in the frequency based rating did not work for our purposes. However, it may work when the recommender is able to impact the user decisions by exposing different items. Furthermore, trading event between players can also be used as they indicate the desired items.
- Because most in-game item require the player to reach a certain level to be able to use the item, we strongly suggest explicitly modeling the users by their experience (McAuley and Leskovec, 2013).
- In order to make the models applicable on a real game, online versions of the methods should be investigated in terms of the speed and maintained prediction performance.
- All of the non-transaction related in-game events were filtered out in the early stages of the data preprocessing. Future work could investigate the usage of these event in the developed models, especially the logistic regression.

Appendix A

Event attributes and categories

Here we provide full list of the event attributes present in the dataset. The names are shown as they appear in the original files.

The 76 attributes:

eventID, partnerID, applicationID, **userID**, sessionID, eventDate, **eventTimestamp**, acquisitionChannel, action, **armour**, **attack**, **birthYear**, characterClass, characterGender, characterID, characterName, clientVersion, collectInsertedTimestamp, convertedProductAmount, currentSkillLevel, currentValue, **damage**, dataVersion, **defence**, **dexterity**, **energy**, **eventLevel**, **eventName**, eventStoreInsertedTimestamp, experienceGained, **focus**, **gender**, **gold**, guildName, **health**, inviteType, isInitiator, isInviteAccepted, isTutorial, mainEventID, missionID, missionName, newSkillLevel, newValue, **parentEventID**, paymentCountry, **platform**, **platinum**, **productAmount**, productCategory, **productName**, productType, recipientID, revenueValidated, senderID, serverVersion, socialType, statPointsGained, **strength**, success, transactionID, transactionName, transactionServer, **transactionType**, transactionVector, transactorID, uniqueTracking, userCountry, **userLevel**, **vitality**, **xp**, accountName, deviceType, manufacturer, operatingSystem, operatingSystemVersion.

The 26 highlighted attributes were used in either event filtering, merging or model training.

Every event belongs to one of 15 categories that are defined by the `eventName` attribute. List of all categories:

missionCompleted, characterCreated, missionFailed, newPlayer, `transaction`, `inviteReceived`, `missionAbandoned`, `guild`, `clientDevice`, `gameEnded`, `social`, `levelUp`, `gameStarted`, `inviteSent`, `missionStarted`.

Appendix B

Raw data filtering

We provide the list of Unix commands that were used to filter the data. Commands can be either chained into a pipeline or executed separately. We use the notation of `input` for the data source.

Step 1 Keep only the transaction related events.

```
grep 'transaction' input
```

Step 2 Remove the parent event lines.

```
egrep '[SALE|TRADE|PURCHASE],$' -v input
```

Step 3 Keep events when user received an item (either buying or trading).

```
grep 'RECEIVED$' input
```

Step 4 Remove spam lines with VIRTUAL_CURRENCY and zero amount.

```
grep '0,VIRTUAL_CURRENCY' -v input
```

Step 5 Remove all sale event where user sales an item to a shop.

```
grep 'SALE' -v input
```

Step 6 Remove trade events between players.

```
grep 'TRADE' -v input
```

Step 7 Remove all virtual currency transaction records.

```
grep 'VIRTUAL_CURRENCY' -v input
```

Bibliography

Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749.

Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics.

Bishop, C. M. et al. (2006). *Pattern recognition and machine learning*, volume 1. springer New York.

Croft, W. B., Metzler, D., and Strohman, T. (2010). *Search engines: Information retrieval in practice*. Addison-Wesley Reading.

Gunawardana, A. and Shani, G. (2009). A survey of accuracy evaluation metrics of recommendation tasks. *The Journal of Machine Learning Research*, 10:2935–2962.

International Telecommunications Union (2013). Percentage of individuals using the internet 2000-2012.

Jones, E., Oliphant, T., Peterson, P., et al. (2001–). SciPy: Open source scientific tools for Python.

Lau, C. W. (2011). News recommendation system using logistic regression and naive bayes classifiers.

Lee, J., Sun, M., and Lebanon, G. (2012). A comparative study of collaborative filtering algorithms. *arXiv preprint arXiv:1205.3193*.

Lemire, D. and Maclachlan, A. (2005). Slope one predictors for online rating-based collaborative filtering. In *SDM*, volume 5, pages 1–5. SIAM.

Linden, G., Smith, B., and York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80.

McAuley, J. J. and Leskovec, J. (2013). From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *Proceedings of the 22nd international conference on World Wide Web*, pages 897–908. International World Wide Web Conferences Steering Committee.

McKinney, W. (2010). Data structures for statistical computing in python. In van der Walt, S. and Millman, J., editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56.

Nair, N. (2009). Interview with pendragon, the future of dota-allstars.com.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Riot Games (2013). League of legends players summit a new peak.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM.

Sculley, D. (2010). Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178. ACM.

Van den Poel, D. and Buckinx, W. (2005). Predicting online-purchasing behaviour. *European Journal of Operational Research*, 166(2):557–575.