TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology

Mihail Smirnov 186022IAIB

# Recommender system engine for Apollo

Bachelor's Thesis

**Supervisor**
Evelin Halling
PhD

Tallinn 2021

# Author's declaration of originality

I hereby certify that I am the sole author of this thesis and this thesis has not been presented for examination or submitted for defence anywhere else. All used materials, references to the literature and work of others have been cited.

| | | |
|---|---|---|
| Author: | Mihail Smirnov | .................................... |
| | | (signature) |
| Date: | 18.05.2020 | |

# Annotatsioon

Soovitussüsteemid on muutumas selliste ettevõtete jaoks väga tõhusaks vahendiks nagu e-kaubandus. Nad suudavad õppida inimese käitumisel põhinevaid mustreid ja soovitada kaupu, mis oleksid konkreetsele inimesele huvitavad, suurendades seetõttu müüki.

Lõputöö eesmärk on analüüsida võimalust tutvustada soovitussüsteemi elemente Apollo Grupi e-kaubanduse kanalites, nagu näiteks veebipõhine raamatupood ja veebipõhine kinopileti pood.

Lõputöös kirjeldatakse andmete analüüsi protsessi ja selle vastavust nõuetele, andmete puhastamist vigastest kirjetest, ebatüüpilise inimkäitumise eemaldamist (näiteks edasimüüjad), mõõdikute ja meetodite valimist, erinevate algoritmide valimist ja võrdlemist ning tulemuste ja probleemide manuaalset valideerimist ja selgitamist.

Selle lõputöö tulemusena tehti kindlaks, et Apollo esitatud andmed on soovitussüsteemi kasutamiseks piisavad ja nõuetele vastavad. Testitud algoritmide põhjal leiti, et SVD ++ annab parimaid tulemusi nii automaatse hindamise kui ka käsitsi kontrollimise seisukohast. Külmkäivituse probleemil ja plahvatuslikul ennustusprobleemil on mõlemad toimivad lahendused.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 30 leheküljel, 8 peatükki, 1 joonist, 9 tabelit.

# Abstract

Recommender systems are becoming a very powerful tool for businesses such as e-commerce. It is able to learn patterns based on human behaviour and recommend goods that would be interesting for a specific person, subsequently increasing sales.

The aim of the thesis is to analyze the possibility of introducing recommender system elements to Apollo Group e-commerce channels, such as online book store and online cinema ticket store.

The thesis describes the process of data analysis and determining its compliance to the requirements, clearing the data from faulty entries and removing non-typical human behaviour such as resellers, choosing evaluating metrics and methods, determining baseline algorithms, choosing and comparing advanced recommender algorithms and manually validating and explaining the results and problems.

As the result of this thesis, it was determined that the data provided by Apollo is sufficient and compliant to be used for the recommender system. From tested algorithms, SVD++ was found to produce the best results, from an automatic evaluation standpoint as well as manual inspection. The cold start problem and the exploding prediction problem both have viable solutions.

The thesis is in English and contains 30 pages of text, 8 chapters, 1 figure, 9 tables.

# List of abbreviations and terms

| | |
|---|---|
| Apollo | collective term for Apollo Group divisions |
| Model | set of data alongside an algorithm that is capable of making decisions based on learned patterns |
| POC | Proof of Concept |
| SVD | Singular Value Decomposition |
| CF | Collaborative Filtering |
| CSV | Comma Separated Values |
| DB | Database |
| CPU | Central Processing Unit |
| MAE | Mean Average Error |
| RMSE | Root Mean Square Error |

# Table of Contents

# List of Figures

# List of Tables

# 1 Introduction

Machine learning, or more broadly Artificial Intelligence, is becoming more and more popular in very different fields and being used for different applications. Machine learning is a tool that could be used for simulating complex decision making without human intervention based on patterns in data.

Recommender system is generally an algorithm that is capable of making thoughtful recommendations to clients so that recommendations are the most likely to interest the client. Such systems are most commonly used in e-commerce businesses, video and music streaming platforms. Data that is used as an input for recommender systems could include:

- purchase history
- click/search history
- birth date, sex, age
- geolocation
- and other

This thesis is based on the project from Apollo I'm doing as an employee from executor company Icefire OÜ. The problem to solve is to improve the interest of people in recommended goods, subsequently improving the revenue stream. This includes such use cases as, but not limited to: advertisement banners on the Apollo websites, an additional area with prioritised recommended goods (mainly books), prioritised search bar results, recommendations in receipt letter, recommendations on self-service kiosks and others. The solution is to create such a recommender systems engine that could predict future goods that are the most likely to be bought by a specific user.

# 2 Project justification

## 2.1 Reasoning

The reasoning for this project is that there are no available tools and ‘as a Service’ products that would not compromise on something. That is because each model should be supervised and tweaked differently, train data should be inspected closely and cleared in a specific way, which is hard to generalize. General recommender systems ’as a Service’ such as Keelabs[1], Recombee[2], Yuspif[3] compromise on flexibility, speed and effectiveness over handmade recommender system. More importantly, you have more control over the handmade model. It allows to optimise and configure it the way the clients want it not purely based on evaluation metrics. In contrast to general solutions, we can introduce market-specific features, such as prioritising promotional films and books, constraining search to a specific category and many others. Furthermore, most of the general solutions only use 1st and 2nd generation recommender algorithms (see section 3.2), while we are building service with migration to deep neural networks in mind.

Another reason is that Apollo might create a service based on this recommender system, combining data from different sources and tenants to achieve better results. This service could then be proposed to other businesses across the globe ’as a Service’ solution for book stores and cinemas.

## 2.2 POC scope

Proof of concept goal for this project is to find such an algorithm or algorithms, that could be able to recommend items that people are the most likely to buy in a form of the top n most likeable items for a person.

POC limits include:

- some items might not be accessible or out of stock

[1]Kealabs - https://kealabs.com/recommendations-api
[2]Recombee - https://www.recombee.com/
[3]Yuspify - https://yuspify.com/blog/recommendation-as-a-service/

- ignore time frame, old items are as important as new ones
- ignore repeatable purchases, one purchase is as important as 100 purchases of the same item

**Theory**: Every person is unique, but groups of people have similar tastes. Every item is unique, but groups of items are similar. Similar groups of items are preferred by similar groups of people.

## 2.3 Bigger picture

This recommender system engine is a part of a bigger service that would be able to automatically:

- receive assortment and sales data increments once a day
- clean the data given constraints
- enrich data for training
- train model
- import new model to working service
- support multi tenancy
- generate top n given constraints such as:
  - location
  - price
  - availability
  - category/tags

Such service might also be able to:

- make personalised discounted offers
- make personalised offers for a bonus card
- conditionally discount online cart

Such a service will support the potential migration of the algorithm to deep neural networks. Deep neural networks will be able to potentially improve the quality of the recommendations by taking into account data such as:

- language
- device type
- cookies
- search history
- online cart

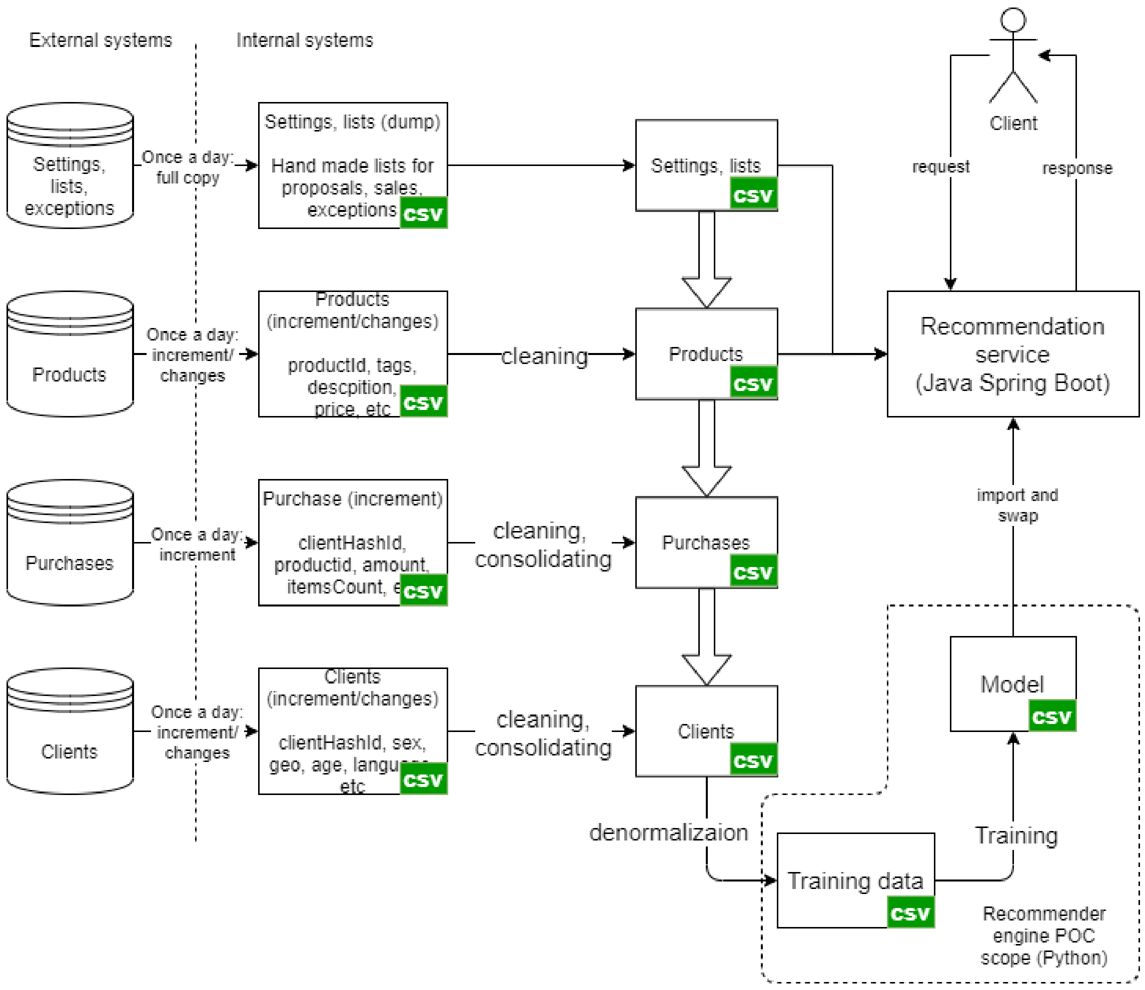
External systems
Internal systems
Settings, lists, exceptions
Once a day: full copy
Settings, lists (dump)
Hand made lists for proposals, sales, exceptions
csv
Settings, lists
csv
Client
request
response
Products
Once a day: increment/ changes
Products (increment/changes)
productId, tags, descpition, price, etc
csv
cleaning
Products
csv
Recommendation service (Java Spring Boot)
Purchases
Once a day: increment
Purchase (increment)
clientHashId, productid, amount, itemsCount, e
csv
cleaning, consolidating
Purchases
csv
import and swap
Clients
Once a day: increment/ changes
Clients (increment/changes)
clientHashId, sex, geo, age, langu
etc
csv
cleaning, consolidating
Clients
csv
Model
csv
denormalizaion
Training data
csv
Training
Recommender engine POC scope (Python)

Figure 1. *A drawing of the service architecture.*

Figure 1 shows the architecture of the service and recommender engine role in it. The service depends on external systems from tenants, either direct access to DB or an interface, that will be able to provide 4 types of data.

- Purchases - main data source, consists of primarily unique client hash id and product unique id. Additional information might include total billing amount, bought item count, date and time of purchase, place or method of purchase. Due to a potentially very big amount of purchase data, all purchases will be fetched once during tenant onboarding, after that new purchases will be fetched once a day as an increment and then joined with the previous history.
- Products - secondary data source, used to enable conditional recommendations, for example per tag/category, price, availability, delivery time and others. This data could later be used in deep neural network training for a better quality of predictions. Products will be fetched once fully during onboarding and then once per day as an increment with changes per tenant.
- Clients - secondary data source, used to make conditional and more personalised recommendations, e.g. by constraining language or availability by location. This data could later be used in deep neural network training for a better quality of predictions. Clients will be fetched once fully during onboarding and then once per day as an increment with changes per tenant.
- Miscellaneous (settings, exceptions, different lists) - optional data source that could be used to improve the tenant experience. This data could include handmade promotional lists that would have priority over other items, some specific exceptions and constraints that should be taken into account during recommendation generation (e.g. always recommend at least 1 item from the global top 10 most popular items) and other settings. Settings will be fetched once per day as a full copy per tenant.

The idea is to combine the data from different tenants so that the algorithm can observe as many examples as possible. This typically results in a better quality of recommendations. Small tenants benefit the most from such a model since they might not have enough data to build a recommendation engine on their own to start with. Such a system also benefits big tenants, since we can combine purchases from all tenants, resulting in a better approximation of client tastes than from one source.

To connect a user from one source to the same user from another source, we introduce a consistent identifier between all sources - unique client hash id. This hash id is based on client email, which is hopefully the same for a client for different sources. The reason for using a hash instead of email directly is legal concerns.

After all the information was received, it will have to go through the cleaning and consolidation process. This process includes removing erroneous entries, removing non-typical clients (such as bots and resellers), and solving merging conflicts. After that, every type of data will create a CSV file for easier managing and storing. The next step is data denormalization, during which we combine all the data points to create train data with positive examples according to the used algorithm. After that, we augment the train set with negative examples if that is needed by the algorithm. We then use the completed training set to train and optimise an algorithm, after which we export the model to CSV or any other suitable format. The last step is importing the new model into the REST service and swapping the currently working model with the most recent model. Then, the tenant is free to use the service by calling it with client hash id, optionally constraining recommended goods by categories, price, availability and others.

Since creating recommendations for a client could take up to a few seconds, some optimisations could be implemented to improve client experience. The first one is to start calculating recommendations right after a client entered his login, not waiting for the password and login attempt. Then, after saving all the ratings for all the items to cache, the client can get the recommendations within 100 milliseconds. Minor optimisation could be based on an assumption that if a client browses some specific category, e.g. 'Medicine', he is likely to continue doing that as the next request. If only a certain amount of items can be displayed on a page at once, then we could prepare recommendations for the next page, not waiting for an actual request.

# 3 Related work

## 3.1 Recommender system building steps

Process of creation recommender system might be roughly divided into 6 steps:

- **Data mining** - during this stage you identify raw data that contains features or patterns that you wish to learn. Raw data can be combined with different data sources to enrich it.
- **Data technical analysis** - it should be determined whether the data has proper format and match all the constraints. For example, every purchase history entry for recommendations needs to contain a unique field for a distinct buyer and a unique field for a distinct item.
- **Data cleaning** - during this stage data is being prepared. All erroneous entries should be deleted. All data that is not produced by the target (such as reseller entities, rather than human purchasing) should be removed.
- **Choosing an algorithm** - during this stage algorithm is chosen that would fulfil your requirements. Requirements could include the size of input data, data structure, training speed, prediction speed, prediction accuracy and others.
- **Optimising an algorithm** - after choosing an algorithm or a few that match your requirements, you need to optimise them accordingly to get the best results. Optimisation might include hyperparameter changes (parameters used to control the model learning process), input data enrichment or simplification, deep neural network model architectural changes.
- **Building a service** - if you intend to bring your recommender system into live, you need to build a service. Depending on your needs you might want to include such features as automatic model retrain, live model optimisation and others.

## 3.2 Recommender system generations

Recommender systems could be divided into three generations:

### 3.2.1 1st generation

- **Knowledge-based** - recommendations are based on explicit information given by the client about himself, usually in form of questionnaires on registration. Every item in the service is also described as a direct or derivative of information given by clients. The recommendation is defined as the best match between user explicit preferences and item properties. An example is a used vehicle selling platform, where cars have some properties such as mark, model, year, price and mileage, while the client might constraint his search using the same metrics to find the best fit. [1]
- **Memory based collaborative filtering** - methods include similarities between items and clients and their explicit and/or implicit feedback to give recommendations. A common example is user-item CF: client A bought X, Y, Z and rated them well, client B bought X, Y and rated them well, thus clients A and B are similar and B is likely to enjoy Z too. [2]
- **Hybrid** - combination of previous methods and different variations.

### 3.2.2 2nd generation

- **Matrix factorisation** - idea is based on the mathematical idea of matrix factorisation into smaller matrices so that on multiplication of resulting matrices you get the best approximation of starting matrix. In this case, matrix factorisation is usually performed on a large and sparse user-item rating matrix.
- **Personality based** - idea is to create a psychological profile of a person using implicit and/or explicit information. In contrast to the knowledge-based model, the personality model represents person trends (e.g. Five Factor Model [3]) and matches people with similar trends, like CF. [4] [5]
- **Web usage mining based** - main data input point is website usage, such as click history, time spent, adding item to cart, wish list. Prediction is made based on these observations from all users (e.g. currently using the website) [6] [7]

### 3.2.3 3rd generation

- **Deep neural networks** - models based on neural networks. These are capable of recognising complex underlying patterns in the data. Deep neural networks can include a different kind of data into its decision making, rather than just user-item pairs. [8]

## 3.3 Examples

### 3.3.1 Amazon

Amazon recommends items to buy on their respective online shops, videos and films to see on Amazon Prime Video. In online shops, Amazon has ' Frequently bought together' section, which is likely to be learned from user purchases and 'Customers who viewed this item also viewed', which is most likely web data mining technique. For recommendations, they primarily use collaborative filtering approaches with matrix factorization. [9]

### 3.3.2 Spotify

Spotify recommends songs on different occasions. It recommends a few radios with everlasting groups of songs (e.g. by genre/style), new releases that are similar to the user's tastes, or songs that are similar to some particular song or group. According to some sources, Spotify uses a combination of content-based, collaborative filtering and neural network strategies to achieve its results. [10]

### 3.3.3 Ebay

Ebay, similar to Amazon, has a net of online shop platforms, that is capable of recommending items that clients are likely to be interested in and items that are similar to some specific item. Like Amazon, it prioritises items on the web page, as well as having 'Similar items' section for each item. There is little information on how eBay recommender system is built.

### 3.3.4 Youtube

Youtube recommends videos and channels for its viewers to see. Youtube recommender system is designed to maximize watching time and uses a complex combination of encoders, decoders and deep neural networks to achieve its results. It uses information such as user average watch time, language, time since last watch, location and other. [11]

# 4 Technologies

Development of the recommender model is done using Python 3.9[12]. Important libraries that were used in the development are:

- sklearn[13] - core library for scikit-surprise, it provides different clustering, classification and regression algorithms that could be used in supervised learning and unsupervised learning models. In contrast to scikit-surprise, sklearn methods are more general, which results in more manual configuration.
- scikit-surprise[14] - Python library for planning, building, evaluating and analysing recommender systems that deal with explicit rating data. The library includes built-in datasets, as well as allowing to use your custom datasets. It also includes multiple ready-to-use prediction algorithms, such as baseline algorithms, matrix factorization-based algorithms and many others. For collaborative filtering algorithms, it includes various similarity measures, such as cosine similarity and Pearson distance. For the ease of optimisation, the library includes algorithms such as GridSearchCV and RandomizedSearchCV for automatic optimisation.
- NumPy[15] - is a core library for scientific computing in Python. The library provides an implementation for complex data structures like matrices, masked arrays and implementing an assortment of fast methods to manipulate the data such as mathematical, logical, sorting, selecting operations and many others.
- Pandas[16] - alongside NumPy provides an assortment of flexible, fast and powerful ways of data analysis and data manipulation. It allows to easily work runtime with common data storing formats such as CSV, JSON, Excel and others. This includes selecting, sorting, filtering, indexing, reshaping, deleting, augmenting, joining and many other ways to process the data.

# 5 Data

## 5.1 Overview

This chapter will introduce the dataset that was used during development.

Data was provided by Apollo in the form of CSV and consists of purchase history from 2014 to 2020 over all departments.

Data consists of:

- 24 million purchases, of which:
  - 6 million ticket sales
  - 4 million book sales
- 14 million unique client-item pairs
- 500 thousand clients
- 120 thousand unique goods
- 25 unique item groups
- Sparsity of 0.0002% (means on average a person has bought about 0.0002% of all items)

We intentionally asked Apollo for as much data as possible with the idea that we might need it later. You could also see that some fields only relate to cinema ticket sales. We asked Apollo to join all sales (book shops, stationery, cinema tickets) for ease of processing. Otherwise, joining tables without an actual indexed database is very slow.

Table 1 shows the structure of the data provided by the Apollo.

Table 1. *A table with data fields*

| **Field** | **Type** |
|---|---|
| InvoiceNumber | Integer |
| InvoiceLineNumber | Integer |
| TheatreID | Integer |
| TheatreName | String |
| SalesPointID | Integer |
| SalesPointName | String |
| ProductCode | String |
| ProductName | String |
| Quantity | String |
| NetSum | String |
| GrossSum | String |
| ProductRevenueGroupID | Integer |
| ProductRevenueGroupName | String |
| CustomerUniqueID | String |
| BirthDay | String |
| Gender | String |
| ZIP | String |
| dttmTransaction | String |
| TicketID | Integer |

## 5.2 Anonymity

Before providing this data, Apollo anonymised all records by replacing CustomerUniqueID value with newly generated value and storing new value alongside the initial value. This anonymises data for us and leaves an opportunity for Apollo to match our recommendations with real people.

## 5.3 Preparation

Before we could use this data to train our models we should do some preparation.

There are two distinct types of data that we know about users

Explicit data - information that is provided intentionally and is not required for the normal use of the service. Examples of explicit data are surveys, likes and dislikes, birthdays.

Implicit data - information that is not provided intentionally, but behaviourally and gathered as a result of the normal use of the service. Examples of implicit data are search history, clicks, watch time, geographic location.

The only kind of data provided by Apollo is implicit, which is purchase history. For our POC purposes, we do not need most of given data fields. As per scikit-surprise library specifications, we need to provide a table that consists of 3 columns:

- Id, that is unique for a client
- Id, that is unique for an item
- Rating

Due to the export method, which is to CSV with '\t' tab delimiter, rows that have tab is their column values (which usually does not happen) are corrupted during the import process. To solve this we find all rows that after splitting on '\t' tab have more columns than usual and remove them from the dataset. It only accounts for 1411 rows, which is 0.006% of the total dataset, which is negligible. Other problem is that 'ProductCode' which is unique for one source (e.g. book store) is not unique across other sources (e.g. tickets). Before we could represent our data to the needed format, we fix 'ProductCode' by combining it with 'ProductRevenueGroupID', so the new field 'ServiceCode' is a unique identifier across all data.

Now, we remove all fields but 'ServiceCode' and 'CustomerUniqueID'. As you might have noticed, we do not have a rating column in our data. So, we just create a new column 'rating' and set its value to 1 to represent the fact that there was a purchase. 0 will represent that there was no and will be no purchase between the given pair. Anything in between will represent how certain are we that the actual value is the one to the closest integer. We will explain why that is required later.

We also separate the dataset into a few datasets based on 'ProductRevenueGroupID' which corresponds to different item types, so that we could experiment with different combinations and find which one has the best result.

Then, we further narrow down the book dataset by removing subgroups that do not correspond to literature, such as maps, postcards, stationery, albums. Now, we are left with 4 million rows for books and 6 million rows for films.

## 5.4 Anomalies

The rule of thumb for preparing a machine learning dataset is that it should represent actual human behaviour. In our case, it means removing any purchase history that was done by bots, resellers and any other organisations that do not have any pattern. Apollo was able to provide us with 'CustomerUniqueID's that correspond to such organisations and resellers. They account for 2500 rows, which is 0.01% of the total dataset. This is negligible and we just delete it from the dataset (on average, a user has bought 0.0002% of all items).

## 5.5 Train set

Before we could use prepared data in algorithms, we still have some way to go. We have a table with positive examples, which for us is a fact of purchase. Unfortunately, that is not enough, because if we were to use only positive examples to train our model, the model will be very biased towards positive results (result towards 1), since the algorithm has never seen any negative examples (with rating 0). Adding negative examples represents buyer behaviour since only a small amount percentage of items will ever be bought by a specific user.

To create negative examples we randomly create user-item pairs that do not appear in positive examples and set its rating to 0 (Algorithm 1). That is a bold assumption that randomly chosen items will never be bought by the randomly selected user, but it works quite well in practice.

```
Algorithm 1: Negative example generating algorithm
Input    : Items $I$, users $U$, user-item pairs $J$, negative examples to add $n$
Output   : Set of negative examples
foreach $u_i \in U$ do
    repeat
        get random $i_j \in I$;
        while $(u_i, i_j) \in J$ do
            get another $i_j \in I$;
        add $(u_i, i_j)$ to negative examples;
    until $n$;
```

Due to the lack of indexed DB for lookups and the amount of created negative examples, the algorithm takes a lot of time even on modern high-frequency CPUs. To optimize it, we split this task to take all threads (Algorithm 2).

```
Algorithm 2: Threaded negative example generating algorithm
Input    : Items $I$, users $U$, user-item pairs $J$, negative examples to add $n$
Output   : Set of negative examples
split $U$ into ($k$=CPU thread count) parts
foreach $U_k \in U$ do
    use Algorithm 1 with $U_k$ user set in thread
join $k$ threads and user-item sets
```

# 6 Algorithms and methods

Table 2. *Notations used in the formulas*

| Notation | Explanation |
|---:|---|
| $Un$ | uniform random distribution |
| $U$ | set of all users |
| $u \in U$ | single user |
| $I$ | set of all items |
| $I_u$ | set of items bought by user $u$ |
| $i \in I$ | single item |
| $r_{ui}$ | actual rating for item $i \in I$ and user $u \in U$ |
| $\hat{r}_{ui}$ | predicted rating for item $i \in I$ and user $u \in U$ |
| $R_{train}$ | train set |
| $r_{ui} \in R_{train}$ | single training example rating for item $i \in I$ and user $u \in U$ |
| $\hat{R}$ | set of predictions |
| $\mu$ | global rating mean average |
| $\lambda$ | regularization term |
| $\gamma$ | learning rate |
| $b_u$ | user biases |
| $b_i$ | item biases |
| $q_i$ | learned item parameters |
| $p_u$ | learned user parameters |

Table 2 lists all the notations used in the mathematical formulas in this chapter.

## 6.1 Baselines

*RandomPredictor* - the prediction $\hat{r}_{ui}$ is uniform random

$$\hat{r}_{ui} = Un([0,1]) \tag{6.1}$$

*NormalPredictor* - the prediction $\hat{r}_{ui}$ is based on the normal distribution $\mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$ of the training set. [17]

$$\hat{\mu} = \frac{1}{|R_{train}|} \sum_{r_{ui} \in R_{train}} r_{ui} \tag{6.2}$$

$$\hat{\sigma} = \sqrt{\sum_{r_{ui} \in R_{train}} \frac{(r_{ui} - \hat{\mu})^2}{|R_{train}|}} \tag{6.3}$$

*BaselineOnly* - the prediction $\hat{r}_{ui}$ is a baseline estimate. [18]

$$\hat{r}_{ui} = b_{ui} = \mu + b_u + b_i \tag{6.4}$$

## 6.2 Algorithms

This chapter will introduce you to the algorithms that were most investigated during the development of the project

### 6.2.1 Memory-based user-item collaborative filtering

First, we decided to start with memory-based user-item collaborative filtering. The idea behind this algorithm is to calculate the score for an item-user pair by finding similarity between other users, multiplying similarity by corresponding user rating and then averaging the result. This represents the idea that similar people tend to like similar items.

For that, we have to define what does similarity between two users means. The go-to method, in this case, is to define similarity between two users as a cosine distance between users purchase lists.

$$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{ab}}{\|\mathbf{a}\|\|\mathbf{b}\|} = \frac{\sum_{i=1}^{n} \mathbf{a}_i \mathbf{b}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{a}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{b}_i)^2}} \tag{6.5}$$

To find similarity between each user we have to create a user-item matrix first by pivoting the user-item-rating table into the user as columns, items as rows and rating as cell values. This will create an extremely sparse n x m matrix, where n is user amount and m is item amount.

$$\begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 1 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{6.6}$$

Now, we find similarity between each user pair. Resulting matrix might look like:

$$\begin{bmatrix} 1 & 0.4 & 0.23 & \cdots & 0 \\ 0.4 & 1 & 0 & \cdots & 1 \\ 0.23 & 0 & 1 & \cdots & 0.15 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & 0.15 & \cdots & 1 \end{bmatrix} \tag{6.7}$$

The prediction $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \frac{1}{\sum_{u' \in U} cos(u, u')} \sum_{u' \in U} |cos(u, u')| r_{u'i} \tag{6.8}$$

Such an algorithm has a few advantages:

- It is fast to implement
- It is easy to understand and explain to a non-tech person
- It is flexible and does not require any additional configuration

Unfortunately, it also comes with disadvantages:

- The algorithm requires to keep in memory n x m and n x n matrices, which is not feasible for bigger numbers of users and items
- Calculation speed for similarity matrix and pivoting table is slow

Since we can not use all data even from a single group, we could not go forward with this algorithm.

### 6.2.2 SVD

The second algorithm we did not implement ourselves but instead used a pre-made library one from scikit-surprise. SVD solves the problem of enormous matrices in memory by learning features and saving them in separate smaller matrices. It uses the mathematical idea of matrix factorisation. [19]

The prediction $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \tag{6.9}$$

To estimate the unknown, we minimize the following expression:

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left( b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2 \right) \tag{6.10}$$

Minimization is defined as stochastic gradient descent:

$$b_u \leftarrow b_u + \gamma(e_{ui} - \lambda b_u) \tag{6.11}$$

$$b_i \leftarrow b_i + \gamma(e_{ui} - \lambda b_i) \tag{6.12}$$

$$p_u \leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u) \tag{6.13}$$

$$q_i \leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i) \tag{6.14}$$

where $e_{ui} = r_{ui} - \hat{r}_{ui}$

### 6.2.3 SVD++

SVD++ is a slightly improved version of SVD, making use of implicit information. [20]

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left( p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right) \tag{6.15}$$

There are some advantages to using SVD based algorithms, such as:

- No limit on input data size
- Pre-made
- Scores the best

Thus, not without disadvantages:

- Train speed is slow
- Requires slow negative example preparation
- The algorithm is hard to understand and explain, making it harder to debug

the results
- It requires hyperparameter tuning

## 6.3 Validation

This chapter will introduce you to the validation metrics that were used in the project

### 6.3.1 MAE

MAE - Mean Average Error, calculates the average error of predictions against actual values. [21]

$$\mathrm{MAE} = \frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} |r_{ui} - \hat{r}_{ui}| \tag{6.16}$$

### 6.3.2 RMSE

RMSE - Root Mean Square Error, punishes for bigger deviation. [22]

$$\mathrm{RMSE} = \sqrt{\frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} (r_{ui} - \hat{r}_{ui})^2.} \tag{6.17}$$

### 6.3.3 Manual

Manual - manually selecting a few users with distinct preferences (e.g. animation films, fictional films or fairy tales for books) and evaluating their predictions.

### 6.3.4 A/B testing

A/B testing - we predict the top 10 for a selection of Apollo clients, put it against the baseline algorithm and ask them to choose which one of the lists is more personalised, without telling them which one is which.

## 6.4 Training

Since any change of hyperparameters leads to a generally unpredicted result, there is no straight way to optimise your algorithm. In order to get the best result for a given dataset, we apply an optimisation algorithm, which searches for the best hyperparameter combination within given constraints.

RandomizedSearchCV - implemented by sklearn, randomly chooses hyperparameter combinations, divides the dataset into splits (to avoid overfit and bias) and then trains such models. Then it repeats the process n times and then chooses the model with the best results. [23]

# 7 Results

Algorithms were trained on 1 positive to 4 negative example ratio.

On the dataset with only books, SVD++ shows the best MAE result of 0.135 and the best RMSE result of 0.213. For book model it is harder to test manually, since there are no distinct tastes. Results of all algorithms are presented in Table 3.

Table 3. *Test results for only books*

| **Algorithm** | **MAE** | **RMSE** |
|---|---|---|
| Random | 0.500 | 0.577 |
| Normal | 0.383 | 0.514 |
| Baseline | 0.197 | 0.299 |
| SVD | 0.137 | 0.215 |
| SVD++ | **0.135** | **0.213** |

On the dataset with only films, SVD++ shows the best MAE result of 0.195 and RMSE of 0.311. The Baseline has the best RMSE result of 0.302, slightly better than SVD++ and SVD. Even though the Baseline is not far behind result-wise, during manual testing Baseline predictions seem to be very general and not personalised. SVD++ predictions seem to have the most sense, such as animation films or Russian films (see Table 8 and Table 9). Results of all algorithms are presented in Table 4.

Table 4. *Test results for only films*

| **Algorithm** | **MAE** | **RMSE** |
|---|---|---|
| Random | 0.500 | 0.577 |
| Normal | 0.374 | 0.507 |
| Baseline | 0.203 | **0.302** |
| SVD | 0.213 | 0.305 |
| SVD++ | **0.195** | 0.311 |

On the dataset with films and books, SVD++ shows the best MAE result of 0.117 and the best RMSE result of 0.204. This model seems to be very biased towards films, likely due to a higher volume of film ticket purchases. On average, there are no books recommended in the top 30 and only films present. What is interesting is that it performs better in automatic testing than books and films individually. Manual examination showed that this model is very predictable, meaning if a person watched a few Russian films mostly Russian films will be recommended. Results of all algorithms are presented in Table 5.

Table 5. *Test results for books and films*

| **Algorithm** | **MAE** | **RMSE** |
|---|---|---|
| Random | 0.500 | 0.577 |
| Normal | 0.371 | 0.503 |
| Baseline | 0.129 | 0.229 |
| SVD | 0.124 | 0.209 |
| SVD++ | **0.117** | **0.204** |

### 7.0.1 Cold start problem

Matrix factorisation algorithms rely on similarities between users and items based on purchase habits. Item cold start problem happens when the system is introduced a completely new item, meaning it has no purchases. Because of that, that item will never get recommended, which is bad marketing. There are a few solutions:

- Randomly include new items into recommendations for a period of time, for example until n purchases were made
- Ask a specialist / people to choose items that are the most similar to the new item, set its parameters as the average of similar items
- Just ignore it, unless all business channels use recommender system, it will still get purchased, for example in a physical shop

The new client cold start problem is similar, when a new client is introduced to the system, the client does not have any purchases. Because of that, the system is unable to make personalised predictions. Solutions to this include:

- Recommend the most popular items overall until the person makes a purchase
- Ask a person to select items that he likes and save them as if he bought them

### 7.0.2 Exploding prediction problem

As table 7 shows, if a user has a lot of different purchases, you might see that problem. It could be explained as follows: if a person buys a lot of different items, then he is similar to many other people, so he must like a lot of other things. Due to how prediction is calculated for SVD and SVD++ this is a common problem for this kind of algorithms. The problem is, if you just clip the prediction value (e.g. between 0 and 1), like most of the algorithms do, predictions that all have rating 1 will not have accurate order. If you do not need accurate prediction numbers and just need top n items to recommend, you can just disable rating clipping. Otherwise, you can use normalisation techniques, e.g. dividing all user predictions by the highest prediction value, so that rating relation stays consistent.

## 7.1 Prediction examples

In section 7.1.1 and section 7.1.2 you can see predictions made by SVD++ for a few manually selected users with distinct features. Column *Place* indicates the position of a specific film for the specific person in the predictions. A lower place value means a higher chance that the person might be interested in a given item. *Name* field is a name of predicted item. *Rating* shows the predicted rating value for a given user-item pair, higher rating indicates a higher chance that the person might be interested in a given item. *Bought* field marks whether a given user-item pair existed in the train set with a positive value. So, *Bought* field has 'Yes' if the user has bought a given item and 'No' otherwise.

Table 6 shows predictions for a client with 10 bought books. It is hard to evaluate the taste, but nothing looks too random. The algorithm correctly identified 3 bought books in predictions.

Table 7 shows predictions for a user with 30+ bought books. As subsection 7.0.2 describes, it introduces 'Exploding prediction problem'. Predictions seem to be fairly random for that user, 4 bought books appeared in the prediction.

Table 8 shows predictions for a user with distinct taste in animated and action films. Predictions appear very logical, many bought films appeared in the prediction.

Table 9 shows predictions for a user with a tendency to watch Russian films. Predictions include 3 Russian films, but not too high in the list. The algorithm could correctly identify a few watched films.

### 7.1.1 Book predictions

Table 6. *Only book SVD++ results for common client*

| **Place** | **Name** | **Rating** | **Bought** |
|---|---|---|---|
| 1 | Kes see Mallukas veel on? | 1 | No |
| 2 | Esmalt küsi "Miks". Kuidas edukad inimesed ennast ja teisi tegudele inspireerivad | 0.9597 | Yes |
| 3 | Tobias ja teine B | 0.9379 | No |
| 4 | Naine. Otse ja ausalt. Marju Karin | 0.9362 | Yes |
| 5 | Eesti looduse kannatuste aastad | 0.928 | No |
| 6 | Emotsioonid. Inimkonna suurim sõltuvus | 0.9006 | No |
| 7 | Rikkaks saamise Õpik. Teine trükk | 0.8968 | No |
| 8 | Oskar ja asjad | 0.8958 | No |
| 9 | Homo Deus. Homse lühiajalugu (PK) | 0.8862 | No |
| 10 | Võtku homme mind või saatan | 0.8825 | No |
| 11 | Rikkaks saamise Õpik. Kolmas täiendatud trükk | 0.8749 | No |
| 12 | Laulud või nii | 0.8725 | No |
| 13 | Loomise õpetus II | 0.8718 | No |
| 14 | Eestlase käsiraamat. 100 asja, mida õige eestlane teeb | 0.8624 | No |
| 15 | Aktsiatega rikkaks saamise õpik | 0.8501 | No |
| 16 | Kus laulavad langustid | 0.849 | No |
| 17 | Teekond iseendani | 0.8464 | No |
| 18 | Ratsionaalne emotsionaalsus | 0.8425 | Yes |
| 19 | Eesti vanaemade lood ja salatarkused | 0.8395 | No |
| 20 | Eesti ümberlõikaja | 0.8393 | No |
| 21 | Minu aktiivne beebi | 0.8291 | No |
| 22 | Minu esimene elu | 0.8256 | No |
| 23 | Kuidas armastada naist? | 0.8256 | No |
| 24 | Lindvistika ehk metsa see lingvistika | 0.8204 | No |
| 25 | Magusaga kaalust alla | 0.8167 | No |
| 26 | ALEMUS! Minu Kennedy | 0.8136 | No |
| 27 | Treeningpäevik. Sinu trenn, toit ja tervis | 0.8101 | No |
| 28 | JÕUL2019 Jamie Oliveri taimetoidud | 0.8072 | No |
| 29 | Leiutajateküla Lotte. KK | 0.8053 | No |
| 30 | Patsient A | 0.8017 | No |

Table 7. *Only book SVD++ results for 30+ books bought user*

| **Place** | **Name** | **Rating** | **Bought** |
|---|---|---|---|
| 1 | Tänavalt troonile | 1 | No |
| 2 | Aastaga haigustest priiks. Tervenduskalender 2018 | 1 | No |
| 3 | XXVIII Pauksoni astroloogiline abimees 2017 B | 1 | No |
| 4 | Eesti vanaemade lood ja salatarkused | 1 | No |
| 5 | Tobias ja teine B | 1 | No |
| 6 | MEES Triloogia | 1 | Yes |
| 7 | Kus ma olen ja kuidas sina võid palju kaugemale jõuda | 1 | No |
| 8 | Mihkel Raud - ISA | 1 | No |
| 9 | Tervis Jumala apteegist | 1 | No |
| 10 | Mees 3. Elu kutse | 1 | No |
| 11 | Nastja maagiline 2018 | 1 | No |
| 12 | Magusaga kaalust alla | 1 | No |
| 13 | Eestlase käsiraamat. 100 asja, mida õige eestlane teeb | 1 | No |
| 14 | Taroskoop 2020 | 1 | No |
| 15 | Kuidas alustada investeerimisega | 1 | No |
| 16 | Eesti 100 torti. Meie tordimeistrite parimad tordid, koogid | 1 | No |
| 17 | Hea une teejuht | 1 | No |
| 18 | Rusikad | 1 | No |
| 19 | Reketiga tüdruk. Kaia Kanepi teekond Ameerika mägedel | 1 | No |
| 20 | Apollo kõige suurem ristsõnaraamat | 1 | No |
| 21 | Rikkaks saamise Õpik. Kolmas täiendatud trükk | 1 | No |
| 22 | Kuidas võita sõpru ja mõjutada inimesi | 1 | No |
| 23 | Kes tappis Urmas Oti? | 1 | Yes |
| 24 | JÕUL2019 Tüdrukune | 1 | No |
| 25 | Kinnisvaraga rikkaks saamise õpik | 1 | No |
| 26 | XXX Pauksoni astroloogiline abimees 2019 | 1 | Yes |
| 27 | Rikkaks saamise Õpik. Teine trükk | 1 | No |
| 28 | Keskööpäike. Videviku saaga V raamat | 1 | No |
| 29 | Lõvi. Nastja tähtkujuraamat | 1 | Yes |
| 30 | Ilusad suured tüdrukud A | 1 | No |

### 7.1.2 Film predictions

Table 8. *Only films SVD++ results for animation fan client*

| **Place** | **Name** | **Rating** | **Bought** |
|---|---|---|---|
| 1 | Tõde ja õigus | 0.93 | Yes |
| 2 | Tenet | 0.9124 | No |
| 3 | Talve | 0.9116 | Yes |
| 4 | Frozen 2 | 0.9036 | Yes |
| 5 | Eia jõulud Tondikakul | 0.8658 | No |
| 6 | Bohemian Rhapsody | 0.8515 | No |
| 7 | The Lion King | 0.8474 | No |
| 8 | Klassikokkutulek 2: Pulmad ja matused | 0.8211 | Yes |
| 9 | The Secret Life of Pets 2 | 0.8206 | No |
| 10 | Klassikokkutulek 3: Ristiisad | 0.812 | Yes |
| 11 | Joker | 0.8015 | No |
| 12 | Incredibles 2 | 0.7963 | No |
| 13 | Despicable Me 3 | 0.7942 | Yes |
| 14 | Klassikokkutulek | 0.7685 | Yes |
| 15 | The Grinch | 0.7684 | Yes |
| 16 | Trolls World Tour | 0.7626 | No |
| 17 | The Secret Life of Pets | 0.7482 | No |
| 18 | Fast & Furious: Hobbs & Shaw | 0.7416 | No |
| 19 | Svingerid | 0.7414 | Yes |
| 20 | How to Train Your Dragon: The Hidden World | 0.7407 | No |
| 21 | Ice Age 5 | 0.7383 | Yes |
| 22 | Sipsik | 0.7377 | No |
| 23 | Sonic the Hedgehog | 0.7362 | Yes |
| 24 | Vanamehe film | 0.7355 | No |
| 25 | Abominable | 0.7316 | No |
| 26 | Pirates of the Caribbean: Salazar's Revenge | 0.727 | No |
| 27 | O2 | 0.715 | Yes |
| 28 | The Boss Baby | 0.7104 | No |
| 29 | Hotel Transylvania 3 | 0.7087 | No |
| 30 | Mamma Mia! Here We Go Again | 0.7085 | No |

Table 9. *Only films SVD++ results for russian client*

| **Place** | **Name** | **Rating** | **Bought** |
|---|---|---|---|
| 1 | Tõde ja õigus | 1 | No |
| 2 | Talve | 1 | No |
| 3 | Joker | 1 | Yes |
| 4 | Bohemian Rhapsody | 1 | No |
| 5 | Tenet | 1 | Yes |
| 6 | O2 | 1 | No |
| 7 | Klassikokkutulek | 0.9884 | Yes |
| 8 | Fred Jüssi. Olemise ilu | 0.9841 | No |
| 9 | Frozen 2 | 0.9693 | Yes |
| 10 | Klassikokkutulek 3: Ristiisad | 0.9672 | Yes |
| 11 | Klassikokkutulek 2: Pulmad ja matused | 0.9146 | No |
| 12 | Eia jõulud Tondikakul | 0.9038 | No |
| 13 | Vanamehe film | 0.8961 | No |
| 14 | Холоп | 0.8876 | No |
| 15 | Sipsik | 0.8619 | No |
| 16 | Seltsimees Laps | 0.8582 | No |
| 17 | The Lion King | 0.8356 | No |
| 18 | Svingerid | 0.8307 | No |
| 19 | Ott Tänak: The movie | 0.8158 | No |
| 20 | Полицейский с Рублевки. Новогодний беспредел 2 | 0.7947 | No |
| 21 | Лед 2 | 0.7779 | No |
| 22 | Once Upon a Time in Hollywood | 0.7768 | No |
| 23 | Mamma Mia! Here We Go Again | 0.7653 | No |
| 24 | Pilvede all. Neljas õde. | 0.7613 | No |
| 25 | Экипаж | 0.7603 | No |
| 26 | A Star is Born | 0.759 | No |
| 27 | Trolls World Tour | 0.7569 | No |
| 28 | Притяжение 2 | 0.7559 | No |
| 29 | The Secret Life of Pets 2 | 0.7494 | No |
| 30 | Стрельцов | 0.7431 | No |

# 8 Conclusion

The goal of this thesis was to determine the possibility of using Apollo purchase history data to extract behavioural patterns and create recommender system. Recommender system should be able to make meaningful predictions that would interest clients.

To achieve this goal we had to analyse the data and conclude that the data meets the requirements. We cleared the data from erroneous entries and removed all non-human made purchases. We tested *Random*, *Normal* and *Baseline* baseline algorithms, memory-based user-item collaborative filtering , as well as matrix factorization based algorithms *SVD* and *SVD++*. We optimised given algorithms and tested them against baseline algorithms. We made predictions for a few people with distinct tastes and manually examined the results.

As a result of the work we found out that *SVD++* algorithm is the best fit for our data and chosen metric, scoring 0.135 MAE for data set with books only, 0.195 MAE for data set with films only and 0.117 MAE for data set with films and books combined. During manual examination algorithms based trained on combined films and books data set was found to be very biased towards films and was not a valuable addition. We encountered Cold start problem and Exploding prediction problem and proposed a few solutions. The algorithm is decided to be good enough to start building minimal viable product for live Apollo systems.

Even though we have created an algorithm that is capable of making meaningful predictions that exceed baseline results, we are positive that it is possible to improve the results by using more implicit information such as item tags/categories, language, price, year of creation with the use of deep neural networks. Unfortunately, Apollo did not manage to perform A/B testing in a given time frame of writing this thesis.

# Bibliography

[1] Robin Burke. “Knowledge-Based Recommender Systems”. In: *Encyclopedia of library and information systems* 69 (May 2000).

[2] P. Aditya, Indra Budi, and Qorib Munajat. “A comparative analysis of memory-based and model-based collaborative filtering on the implementation of recommender system for E-commerce in Indonesia: A case study PT X”. In: Oct. 2016, pp. 303–308. DOI: `10.1109/ICACSIS.2016.7872755`.

[3] Shahpar Yakhchi et al. *Enabling the Analysis of Personality Aspects in Recommender Systems*. [Accessed: 27-04-2021]. 2020.

[4] Marko Tkalčič et al. “Personality based user similarity measure for a collaborative recommender system”. In: (Jan. 2009).

[5] Alexandra Roshchina, John Cardiff, and Paolo Rosso. “TWIN: Personality-based Intelligent Recommender System”. In: *Journal of Intelligent & Fuzzy Systems* 28 (June 2015), pp. 2059–2071. DOI: `10.3233/IFS-141484`.

[6] Prajyoti Lopes and Bidisha Roy. “Dynamic Recommendation System Using Web Usage Mining for E-commerce Users”. In: *Procedia Computer Science* 45 (2015). International Conference on Advanced Computing Technologies and Applications (ICACTA), pp. 60–69. ISSN: 1877-0509. DOI: `https://doi.org/10.1016/j.procs.2015.03.086`. URL: `https://www.sciencedirect.com/science/article/pii/S1877050915003221`.

[7] Ya Luo. “An Analysis of Web Mining-based Recommender Systems for E-commerce”. In: Atlantis Press, 2012/08, pp. 167–170. ISBN: 978-94-91216-00-8. DOI: `https://doi.org/10.2991/iccasm.2012.43`. URL: `https://doi.org/10.2991/iccasm.2012.43`.

[8] Libo Zhang et al. “A Recommendation Model Based on Deep Neural Network”. In: *IEEE Access* 6 (2018), pp. 9454–9463. DOI: `10.1109/ACCESS.2018.2789866`.

[9] Larry Hardesty. *The history of Amazon’s recommendation algorithm*. [Accessed: 19-04-2021]. URL: `https://www.amazon.science/the-history-of-amazons-recommendation-algorithm`.

[10] Proma Huq. *Music To My Ears: De-Blackboxing Spotify's Recommendation Engine.* [Accessed: 19-04-2021]. URL: https://blogs.commons.georgetown.edu/cctp-607-spring2019/2019/05/06/music-to-my-ears-de-blackboxing-spotifys-recommendation-algorithm/.

[11] Covington Paul, Adams Jay, and Sargin Emre. *Deep Neural Networks for YouTube Recommendations.* [Accessed: 19-04-2021]. URL: https://static.googleusercontent.com/media/research.google.com/ru//pubs/archive/45530.pdf.

[12] *Python.* [Accessed: 19-04-2021]. URL: https://www.python.org/.

[13] *Scikit-learn.* [Accessed: 19-04-2021]. URL: https://scikit-learn.org/stable/.

[14] *Surprise.* [Accessed: 19-04-2021]. URL: http://surpriselib.com/.

[15] *NumPy.* [Accessed: 19-04-2021]. URL: https://numpy.org/.

[16] *Pandas.* [Accessed: 19-04-2021]. URL: https://pandas.pydata.org/.

[17] *Surprise NormalPredictor.* [Accessed: 19-04-2021]. URL: https://surprise.readthedocs.io/en/stable/basic_algorithms.html#surprise.prediction_algorithms.random_pred.NormalPredictor.

[18] *Surprise BaselineOnly.* [Accessed: 19-04-2021]. URL: https://surprise.readthedocs.io/en/stable/basic_algorithms.html#surprise.prediction_algorithms.baseline_only.BaselineOnly.

[19] *Surprise SVD.* [Accessed: 04-05-2021]. URL: https://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.prediction_algorithms.matrix_factorization.SVD.

[20] *Surprise SVD++.* [Accessed: 04-05-2021]. URL: https://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.prediction_algorithms.matrix_factorization.SVDpp.

[21] *Surprise MAE.* [Accessed: 04-05-2021]. URL: https://surprise.readthedocs.io/en/stable/accuracy.html#surprise.accuracy.mae.

[22] *Surprise MAE.* [Accessed: 04-05-2021]. URL: https://surprise.readthedocs.io/en/stable/accuracy.html#surprise.accuracy.rmse.

[23] *Sklearn RandomizedSearchCV.* [Accessed: 04-05-2021]. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html.

# Appendices

# Appendix 1 - Lihtlitsents