FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	User subaccounts detection for better recommendation
Student:	Tomáš Vopat
Supervisor:	doc. Ing. Pavel Kordík, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2019/20

Instructions

Survey methods for identification of users under shared accounts in online content streaming. Design and implement algorithms capable of detecting shared accounts and for account segmentation. Measure the effect of user account segmentation into subaccounts on the performance of a collaborative filtering based recommendation system. Evaluate on real data provided by Showmax online video streaming service.

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague December 10, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

User Subaccounts Detection for Better Recommendation

Tomáš Vopat

Department of Applied Mathematics
Supervisor: doc. Ing. Pavel Kordík, Ph.D.

May 16, 2019

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In V Praze on May 16, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Vopat, Tomáš. *User Subaccounts Detection for Better Recommendation.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Sdílení účtů ve streamovacích službách má negativní dopad na doporučovací systémy a následně i na kvalitu služeb poskytovanou jejich uživatelům. Tato práce má za cíl navrhnout metodu schopnou detekce takových uživatelských účtu. Navrhli jsme tedy a následně implementovali algoritmy založené na kolaborativním filtrování a metodě klouzajícího okénka, které jsou schopné odhalit sdílené účty. Představené algoritmy umožňují detekovat aktivitu jiných osob s vysokou přesností. Tato aktivita může být z účtu odfiltrována tak, aby doporučovací systém obdržel pouze relevantní data. Kromě toho se uživatelé snaží sdílením účtu vyhnout platbě předplatného, a tak mohou být takové účty omezeny.

Klíčová slova detekce uživatelských podúčtů, sdílené účty, identifikace uživatele, segmentace uživatelských účtů, podezřelá aktivita uživatele, kolaborativní filtrování, klouzající okénko, doporučovací systémy, video streaming service, Showmax

Abstract

Account sharing in online streaming services has a negative impact on recommendation systems and consequently on the quality of services provided to the users. This thesis aims to design a method capable of detecting such accounts. To address this issue, we designed and implemented algorithms based on collaborative filtering and sliding window method revealing shared accounts. The proposed algorithms allow detection of other user's activity with high accuracy, which can be filtered and recommendation system gets only relevant data for a given user. Furthermore shared accounts can be restricted since there is a subscription that users try to avoid.

Keywords subaccounts detection, shared accounts, user identification, account segmentation, suspicious user activity, collaborative filtering, sliding window, recommender systems, video streaming service, Showmax

Contents

List of Figures

List of Tables

Introduction

Over the past years, many online streaming services, focused on TV series, movies or music, were developed. Each of these services provides a lot of multimedia content. The user has to spend an increasing amount of time to browse the library, to find interesting content based on his (her) preferences.

As a reaction to this problem, recommender systems were introduced. Their aim is to eventually recommend suitable content, in dependence on the user's preferences. The most simple one selects assets for the user with the use of predefined preferences (e.g. favorite genre, actor, director, studio). These preferences can be hard-typed during registration or may be learned from the user's behavior.

Online streaming services are under a tuition fee, therefore some users share their accounts between multiple personas to avoid a periodical fee. Every person has different preferences and consumes different set of multimedia. Described observation has a negative impact on system recommending content based on historical session logs. It causes the suggestions of inappropriate assets, not just for a single user using a shared account, but for all of them in extreme cases.

The aim of this study is the design an algorithm capable of analyzing any given user account whether it is used by more than one single real person. To evaluate it, we use historical session logs of user's activity on online streaming services. If we succeed, we will demand to mark suspicious activity that probably belongs to another persona.

Thanks to this determination, it is possible to suppress the other persona's activity and not consider it during the recommendation process. It leads to the increase of efficiency of the recommending systems as only the relevant information is given to the user. Other potential application is to split user account into multiple virtual sub-accounts, each for one real persona with corresponding logs, so the recommending can be applied on a subset of relevant data. Furthermore, there may be financial interest in sharing accounts recognition since monetary loss grows together with the number of these accounts.

In the Chapter 1 we mention a few works that are related to this thesis. Afterwards we analyze the shared accounts and the problems that account's sharing carries. Based on this knowledge we design and implement algorithms capable of detecting such accounts in the Chapter 2. Then in Chapter 3 we process the data provided by Showmax company. Finally, we perform experiments with proposed algorithms and evaluate their results in the Chapter 4.

Chapter 1

Related Concepts

"*Generating accurate and personalized recommendation when there are multiple individuals sharing the same user account is a challenging problem, which can greatly affect performance of recommender systems.*" [1] Hence, there are several studies focused on detecting the existence of multiple personas behind the same user account.

The study [2] solves the problem of user identification in share accounts with these goals: (1) Given an account with its historical session logs – identify a set of users who share such an account; (2) Given a new session issued by an account – find the corresponding user among the identified user of such account; (3) Boost the performance of item recommendation by user identification.

They propose an unsupervised learning-based framework *Session-based Heterogeneous graph Embedding for User Identification* (SHE-UI) to differentiate and model the preferences of users in an account and to group sessions by these users. The heterogeneous graph is constructed to represent items such as songs and their available metadata.

After creating the graph they, apply sampling a method based on normalized random walk to learn node embedding from which they compute the feature representation of the session by combining item features. After obtaining the the session features, they detect the number of users of each account and group sessions by their actual users. This approach supposedly improves recommendation by almost 39 %.

Another study [1] aims to identify users in shared accounts with use of rich contextual information (e.g. device, location, time). They try to discover whether the viewing log contains a pattern implying dissimilar preferences within a given account with a projection based method – Principal Component Analysis (PCA).

Their method is based on the observation: *If there is a single persona in an account the lower dimensional projection of the rating events will be close to each other* [1]. After discovering the shared accounts, they identify distinct

personas by clustering the rating event points. In the conclusion, they find generating accurate and personalized recommendation for shared accounts to be a challenging problem, but their solution slightly improves recommendation system.

There is also an algorithm [3] based on mining different preferences over different time periods from consumption logs addressed to the user identification in IP-TV services. They expect users within a shared account to not only have a different preferences for programs, but also get used to consuming services in different time periods. Hence, the account preference over a time period can be captured in the {account × item × time} 3-dimensional space.

They use empirical split method or average split method to determine non-overlapping sub-periods, so the user account will be split into many virtual sub-accounts. These virtual accounts are subsequently merged by modified DFS (BFS) algorithm into a few accounts according to the similarity measure (these merged accounts represent real personas that use the given account). [3]

A similar idea of a 3-dimensional space – the {account × item × time} clustering – is also used by Tensor factorization based subspace clustering and preferences consolidation (TCC) [4].

All the previously mentioned algorithms use contextual information (e.g. time, duration, item's metadata), however this information is not always available. This problem is tackled by Verstrepen and Goethals [5]. They represent users in binary matrix (users × items), where 1 represents a known preference and 0 represents unknown, without any other information.

In that paper, it is determined that there are three problems that are cause by sharing account: (1) Recommendations are relevant only to few users in the shared account (dominance); (2) Recommendations are not appealing to any of the users (generality); (3) Recommendations are relevant, but users don't know to which of they belong (presentation).

Chapter 2

Analysis and Design

To design algorithms capable of deciding whether a user account is shared between multiple personas, we have to analyze in Section 2.1 user's behavior on online streaming services – which preferences they have, what assets they watch and describe what shared accounts should look like. Then it is possible to propose the main idea of our algorithm (Section 2.2), using these account's differences to discover shared accounts. Lastly, we design and introduce a few implementations (Section 2.4) to achieve our intention.

2.1 Analysis

Each user at online streaming services has some historical view-log that contains assets watched in the past. These historical view-logs of all the users have been provided to us by Showmax company. It is visible in the view-logs which user watched which asset, what time the user started to watch and how long the user was watching that asset.

We suppose that every single user, according to his (her) preferences, has some subset of assets that he intends to watch. The subset of assets is not unique for every user, but there exist groups of users with the same preferences. In other words, teenage boys most likely watch sci-fi movies and comedy series. On the other hand, the older generation watches mostly historical documents and news.

The preferences may change from time to time depending on the user's age, time of the year or user's surroundings. But in a small time period, the preferences should be consistent. For example, a single person is not supposed to watch Chinese anime, a romantic movie, gardening series and a sci-fi movie in a small time period (e.g. week).

We can draw from these presumptions that a user account containing view-log, which is not specific to any group of users with similar preferences, is *used by multiple personas*. Each persona using this single account has different preferences and belongs to another group of users with different preferences.

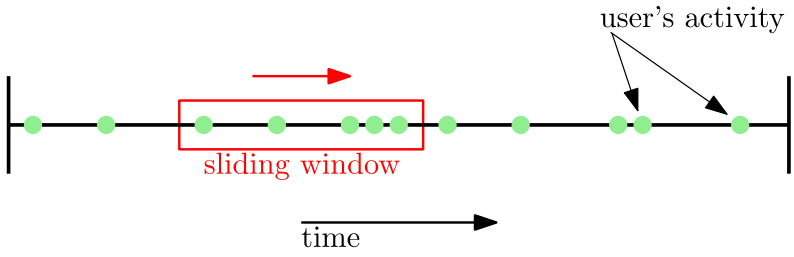


Figure 2.1: Account's historical view-log with sliding window

Of course, there can be a person with diversified preferences that doesn't fit into any group of users, but these persons are very rare and our approach doesn't fit each user into a box of preferences that he may not leave. We admit that these users may exist, but the preferenced hard core can be exactly described.

From a different perspective, two personas using the same account may have exactly the same preferences. Then it is not possible, with resources that have been provided to us (view-log), to tell how many personas are hidden behind the shared account, not even separate activity that comes from each one.

2.2 Evaluating User Consistency in Sliding Window

Based on historical view-logs, users have interaction embedding (IE), a set of assets that the user has interacted with. The IE can clearly capture the user's preferences and subsequently classify the given user to the exact group of the most similar users. Supposedly, there should be only a few distinctions in the IE – the penetration of the set of IEs in a single group of users should be numerous.

As time goes by, preferences may change, but the IE still contains all the logs including those that are not relevant anymore. This problem should be solved by selecting a small continuous interval in the account's view-log as shows Figure 2.1. In this interval, there are user's preferences considered to be consistent – user preferences should be easily classifiable. In summary, the main idea of our algorithm is that the user's preferences are consistent in a small interval, otherwise, there is another persona using this account.

Our proposal is to learn the user's preferences on a small interval, where the account is considered to be consistent, then compute the probability with which the user of the account intends to watch the first asset that follows the selected interval as shows Figure 2.2. We assume that the single-user account is easily tracked and the user doesn't digress in most of the cases. There can be

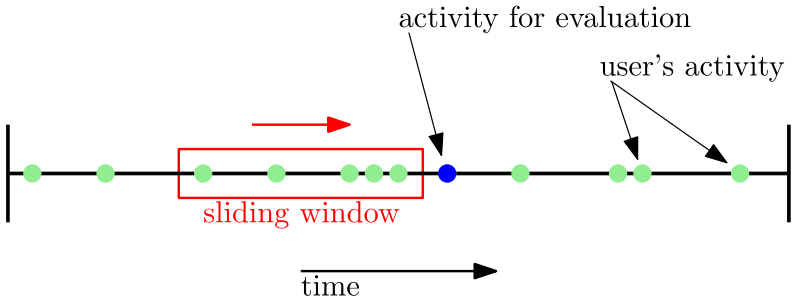


Figure 2.2: Evaluating account's historical view-log

some abnormalities, but only rarely. Hence, accounts with lots of distractions are used by more than one person.

Sliding window technique is unique across all the related studies. The only analogy can be found in [3, 4], where they expect periodic behavior – users have different preferences for programs, but they are also used to consuming services in different time periods.

2.3 Theoretical Background

In this section, we introduce the terms that we will use in our thesis. We define the mathematical signage, interaction embedding, which is important for our approach, then metrics and their conditions and on of the most popular machine learning (ML) method *collaborative filtering*. It's important to go through this section to fully understand the proposed algorithms.

2.3.1 Basic Terms

For better orientation and simplification designing the algorithms, we define some basic mathematical signage. It will help us while introducing the algorithms in the following Section 2.4. We define:

$$U = (u_0, u_1, u_2, \ldots, u_{n-1}) \tag{2.1}$$
$$A = (a_0, a_1, a_2, \ldots, a_{m-1}) \tag{2.2}$$
$$W_i = (w_{i0}, w_{i1}, w_{i2}, \ldots, w_{ip-1}) \tag{2.3}$$
$$S_{iq} = (w_{iq}, w_{iq+1}, w_{iq+2}, \ldots, w_{iq+r-1}) \tag{2.4}$$

where

- U is a set of all users,
- A is a set of all assets in our database,
- W_i is a view-log of user u_i and

- S_{iq} is a q-th interval (in Section 2.2 described as a sliding window) of view-log W_i with size r.

With these conditions:

$$|U| = n \tag{2.5}$$
$$|A| = m \tag{2.6}$$
$$|W_i| = p \tag{2.7}$$
$$0 < r < p \tag{2.8}$$
$$i \in \{0, 1, \ldots, n-1\} \tag{2.9}$$
$$j \in \{0, 1, \ldots, p-1\} \tag{2.10}$$
$$q \in \{0, \ldots, p-r-1\} \tag{2.11}$$
$$w_{ij} \in A \tag{2.12}$$

2.3.2 Interaction Embedding

We define an interaction embedding $IE_{i,\bullet}$ for every user u_i from the set U. It is a *bar vector* where each element represents some asset from A. Based on that, if the given user watched the asset, its value is 0 or 1.

$$ie_{i,j} = \begin{cases} 1, & \text{user } u_i \text{ saw asset } a_j \\ 0, & \text{otherwise} \end{cases}$$

If we put all those vectors into a matrix, we create an $IE \in \{0,1\}^{n,m}$ for all user's interactions with assets. This matrix would be extremely sparse, so our inner implementation will be via a sparse matrix.

$$IE = \begin{pmatrix} ie_{0,0} & ie_{0,1} & \cdots & ie_{0,m-1} \\ ie_{1,0} & ie_{1,1} & \cdots & ie_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ ie_{n-1,0} & ie_{n-1,1} & \cdots & ie_{n-1,m-1} \end{pmatrix}$$

In [1] represents a user as a featured vector. We do the same thing for an asset, for which similarity is then computed. The approaches are similar up to a point. We do not attempt to lower dimension of this vector as they do from n-dimensional to 1-dimensional space.

2.3.3 Metrics

The distance metric d is used to compute the distance between vectors. It is defined as a function $d(a, b)$, where a, b are vectors with the same number of features. There are a few conditions the must be satisfied [6, p. 10]. The x, y, z

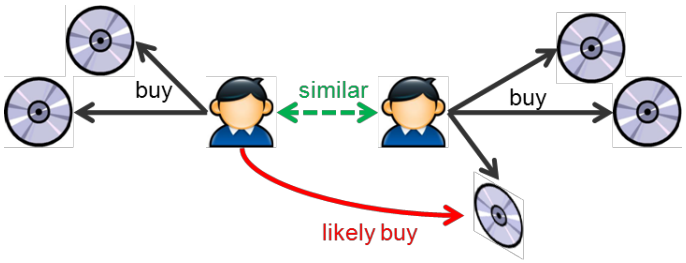


Figure 2.3: User-based CF. Source: [8]

are vectors with n features.

$$d(x, y) \geq 0 \qquad \text{(non-negativity)} \qquad (2.13)$$

$$d(x, y) = 0 \iff x = y \qquad \text{(identity)} \qquad (2.14)$$

$$d(x, y) = d(y, x) \qquad \text{(symmetry)} \qquad (2.15)$$

$$d(x, z) \leq d(x, y) + d(y, z)) \qquad \text{(triangle inequality)} \qquad (2.16)$$

There are several well-known metrics that meet these conditions, but our algorithms will use the *cosine distance* 2.18 in most of the cases.

$$d_E(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \qquad \text{(Euclidean distance)} \qquad (2.17)$$

$$d_C(x, y) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}} \qquad \text{(Cosine distance)} \qquad (2.18)$$

$$d_M(x, y) = \sum_{i=1}^{n} |x_i - y_i| \qquad \text{(Manhattan distance)} \qquad (2.19)$$

2.3.4 Collaborative Filtering

"*Collaborative filtering (CF) systems work by collecting user feedback in the form of ratings for items in a given domain and exploiting similarities in rating behavior amongst several users in determining how to recommend an item.*" [7]

The key idea of CF is that similar users share the same interests and user likes similar items. This method is based on users' past behavior and can be further divided into two categories of CF:

user-based recommends items suitable for similar users as show in picture 2.3,

item-based recommends items that are similar to previously consumed items.

At user-based CF, let's have an $n \times m$ matrix, with user u_i, where $i \in \{0, \ldots, n-1\}$ and item p_j, where $j \in \{0, \ldots, m-1\}$. We want to predict

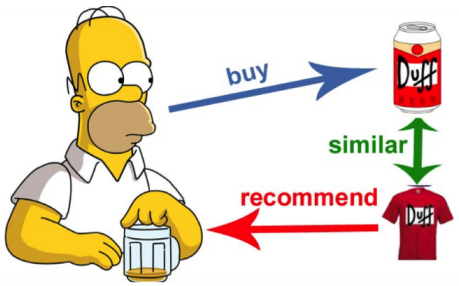


Figure 2.4: Item-based CF. Source: [10]

the rating $r_{i,j}$ if target (user) did not watch/rate an item p_j. The process is to calculate the similarities between target user u_i and all the other users, select the top-k similar users and take the weighted average of ratings from these top-k users with similarities as weights. Two ways to calculate similarity are *Pearson Correlation* and *Cosine Similarity* [9].

Without knowing anything about the items and users themselves, we think two users are similar when they give the same item similar ratings. Analogously the item-based approach, [9] says that two items are similar when they receive similar ratings from the same user as shown at figure 2.4.

2.4 Algorithms

Proposed algorithms implement the idea of sliding window and collaborative filtering, each in a slightly different way. All of them compute the probability with which the user intends to watch the first asset that follows a small interval. The leeway lies in the method of computing probability and selecting a set of the most similar users.

Each variant has the hyperparameter – the size of the interval – which may in a certain extent affect the results. Algorithms are iterative – in each step (for each interval) there is a computation of the probability with which the user watches the asset that follows the interval. Also, the computational complexity varies.

It's necessary to go through all the basic terms from section 2.3.1 to understand the algorithms. All the variables are exactly as described in that section.

In relation to other studies, same as the [5], we do not use any contextual information about assets or users except the watching start time. What we also have in common with this study is representing users and assets in binary matrix (users $\times$ assets) where 1 represents preference and 0 unknown.

On the other hand, they use simple collaborative filtering without considering the varying of preferences.

Most of the related theses [2, 4, 3] have a two-fold approach: (1) Identify a set of users who share an account; (2) Find corresponding user to a given new session. Our aim is only to decide whether the account is shared and to mark the activity that belongs to another user. We do not attempt to identify which user is active at the moment.

2.4.1 Freeride

We will need some auxiliary functions for this algorithm. The Top-k algorithm (Algorithm 1) returns the greatest values from an array *including indices* of the values. Those indices will be used later for recognizing which user was the most similar. A min-heap to select the values with function is used:

createHeap creates min-heap,

getMinHeap removes the minimal element from the heap, ,

insertHeap pushes an element into the heap.

Algorithm 1 Top-k algorithm

```
1: function getTopK(vector, k)
2:     h ← createMinHeap(
3:     for ) do(v, i) in vector        ▷ i is the index of the value in the vector.
4:         if size(h) > k then
5:             getMinHeap(h)
6:         end if
7:         insertHeap((v, i), h)    ▷ To the heap are inserted tuples, ordered
   by value v.
8:     end for
9:     return h                  ▷ Inner implementation of the heap is an array.
10: end function
```

Another auxiliary function *Compute Probability algorithm* (Algorithm 2)is needed for computing the probability with which the user watches the given asset. The probability $p_{u,a}$ is defined as

$$p_{u,a} = \frac{\sum_k v_{k,a} w_{u,k}}{\sum_k w_{u,k}}$$

where $v_{k,a}$ is 1 (user u_k saw asset a_a) or 0 (unknown). $w_{u,k}$ is the similarity rate of user u_u to user u_k. This function also uses the function $userSawAsset(i, asset)$ which return `True` or `False`, whether the user u_i saw given asset. The pseudocode is described in Algorithm 3. The time complexity of this algorithm is: $\mathcal{O}(p(rn + n \log n))$.

Algorithm 2 Compute probability algorithm

```
1: function computeProbability(top, asset)
2:     num ← 0
3:     den ← 0
4:     for (t, i) in top do
5:         if userSawAsset(i, asset) then
6:             num ← num + t
7:         end if
8:         den ← den + t
9:     end for
10:    return num/den
11: end function
```

The first version of our algorithm for marking suspicious activity has two hyperparameters – r size of the interval and k number of the most similar users that are selected.

Algorithm 3 Freeride algorithm

```
1: result ← []
2: for all q do
3:     next ← w_{iq+r}                ▷ q + r is an index to the view-log of user i
4:     sum ← 0⃗                        ▷ 0⃗ is zero-vector
5:     for all s in S_{iq} do         ▷ Iteration over all assets in the interval.
6:         ie ← getInteractionEmbedding(s)
7:         sum ← sum + ie
8:     end for
9:     top ← getTopK(sum, k)          ▷ Returns array of tuples (value, index).
10:    result[q] ← computeProbability(top, next)
11: end for
12: return result
```

2.4.2 Downtown

The second algorithm (Algorithm 4) is less computationally demanding and uses a straight approach to recognize suspicious activity. It uses *cosine similarity* 2.18 to compute the similarity between two interaction embeddings 2.3.2. The resulting probability is the minimal distance (maximal similarity) between two embeddings. This algorithm has only one hyperparameter – size r of the interval. Its time complexity is: $\mathcal{O}(prn)$.

Algorithm 4 Downtown algorithm

```
1:  result ← []
2:  for all q do
3:      next ← w_{iq+r}                ▷ q + r is an index to the view-log of user i
4:      nextIE ← getInteractionEmbedding(next)
5:      minDist ← 1
6:      for all s in S_{iq} do
7:          ie ← getInteractionEmbedding(s)
8:          dist ← cosineDistance(ie, nextIE)
9:          if dist < minDist then
10:             minDist ← dist
11:         end if
12:     end for
13:     result[q] ← minDist
14: end for
15: return result
```

2.4.3 West Coast

This modification (Algorithm 5) of the designed algorithm is clearly based on item-based *collaborative filtering* 2.3.4 approach with the main idea implemented. In every iterating step, we select top-k most similar assets using IE, on which cosine distance is later applied. Its time complexity is: $\mathcal{O}(pr(mn + m \log m)$.

2.4.4 Upper East Side

The last algorithm *Upper East Side* (Algorithmn 6) is an accelerated version of algorithm *West Coast* 2.4.3. It also searches for the most similar assets to the ones in the interval, but all the IEs in the interval are summarized, so we don't have to go through all the IEs over and over to find the top-k. In other words, by summarization of the IEs, we get all the preferences in one shot. We expect this algorithm to have slightly worse results, but the computational effectiveness is counterbalancing. The time complexity is: $\mathcal{O}(p(rn + mn))$.

The function `getMostSimilarIE` 11 computes the *cosine distance* 2.18 between the given IE and the all others IEs in the database. The result of this function is top-k most similar IEs. This is the most complex part of the algorithm. It can be accelerated by pre-computing the top-k for each asset.

Algorithm 5 West Coast algorithm

```
1: result ← []
2: for all q do
3:     next ← w_{iq+r}                ▷ q + r is an index to the view-log of user i
4:     nextIE ← getInteractionEmbedding(next)
5:     minDist ← 1
6:     for all s in S_{iq} do
7:         topK ← getMostSimilarAssets(s, k)  ▷ Return an array of the most similar assets.
8:         for t in topK do
9:             ie ← getInteractionEmbedding(t)
10:            dist ← cosineDistance(ie, nextIE)
11:            if dist < minDist then
12:                minDist ← dist
13:            end if
14:        end for
15:    end for
16:    result[q] ← minDist
17: end for
18: return result
```

2.5 Implementation

To implement proposed algorithms we have chosen *Jupyter Notebook*[1], which is an open-source web application for creating documents consisted of *Markdown* text blocks and *Python*[2] source code that can be executed right in this environment. None of the proposed algorithms were taken, all of them have been designed and implemented by ourselves.

First of all, we load users and assets from files *users.csv* and *assets.csv* for keeping them in an exact order and to avoid the need for loading them over and over again from the database. Keeping this information in the main memory will speed-up the execution. We also create dictionaries to be able to get the array index from the user's id or asset's id. Also the IE will be loaded from file `ua_mat_csr.npz` as a sparse matrix, whereas rows represent users and columns represent assets. Both of them are in the same order as in the previously loaded files.

SciPy[3], the scientific library was used for storing and manipulating with IE matrix with dimensions $813\,932 \times 35\,686$, because even a bit representation of this matrix would take 3,381 GiB. SciPy's Sparse library has lowered this

[1] https://jupyter.org
[2] https://www.python.org
[3] https://www.scipy.org

Algorithm 6 Upper East Side algorithm

```
1:  result ← []
2:  for all q do
3:      next ← w_{iq+r}                ▷ q + r is an index to the view-log of user i
4:      nextIE ← getInteractionEmbedding(next)
5:      minDist ← 1
6:      sum ← 0⃗
7:      for all s in S_{iq} do
8:          ie ← getInteractionEmbedding(t)
9:          sum ← sum + ie
10:     end for
11:     topK ← getMostSimilarIE(sum, k)
12:     for t in topK do
13:         dist ← cosineDistance(ie, nextIE)
14:         if dist < minDist then
15:             minDist ← dist
16:         end if
17:     end for
18:     result[q] ← minDist
19: end for
20: return result
```

size to 76,2 MiB. Very handy was also the manipulation with vectors that this library contains (e.g. transponing).

To get the view-log for the given user we use the connection to the database, where we select the records by user's identifier ordered by `start_time`. We used *Pandas*[4] library, which provides tools to process structured data (especially in CSV format) and communication with the database as well.

For proceeding users in intervals, the pre-set number of records from view-log was pushed to the queue, which was cyclically shifted, while we needed to compute the probability of watching all the assets in the view-log.

Computation of the *Cosine Distance* is done by *scikit-learn*[5] a machine learning library for *Python* which provides many useful algorithms (e.g. clustering, regression or dimensionality reduction) as well as computing the cosine similarity between two vectors.

A heap for selecting the top-k users (assets) in our algorithms. Python's standard library provides an implementation of the heap queue algorithm (also known as the *priority queue*) called `heapq`. This implementation uses arrays, which is probably the best way (clear and easy). Our algorithms use these functions:

[4] https://pandas.pydata.org
[5] https://scikit-learn.org

heappushpop(heap, item) Push `item` on the heap, then pop and return the smallest item from the `heap`,

heappush(heap, item) Push the value `item` onto the `heap` [11].

Lastly, the results of the algorithms were saved to the *CSV* files in the corresponding folders on the enclosed medium in folder `/src/impl/`, where complete source codes in *Jupyter Notebook* can be found as well. Results have this structure:

id view-log indentifier,

start_time start time of watching,

net_time duration of watching,

user_id user's identifier,

asset_id asset's identifier,

probability probability of watchning this asset.

Chapter 3

Data Preparation

3.1 Preprocessing

Showmax company has provided its data about multimedia content available in their online streaming service and user's data about their activity. First of all, we have to the explore received data – their structure, which information they contain and how many records there are. In total we got 37 GiB data with 324 263 311 records about assets and users. It contains 819 949 records of metadata and 323 443 362 records of user's activity. A more detailed description can be found in Table 3.1. It is curious that some files are really small and some are really large.

All those files are in *CSV* (Comma Separated Values) format. It is a very old, very simple and a very common standard for tabular data supported by a huge number of tools from spreadsheets like *Excel* to complex databases

Table 3.1: Files Description

File name	Number of Lines	Size
media_asset.csv	104 925	30,3 MiB
media_assetcategory.csv	201 494	31,4 MiB
media_assetmetadata.csv	208 987	2,9 MiB
media_assetmetadata_rating_levels.csv	33 518	10,8 MiB
media_assetmetadata_language.csv	68 111	16,3 MiB
media_assetmetadata_person.csv	183 261	16,7 MiB
media_category.csv	648	38 KiB
media_person.csv	18 833	2,1 MiB
media_rating.csv	4	70 B
media_ratinglevel.csv	168	56 KiB
viewership.csv	323 443 362	36,81 GiB
TOTAL	324 263 311	36,92 GiB

or programming languages. *CSV* is a two-dimensional structure consisting of rows of data, each row containing multiple cells. Cells within a row are separated by commas. [12]

File *media_asset.csv* contains information about each asset, which is available to users for watching. Its structure is as follows:

uuid unique identifier (primary key),

provider_id provider's identifier,

name asset's name,

created_at creation date,

updated_at date of last update,

type asset's type,

parent_asset_id identifier of parent asset,

number unknown parameter,

disabled asset is disabled for watching,

network_id unkown parameter,

vod_model unkown parameter,

weight unkown parameter,

deleted asset is deleted,

slug unknown parameter,

published_at date of publication,

download_policy_id unknown parameter,

show_in_search enabled for search,

show_children visible for children,

next_live_episode_start unknown parameter,

auto_published_at date of auto publication,

promoted asset is promoted.

Some of these parameters are unknown or useless, so we choose only relevant ones for our propose – `uuid`, `name`, `created_at`, `updated_at`, `type`, `parent_asset_id`, `disabled`, `published`, `weight`, `deleted`, `published_at`, `show_is_search`, `show_children` and `promoted`. For an unknown reason there is a comma at the beginning of each file. The comma has to be removed, because it adds a blank column to the file, so we remove it with Source Code 3.1. Afterwards it is possible to cut off columns that we won't need with Source Code 3.2.

Source Code 3.1: Removing commas

```
#! /bin/bash

sed -i '1s/^,//' media_*.csv
```

Source Code 3.2: Cutting columns in *media_asset.csv*

```
#! /bin/bash

cat media_asset.csv | \
cut -d',' -f2,4-8,10,11,14,15,17,19,20,23 > tmp.csv
mv tmp.csv media_asset.csv
```

To make changes in the files we used two basic UNIX programmes – *sed* and *cut*. The *sed* stream editor is a text editor that performs editing operations on information coming from standard input or files. [13] On the other hand, the *cut* is used for cutting out the sections from each line of files. It can be used to cut parts of a line by byte position, character or field. [14]

Columns `created_at` and `updated_at` contains date and time values. International Standard *ISO 8601* specifies numeric representations of date and time. This standard notation helps to avoid confusion in international communication caused by many different national notations and increases the portability of computer user interfaces [15]. In our case the format

```
YYYY-MM-DD hh:mm:ss.ssssss+hh:mm
```

is used for capturing date and time. This format is used across all the files.

File *media_category.csv* contains categories into which some of the assets belong to. The file has the following structure:

id identifier,

name name of the category,

category_id category identifier (primary key),

type type of category (genre, tag, type or other).

There is a little confusing occurrence of two possible primary keys – `id` and `category_id`. Let's suppose that `id` is generated during export as a record number, so we won't need it during further processing and can be removed with the Source Code 3.3.

Source Code 3.3: Cutting columns in *media_category.csv*

```
#! /bin/bash

cat media_category.csv | cut -d',' -f2-4 > tmp.csv
mv tmp.csv media_category.csv
```

File *media_assetcategory.csv* contains categories into which some of the assets belong to. The file has the following structure:

unknown index generated during export,

id unique identifier,

created_at creation date,

updated_at date of last update,

asset_id asset's identifier,

category_id category identifier,

weight weight with which the asset belongs to the category.

From the structure, it is visible that this data is used to resolve two database entities in relationship many-to-many (m:n). In other words, any given asset may belong to an unlimited number of categories and a category may be assigned to any number of assets. Another strange thing is that the file contained in its header lonely comma, which was removed a few steps above. It wasn't a stray comma like in other cases, but there is a real unnamed column containing indexes generated during the export. We will drop this column and keep all the others using Source Code 3.4.

Source Code 3.4: Cutting columns in *media_assetcategory.csv*

```
#! /bin/bash

cat media_assetcategory.csv | awk 'BEGIN {FS=","; OFS=","}
NR==1 {print "," $0}
NR>1 {print $0}' | cut -d',' -f2- > tmp.csv
mv tmp.csv media_assetcategory.csv
```

In this source code, we used a new program called *awk*. "*It is a utility that enables a programmer to write tiny but effective programs in the form of statements that define text patterns that are to be searched for in each line of a document and the action that is to be taken when a match is found within a line.*" [16]

Next file *media_assetmetadata.csv* has structure as follows:

unknown index generated during export,

asset_id asset's identifier,

section identifier of section,

year year of movie release,

imdb_id imdb identifier,

showmax_rating_id unknown parameter.

Some of the columns are foreign keys to some tables that we do not have access to (e.g. `section`, `imdb_id`, `showmax_rating_id`). These columns won't help us at all since we do not know their meaning. Furthermore, the same problem occurs with *media_assetcategory.csv* and stray comma, but there is no other unique identifier, so we need to keep it. The Source Code 3.5 will be applied.

Source Code 3.5: Cutting columns in *media_assetmetadata.csv*

```
#! /bin/bash

cat media_assetmetadata.csv | \
awk 'BEGIN {FS=","; OFS=","} NR==1 {print "id",$0}
NR>1 {print $0}' | cut -d',' -f1,2,4 > tmp.csv
mv tmp.csv media_assetmetadata.csv
```

File *media_assetmetadata_language.csv* contains titles and descriptions in different languages according to the language identifier. Its structure is:

unknown index generated during export,

id unique identifier,

asset_metadata_id asset's identifier,

language_id language identifier,

title asset's title in some language,

description asset's description in some language.

As in previous cases, the comma at the beginning of the file wasn't a mistake, but there is an unnamed column created during export from the database containing row indices. We will add the column to the header again and then we will cut off the column from the whole file. It is not an easy process, because columns are comma separated, but the last column `description` may contain comma and even a new-line character. If so, whole string in this column is surrounded by quotation marks. Our approach is to remove new-line characters in the strings and to change all the field separators from comma to left brace (left brace character does not appear in the file). Then we can exactly select columns to cut off using cut command.

The above-mentioned approach is too complicated to proceed in *Bash*. A better and easier way to accomplish our goal is to write a Python's script (Source Code 3.6) with the use of the library for parsing *CSV*.

Source Code 3.6: Parsing quted column with string

```
import csv
import os

f = open("media_assetmetadata_language.csv", "w")
with open("tmp.csv", "rb") as csvfile:
  reader = csv.reader(csvfile, delimiter=",",
  quotechar='"', quoting=csv.QUOTE_MINIMAL)
  for row in reader:
     f.write(row[1] + "{" + row[2] + "{" + row[3] + "{" +
     row[4].rstrip() + "{" + row[5].rstrip() + "\n")
f.close()

os.remove("media_assetmetadata_language.csv")
os.rename("tmp.csv", "media_assetmetadata_language.csv")
```

We have received in the file *media_ratinglevel.csv* some asset's ratings with this structure:

unknown index generated during export,

uuid unique identifier,

created_at date of creation,

updated_at date of last modification,

rating_id rating identifier,

name unknown parameter,

image_id image identifier,

image_light_id light image identifier,

image_dark_id dark image identifier,

image_dark_solid_id solid dark image identifier,

image_light_solid_id solid light image identifier,

showmax_rating_id unknown parameter,

slug unknown parameter,

weight rating's weight.

There are a few columns containing some image identifiers, which are unnecessary for our purposes because these are only some additional metadata for user interface with no measurable impact on the recommendation. As in previous cases, there is a nameless column with indices generated during the export. The column may be cut off due to `uuid` column that contains unique identifiers. `rating_id` parameter expresses, what company provided the rating – it could be Showmax, Polish TV or FPB (all of them are video streaming services). We do not know what values are in columns `name`, `showmax_rating_id` and `slug`, therefore we will remove them as well. After that we have a few columns left – `uuid`, `created_at`, `updated_at`, `rating_id` and `weight` which provide probably the only relevant data from this table and are possibly useful for some further processing. For modification of this file, we will use Source Code 3.7.

Source Code 3.7: Cutting columns in *media_ratinglevel.csv*

```
#! /bin/bash

cat media_ratinglevel.csv | awk 'BEGIN {FS=","; OFS=","}
NR==1 {print "id",$0} NR>1 {print $0}' | \
cut -d',' -f2-5,14 > tmp.csv
mv tmp.csv media_ratinglevel.csv
```

Next file *media_assetmetadata_rating_levels.csv* contains data for table decomposing many-to-many relationship between data from files *media_asset.csv* and *media_ratinglevel.csv*. We don't know why there is this type of relationship, because it is only necessary to store more ratings to the given asset (no need to assign one rating to more assets). Even stranger is the fact that this table does not contain any additional information (e.g. date). It would make sense if there were relation shown in schema in Figure 3.1. In this schema, it is possible to assign the same rating to more assets and the same asset may have more ratings awarded by users with some additional information like date or user identifier. Another option is to omit the decomposing table and use only

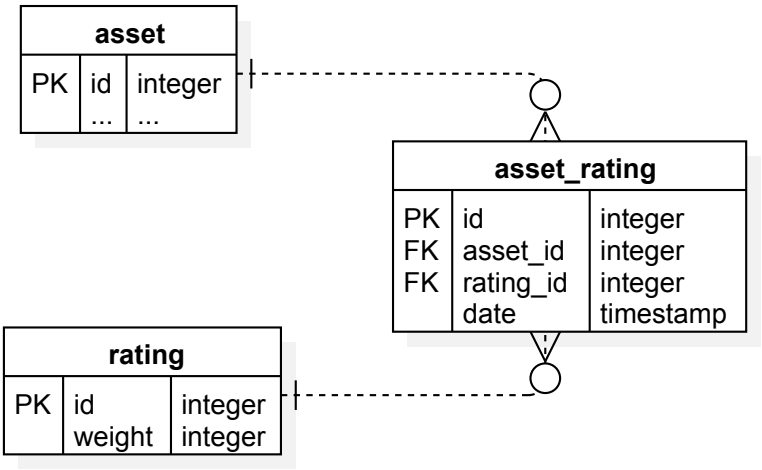


Figure 3.1: Many-to-many schema

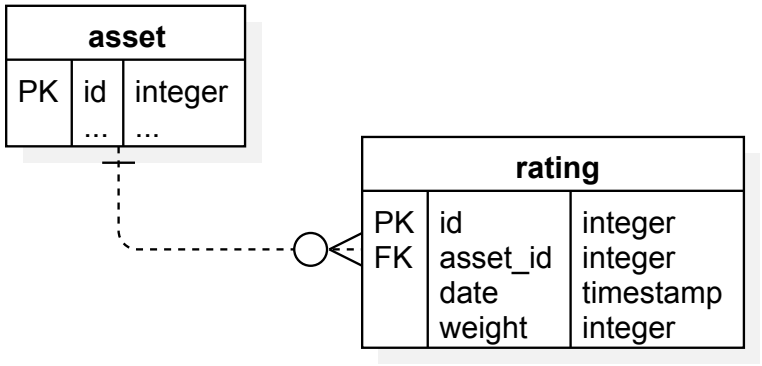


Figure 3.2: One-to-many schema

one table in relation one to many as described above. Then the tables would look like those in schema in Figure 3.2. We found the second solution more practical, easier to implement and without any loss of value.

The structure of this file is as follows:

unknown index generated during export,

id record identifier,

assetmetadata_id foreign key to media_asset table,

ratinglevel_id foreign key to media_ratinglevel table.

There is only needed to remove `unknown` column with indices generated during the export. Just like in previous cases we will add the stray comma

back to file and then we will cut off the whole column from the file with the Source Code 3.8.

Source Code 3.8: Removing column with indices

```
#! /bin/bash

cat media_assetmetadata_rating_levels.csv | \
awk 'BEGIN {FS=","; OFS=","}
NR==1 {print "id",$0} NR>1 {print $0}' | \
cut -d',' -f2- > tmp.csv
mv tmp.csv media_assetmetadata_rating_levels.csv
```

In the file *media_person.csv* there is a list of persons that are somehow connected to the exact asset (e.g. director, actor, composer). Its structure is:

unknown index generated during export,

uuid unique identifier,

created_at date of creation,

updated_at date of last update,

imdb_id identifier from IMDb.

We have a rather straightforward approach. Just like in the previous cases, we only remove indices generated during export, because there is another column `uuid` containing unique identifiers among the records. This includes adding back the stray comma as we have seen in the processing of file *media_ratinglevel.csv*. We will keep all the other columns, even though the last column `imdb_id` is inconsistent – it contains URLs, names and in the most cases it is empty – but we can use it as an auxiliary element during similarity validation. File modifications will be proceeded using the Source Code 3.9.

Source Code 3.9: Removing column with indices

```
#! /bin/bash

cat media_person.csv | awk 'BEGIN {FS=","; OFS=","}
NR==1 {print "id",$0} NR>1 {print $0}' | \
cut -d',' -f2- > tmp.csv
mv tmp.csv media_person.csv
```

File *media_assetmetadata_person.csv* is used for decomposing many-to-many relation between assets and persons. It has this structure:

unknown index generated during export,

id record identifier,

asset_metadata_id asset's identifier,

person_id person's identifier,

type unknown parameter, with range 0–3,

priority priority of the person, range 0–19.

There is also a unnecessary column containing indices generated during export that we won't need, so we can cut off this unnamed column, because there is another column `id` with unique record identifiers. Asset's identifier and person's identifier are foreign keys that are necessary to link the files. Column `type` has unknown values, that probably have some meaning regarding the relation between asset and person. Since we do not know it's meaning, it can be removed. The column `priority` expresses the importance of the relationship between a person and a asset. It can be used for computing the asset's similarity based on persons, so we will keep it. To perform these modifications we will use the Source Code 3.10.

Source Code 3.10: Removing column with indices and column with type

```
#! /bin/bash

cat media_assetmetadata_person.csv | awk 'BEGIN {FS=","; OFS=","}
NR==1 {print "id",$0} NR>1 {print $0}' | \
cut -d',' -f2-4,6 > tmp.csv
mv tmp.csv media_assetmetadata_person.csv
```

The last file *media_rating.csv* containing metadata about assets has this structure:

unknown index generated during export,

id record identifier,

name name of the rating,

slug optimized name identifier (lower-case and no spaces or special characters).

There are only three records in this file as shown in Table 3.2. The *media_ratinglevel.csv* file analysis provided the fact that the file contains information about the company that has provided the rating – not all the ratings come from Shomax streaming service. It's really important for some types of recommendation systems to have a sufficient amount of ratings because

Table 3.2: Content of *media_rating.csv*

	id	name	slug
0	3	FPB	fpb
1	5	Polish TV	polish-tv
2	6	Showmax	showmax

better-rated asset's are more likely suitable for recommendations than those with a lower rating.

The only modification that we will make is to cut off the column with generated indices. It's not necessary to keep it as long as there is a column with a record identifier. To achieve the modification, we will use the script in Source Code 3.11.

Source Code 3.11: Removing the column with indices

```
#! /bin/bash

cat media_rating.csv | awk 'BEGIN {FS=","; OFS=","}
NR==1 {print "id",$0} NR>1 {print $0}' | \
cut -d',' -f2- > tmp.csv
mv tmp.csv media_rating.csv
```

Finally, the most important file *viewership.csv* containing the user's activity logs (interactions with the assets). Its size is 36,81 GiB and has the following structure:

start_time start time of watching asset,

net_time duration of watching asset,

user_id user's identifier,

asset_id asset's identifier.

The data in this file decomposes many-to-many relationship between users and assets – it's a typical decomposition containing some additional information (`start_time` and `net_time`). `start_time` expresses the time when a user started to watch an asset and `net_time` expresses the duration of time the user spends watching it. But some column with a unique identifier for each record is missing, therefore we have to generate some. It will be done by the Source Code 3.12.

`&&` operator is used in this script. Lists (sequence of one or more pipelines) containing the AND conditional operator are evaluated from left to right. A command following the AND operator (`&&`) is executed if the previous command is successful [17, p. 24].

Due to the file size, every operation is very time consuming, even the simplest one. Therefore we have to choose programs that are effective and use stream data processing. Another problem is the file size – it's necessary to keep in mind that every copy takes a big amount of the limited disk space.

Source Code 3.12: Generating identifier

```
#! /bin/bash
cat viewership.csv | \
awk 'BEGIN {FS=","; OFS=","} NR==1 {print "id",$0}
NR > 1 {print NR-1,$0}' > tmp.csv && \
mv tmp.csv viewership.csv
```

We have analyzed all of the provided files and prepared them for further processing – inserting into the database. A careful reader may have noticed there is no information about users themselves (e.g. personal data, date of account creation or user preferences). This information would help us for discovering shared accounts, but it's not necessary to have it. We will extract users (their identifiers) from view-log in the next Section 3.2, during insertion into the database.

3.2 Database

In the previous section, we have analyzed raw data provided by Showmax company – we have investigated its structure, removed useless information (columns) to reduce its size and modified its structure to be easily processed. Our task is to find the best tool for data manipulation and selecting demanded information.

The data are too big to proceed in the main memory, which means that we can not use any programming language. Actually, it is possible, but our task would be greatly inefficient. Another option is to use some type of traditional database system. Mostly, these systems are designed to handle data ranging to hundreds of gigabytes. Many of them support only single work-station and serial processing, which might be in some cases really time-consuming.

Lastly, it is possible to use big data tools. Since relational databases have been pushed to the limit, a new breed of technologies has emerged. Many of these new technologies have been grouped under the term *NoSQL*. These technologies are complex and can scale to vastly larger sets of data [18, p. 2]. [19, p. 5] describes that the use of these tools is appropriate if *3 Vs* are met:

Volume amount of data,

Variety the data structure (e.g. structured, semistructured, unstructured),

Velocity speed of data generation and speed of proceeding data.

Our data conform only the variety so it will be best and most comfortable to store our data in a basic relational database. There is also a possibility to leave the data in their current state and place them into *Hadoop Distributed File System* (HDFS) for further processing.

HDFS supports the rapid transfer of data between compute nodes with the use of master/slave architecture. It provides high-performance access to data across a highly scalable Hadoop cluster. Hadoop ecosystem technologies provide reliable means for managing pools of big data and supporting related big data analytics applications [20].

The Hadoop approach would mean that we have used some engine for data processing (e.g. Spark[6], Hadoop[7]). These engines are supposed to be used in a production environment, but for the purposes of our experiments, it would be better to settle for writing custom disposable *SQL* queries and serial processing as our needs require.

To perform our experiments, we have chosen the relational database management system (RDBMS). *"A relational database is a set of formally described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables. The standard user and application programming interface (API) of a relational database is the Structured Query Language (SQL). SQL statements are used both for interactive queries for information from a relational database and for gathering data reports."* [21]

There are many implementations of RDBMS such as Oracle[8], MySQL[9] or Microsoft SQL Server. We will use *PostgreSQL* database, which is released under open-source licence PostgreSQL Licence, similar to the BSD or MIT licences. *PostgreSQL* provides all enterprise-class features such as SQL windowing functions, the ability to create aggregate functions [22, p. 9].

3.2.1 Creating Schema and Data Insertion

We have already preprocessed provided data in Section 3.1 so our next task is to create a database schema based on this data. Each data file will represent a single database table with columns corresponding to the file's columns and with some relations in-between.

We have to be careful while designing the schema because some of the columns contain empty records or inconsistent data types, thus we will make as few constraints as possible. Thus we will limit ourselves to setting up primary keys only. Because if there was any constraint that our data does not meet, the insertion process would fail. Later, we will set stricter constraints to the tables to keep only relevant and processable data.

[6] https://spark.apache.org
[7] https://hadoop.apache.org
[8] https://www.oracle.com/cz/database/
[9] https://www.mysql.com

To create a new database we have to be a superuser or have the special CREATEDB privilege. By default, it creates a copy of the standard database `template1`. [22, p. 27]. The creation is performed with command: `create database showmax;`.

After creating the database, it is possible to start creating tables. It's necessary to create tables step by step starting with the root table. Root table is a table that doesn't have any foreign keys to different tables (except foreign key to itself). Names of the tables will be based on the file names – file *example.csv* has the corresponding table named `example`. There are a few root tables that we can begin with (e.g. `media_category`, `media_rating`, `media_person` or `media_asset`).

We will create the set of root tables and then we will try to insert data from corresponding tables. We expect some troubles during the insertion so the tables will have primary keys only *without any not-null constraints*. These constraints will be added later if everything succeeds. To perform these actions we execute *SQL* Source Code 3.13.

Source Code 3.13: Creating tables – part 1

```
create table media_category (
    name varchar(255),
    category_id varchar(255) primary key,
    type varchar(255)
);
create table media_rating (
    id integer primary key,
    name varchar(255),
    slug varchar(255)
);
create table media_person (
    uuid varchar(255) primary key,
    created_at timestamp with time zone,
    updated_at timestamp with time zone,
    imdb_id varchar(255)
);
create table media_asset (
    uuid varchar(255) primary key,
    name varchar(255),
    created_at timestamp with time zone,
    updated_at timestamp with time zone,
    type integer,
    parent_asset_id varchar(255),
    number real,
    disabled boolean,
```

```
    published boolean,
    weight real,
    deleted boolean,
    published_at timestamp with time zone,
    show_in_search boolean,
    show_children boolean,
    promoted boolean
);
```

PostgreSQL supports moving files between the database and standard file-system files. `copy to` copies contents of a table to a file, while `copy from` copies data from a file to a table. Copied can be a result of a select query as well [22, p. 52]. The file for copying into has to be specified by the absolute path and the files may be in text, *CSV* or binary format (default is text format). In our case, it is a *CSV* format. Several parameters may be specified for copying – a list of columns to be copied, delimiter for separating columns, whether the file contains a header line with the names of each file and so on. We will only specify the delimiter, type of file and that the file contains header as stated in script in Source Code 3.14.

Source Code 3.14: Importing data to root tables

```
copy media_category from '/home/username/data/media_category.csv'
delimiter ',' csv header;
copy media_rating from '/home/username/data/media_rating.csv'
delimiter ',' csv header;
copy media_person from '/home/username/data/media_person.csv'
delimiter ',' csv header;
copy media_asset from '/home/username/data/media_asset.csv'
delimiter ',' csv header;
```

Execution of the script has failed on the last command while importing data from *media_asset.csv* file. We have received an error:

```
ERROR:  invalid input syntax for type boolean: "2mr4zbf0"
CONTEXT:  COPY media_asset, line 1996, column deleted: "2mr4zbf0"
```

After further inspection we have found out that on the 1996th line of the file *media_asset.csv* in the column `deleted` there is a value `2mr4zbf0`. This column should contain only boolean values (`true` or `false`). First of all, we have tried to delete this record, but after deleting this record same failure came on the line 2011 and then on the line 2014. So, we have written the Source Code 3.15 to count lines with inconsistent values. In summary, there are 536 records with inconsistent values. There are two possibilities how to deal with these lines – we can remove all the lines with inconsistent values

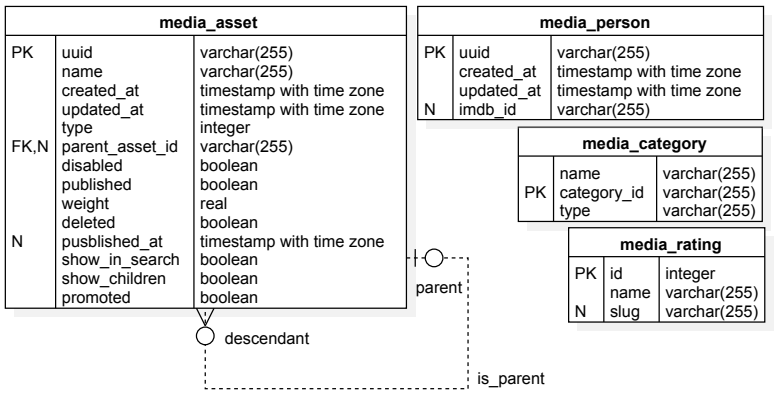


Figure 3.3: UML diagram of tables – part 1

or change it to a consistent state. We suppose that these lines are incorrect and we don't know which boolean value we should set in these records, so the better option is to remove these lines. All these commands are described in the Source Code 3.15. Then we can once again execute the Source Code 3.14 for copying data from *media_asset.csv* file to the database table `media_asset`. In this case, the process will success finally.

Source Code 3.15: Counting and removing inconsistent records

```
#! /bin/bash

cat media_asset.csv | awk 'BEGIN {FS=";"; OFS=","; total=0}
NR>1 {if ($10 != "True" && $10 != "False") {total+=1;}}
END {print "Number of inconsistent lines: " total}'
cat media_asset.csv | \
rm awk 'BEGIN {FS=";"; OFS=","} NR==1 {print $0}
NR>1 {if ($10 == "True" || $10 == "False")
{print $0;}}' > tmp.csv && mv tmp.csv media_asset.csv
```

We have imported data from these four tables to the database, so it is now possible to set stricter constraints. We have created UML diagrams shown in Figure 3.3 for each table to see more clearly, what state we want to achieve. In the diagram there is letter `N` next to a column that may be nullable (column may contain a null value), others must not contain null values. Letter `P` stands for *primary key*. Our current approach is to remove all the records that don't meet our requirements (according to the UML diagram in Figure 3.3). In other words, if there is a not-null constraint at some column, we will delete every row, that has a null value in this column and then we can set the not-null constraint. This will be done by script in Source Code 3.16 for every column with not-null constraint.

Source Code 3.16: Setting not-null constraints – part 1

```
delete from media_person
where created_at is null or updated_at is null;
alter table media_person alter column created_at set not null;
alter table media_person alter column updated_at set not null;

delete from media_category
where name is null or type is null;
alter table media_category alter column name set not null;
alter table media_category alter column type set not null;

delete from media_rating
where name is null;
alter table media_rating alter column name set not null;

delete from media_asset
where name is null or created_at is null or
updated_at is null or type is null or
disabled is null or weight is null or
deleted is null or show_in_search is null or
show_children is null or promoted is null;
alter table media_asset alter column name set not null;
alter table media_asset alter column created_at set not null;
alter table media_asset alter column updated_at set not null;
alter table media_asset alter column type set not null;
alter table media_asset alter column disabled set not null;
alter table media_asset alter column weight set not null;
alter table media_asset alter column deleted set not null;
alter table media_asset alter column show_in_search set not null;
alter table media_asset alter column show_children set not null;
alter table media_asset alter column promoted set not null;
```

We want to have as few nullable columns as possible. Only those columns that we consider not so important for our purposes are allowed (e.g. `imdb_id`, `slug`). There are also columns that have to be nullable – there must be at least one record with a null value in column `parent_asset_id` (table `media_asset`), otherwise, there would be a cyclical dependency. The column `published_at` (table `media_asset`) may contain null values as well – it's not necessary for all the assets to be published yet.

As visible from the UML diagram in Figure 3.3, there is a *recursive relationship*, it is a relationship between two entities of similar entity type. For example let's take the entity `person`. There may be two being of this type (e.g. mother and son) where both of them are of the same entity type but

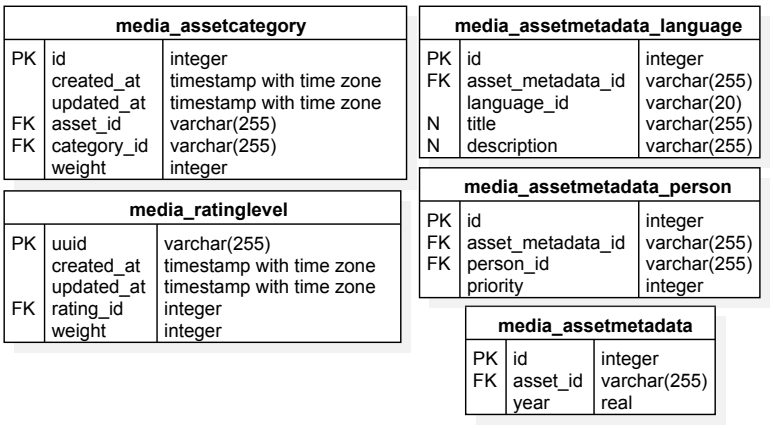


Figure 3.4: UML diagram of tables – part 2

we want to capture the relation between them (e.g. son is the descendant of the mother, the son is mother's relative). In our case it's almost the same – some asset may be superior to another asset (e.g. episode of TV series can be subordinate to TV series as a whole). That's why the recursive relationship is used in this table.

To let this relationship come into force, we have to make foreign key constraint that will check if there exists is a parent with the entered identifier. At first we will check for each asset if there is a parental asset with that identifier (null parents will be skipped) and then we can turn on the constraint using the Source Code 3.17.

Source Code 3.17: Adding foreign key constraint

```
delete from media_asset a
where parent_asset_id is not null and not exists (
    select * from media_asset b
    where a.parent_asset_id = b.uuid
);
alter table media_asset
add constraint fk_parent_asset foreign key(parent_asset_id)
references media_asset(uuid);
```

We have created root tables and that has allowed us to create other tables that depend on root the ones. It means that we weren't able to create other tables than those we have already created because other tables contain some foreign keys that were limiting us. In this step (Source Code 3.18), we will create these tables: `media_assetmetadata`, `media_assetmetadata_language`, `media_assetcategory`, `media_ratinglevel`, `media_assetmetadata_person`. We have designed UML diagrams (shown in Figure 3.4) as well, so we will create tables according to these diagrams. The process will be similar as in the

previous step – creating tables, importing data, setting not-null constraints. Then we will make foreign key constraints to other tables.

Source Code 3.18: Creating tables – part 2

```
create table media_assetmetadata (
    id integer primary key,
    asset_id varchar(255),
    year real
);
create table media_assetmetadata_language (
    id integer primary key,
    asset_metadata_id varchar(255),
    language_id varchar(20),
    title varchar(255)
);
create table media_assetcategory (
    id integer primary key,
    created_at timestamp with time zone,
    updated_at timestamp with time zone,
    asset_id varchar(255),
    category_id varchar(255),
    weight integer
);
create table media_ratinglevel (
    uuid varchar(255) primary key,
    created_at timestamp with time zone,
    updated_at timestamp with time zone,
    rating_id integer,
    weight integer
);
create table media_assetmetadata_person (
    id integer primary key,
    asset_metadata_id varchar(255),
    person_id varchar(255),
    priority integer
);
```

Source Code 3.19: Creating tables

```
copy media_assetmetadata from
'/home/username/data/media_assetmetadata.csv'
delimiter ',' csv header;
copy media_assetmetadata_language from
```

```
'/home/username/data/media_assetmetadata_language.csv'
delimiter '{' csv header;
copy media_assetcategory from
'/home/username/data/media_assetcategory.csv'
delimiter ',' csv header;
copy media_ratinglevel from
'/home/username/data/media_ratinglevel.csv'
delimiter ',' csv header;
copy media_assetmetadata_person from
'/home/username/data/media_assetmetadata_person.csv'
delimiter ',' csv header;
```

The Source Code 3.19 has initiated importing of the data to the table `media_assetmetadata_language`. Proceeding failed on line 876 with the following message. We have fixed that problem by removing this row from the file and tried to execute the command once again, but we have received the next failure saying that the column description doesn't have enough capacity to store a whole string. The first approach was to set a bigger length to that column (initial was 255 characters). We were increasing the column's size, but there were always some bigger strings. So we have written a Source Code 3.20 that counts the length of the string on each row. We have sorted those values and we have found out that the longest string has 1 511 characters. Although we have already set the column limit to 4 000 characters, the importing command was still failing because of the excessive length of the string in this column. We suppose that there is some inner limit for *PostgreSQL* that doesn't allow us to exceed the number of characters.

Source Code 3.20: Finding maximum length of the string

```
cat tmp.csv | \
awk 'BEGIN {FS="{"; OFS="{"; max=2000; total=0}
NR>1 {if (length($5) > max) {total+=1;} print length($5)}
END {print "Number of lines with description longer
than " max " chars is: " total}' | tail -n +1 | sort -nr | head
```

To avoid this problem, we have decided to remove the description column. It's not necessary to keep this data that occupy so much space and do not carry such important information. We will change the database structure to remove this column, then we will cut off that column in the data file and try to import these data once again, but the same problem comes up with the column `title`. There is probably some problem with string bounding. It's not necessary for us to have the asset's titles for all the languages, only a list of languages should be enough. Thus we also drop this column. After all these modifications, the import of all files finally succeeded.

After importing the data to the database we are able to set the constraints according to schema in Figure 3.4. For achieving our goal we will try to set the constraints without removing lines that do not meet our requirements at this time. If any of those alterings fail, we remove the wrong rows.

All constraints have been set successfully except the not null constraint for column `year` at the table `media_assetmetadata`. As long as it is the only useful information in the table, we had to remove all the null rows. This operation has removed 166 912 rows out of 208 986.

Now it is possible to connect the tables using foreign keys. We suppose the tables should be connected as described in schema in Figure 3.5 according to the columns' names. At first, we should make sure that all the foreign keys that tables contain are valid, but we will try to make references and if it fails, we will clear the data.

While setting these constraints there were (as expected) many unknown foreign keys that referenced table `media_asset`. These problems were probably caused by removing the rows from this data file, while importing into the database (there were inconsistent rows that we had to deal with). There were no other problems while creating references.

The last step is to create the last three tables in the database according to the schema in Figure 3.6. We will use the same approach – we create the tables without any constraint except primary keys, set the not-null constraints and finally we connect the tables using foreign keys. Creating of the tables will be done by Source Code 3.21.

Source Code 3.21: Creating tables – part 3

```
create table media_assetmetadata_rating_levels (
    id integer primary key,
    assetmetadata_id varchar(255),
    ratinglevel_id varchar(255)
);

create table viewership (
    id integer primary key,
    start_time timestamp with time zone,
    net_time bigint,
    user_id varchar(255),
    asset_id varchar(255)
);

create table media_user (
    uuid varchar(255) primary key
);
```

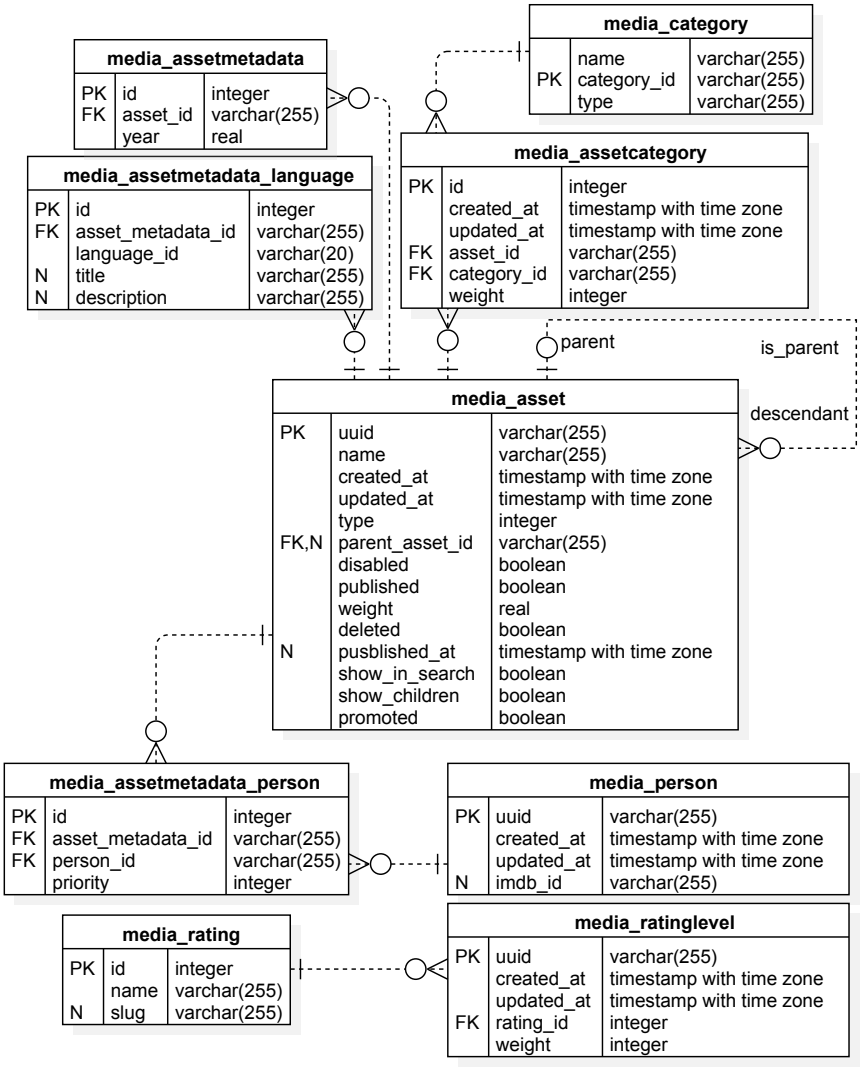


Figure 3.5: Relationships between tables

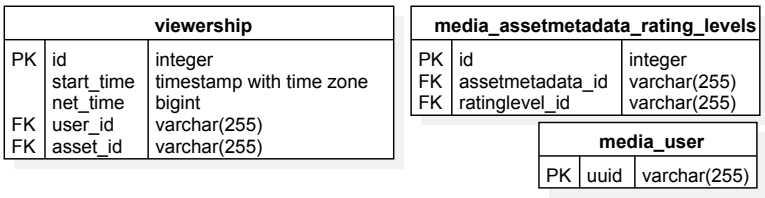


Figure 3.6: UML diagram of tables – part 3

It was necessary to delete some rows from table `viewership`, because not all of them met our not-null constraints. But there is a bigger problem with importing user's data since we do not have any. At least we export all the user's identifiers from `viewership` to the `media_user` table.

After all these steps we can finally create the whole database according to the schema in Figure 3.7, that contains all provided data and makes the foreign key constraints as described in the schema. Afterwards we are able to step forward to make further data analysis and select data from database that are needed by our algorithms.

3.3 Data Clearance

While importing the data to the database, there were some records with senseless values observed. It's really important for our algorithms to have valid data that we can rely on. It is really suspicious how many records had to be removed, because of this reason. In addition, we can only guess, which volume of the kept data is real. We went through the `viewership` only, because these data are the most important for our propose.

At the begining the `viewership` table contained 323 443 361 records and these columns:

id identifier,

start_time watching start time,

net_time watching duration,

user_id user's identifier,

asset_id asset's identifier.

We won't mention the unknown user's (asset's) identifiers. It is possible to find these records all over the provided data, but they were deleted during importing to the database. In this case it was 25 242 358 rows, that have to be deleted for this reason.

Another curious discovery was that in the column `net_time`, there are many records that contain *non-positive time of watching duration*. It is incomprehensible how these records could be included to the `viewership` table, since the user hasn't actually watched the asset even for a second. We won't deal with the other values of duration, because sometimes there can be really small or really big values including the case when the user watches an asset dozens of times with short duration – we consider that watched anyway.

We will remove all these records that contains non-positive value of watching duration from the database using the Source Code 3.22. There were 17 383 267 such records.

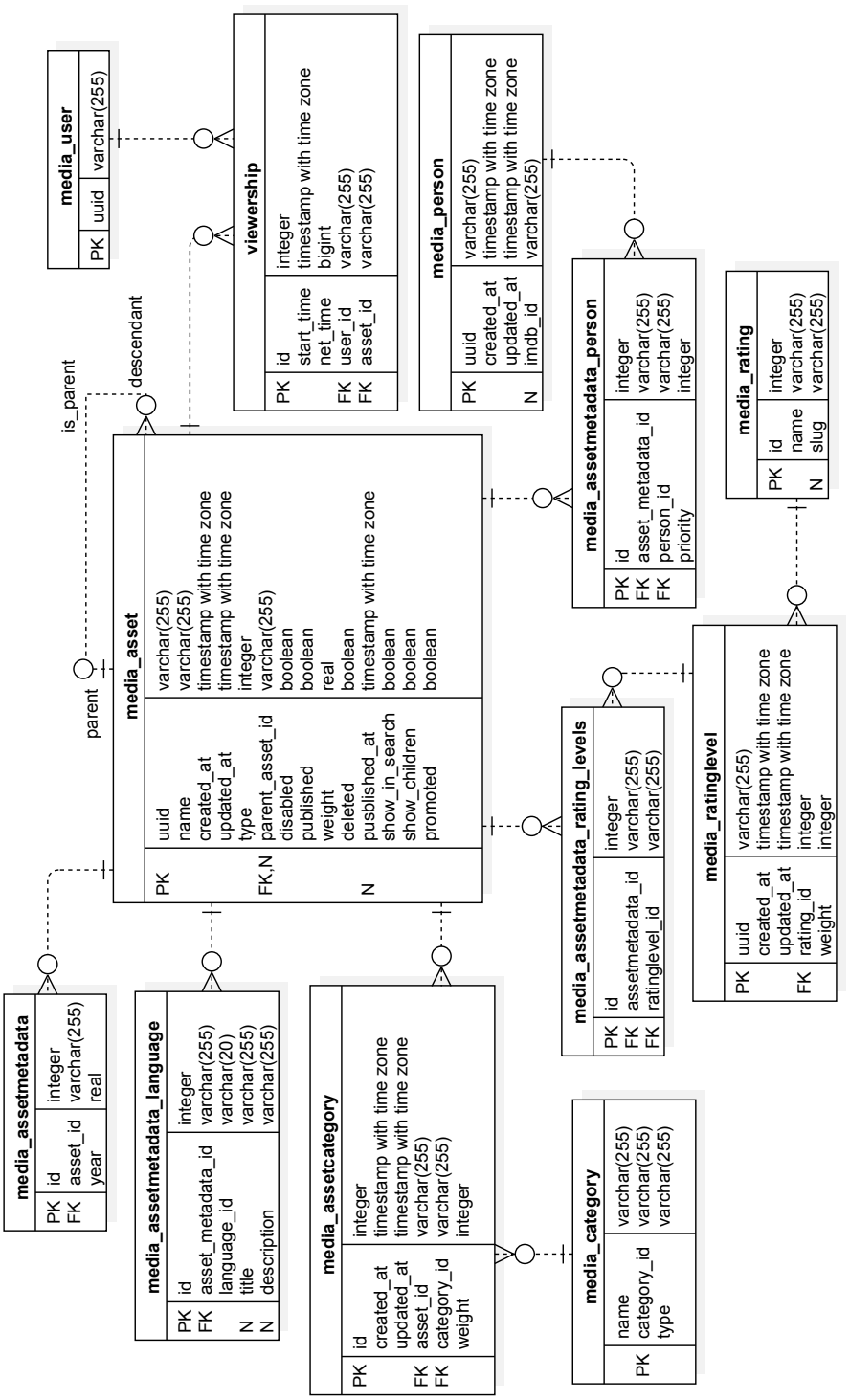


Figure 3.7: Database Schema

Source Code 3.22: Removing non-positive duration times

```
delete from viewership
where net_time <= 0;
```

Moreover, the `start_time` column contained startling values as well. Showmax company has lunched its online service in 2015, however in the data file, there are 301 records with time of start watching in year 1970 (probably it is related to C++ function `time`, which returns the number of seconds since the 1st January 1970) and 3 857 records with start time before year 2015.

For our purposes (to lower database size as much as possible without any impair) we only keep records from year 2017 till now. It won't affect our result and saves some disk space. It will be done using script in Source Code 3.23. In total we have removed 35 863 735 out of 320 245 756.

Source Code 3.23: Removing old records

```
delete from viewership
where start_time < '2017-01-01';
```

Chapter 4

Experiments

In this chapter, we make experiments with the provided data. We have already implemented algorithms introduced in the Section 2.4. For these implementations, we select data from the database according to required conditions and format. Afterward, we execute the algorithms with the given data. Then we visualize collected outputs, describe the results and compare it to our expectations. Lastly, we make a discussion about possible interpretations, what could affect the results and we also evaluate whether the algorithms are usable.

4.1 Data Selection

Proposed algorithms only require the viewership data, asset's data and user's data. We have prepared data in `viewership` table already, so it should be valid. Asset's data are ready as well. Only the data about users are not available. The only thing we have is user's identifier that we can extract from the `viewership` table.

Source Code 4.1: Selecting number of views

```
select user_id, count(distinct asset_id) as count
from viewership
group by user_id
order by count asc;

select asset_id, count(distinct user_id) as count
from viewership
group by asset_id
order by count asc;
```

To get more familiar with those data, using script in Source Code 4.1, we have extracted the data about how many assets the user has watched and how many users have seen the given asset from the database. These data have been

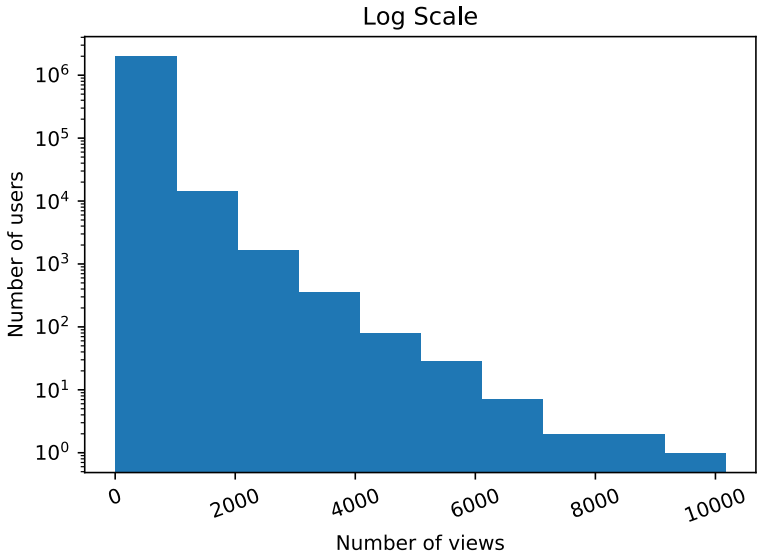


Figure 4.1: Log scaled distribution of users

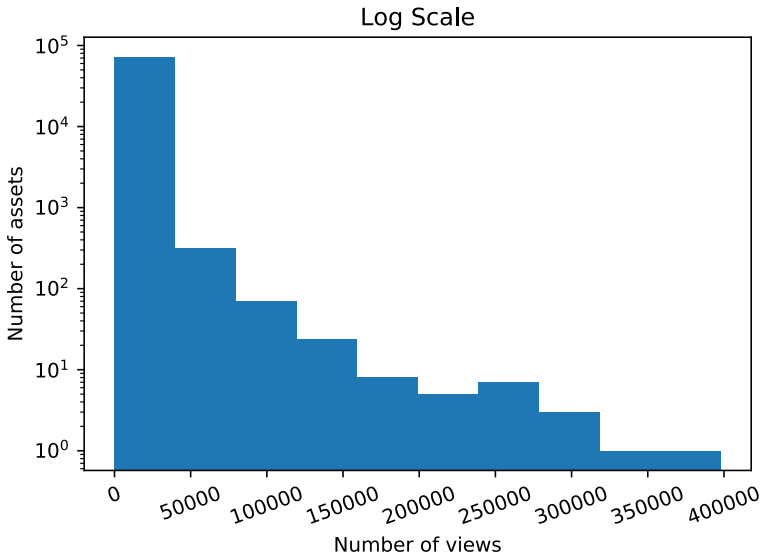


Figure 4.2: Log scaled distribution of assets

used for creating histograms describing distribution of *users* that watched the same number of *assets* (in Figure 4.1) and describing distribution of *assets* that were watched by the same number of *users* (in Figure 4.2).

From user's histograms it is visible that there is a great number of users that watched up to ten assets and only a few with greater activity. Those users are not useful for our algorithms as long as we need to select intervals in

Table 4.1: Number of views quantils

Type	Quantil	Value
users	0,5	21
users	0,9	231
assets	0,5	261
assets	0,99	28 291

their view-log. Another reason for removing these users is that some of them were created for testing purposes or massively generated by some attacking robots. Similar problem comes up with users that have seen almost all the assets – we suppose they have been created for testing purposes as well.

The same situation comes up with assets. There is a great number of assets, which were seen only by a few users. Number of views decreases exponentially (as at user's histograms). Assets which were seen by almost no users or which were seen by all the users won't bring any new information since we use item-based similarity.

According to these reasons we have decided to cut off the inactive and overly active users (assets). Using quantiles we cut off the left-half of users that watched less than 21 or more than 231 assets. Subsequently assets that saw less than 261 or more than 28 291 users as describes Table 4.1 were removed.

4.2 Visualization

Before executing the algorithms we wanted to see what the users are distributed like in the space based on what they watch. There might be some interesting distribution visible that might help us with selecting data for our experiments. For the visualization we have used the boolean IE matrix containing information about users – what assets they have seen. To be able to visualize the users, we had to reduce the dimension of the space they are situated in. This space has 35 686 dimensions.

We will apply one of the most widely used algorithms to reduce dimensionality – Singular Value Decomposition (SVD). To perform the reduction we use implementation in *Scikit-learn*[10] library for machine learning. To be able to display the users, we have to lower the number of features to 3-dimensional space.

After lowering the dimension, we select only a small sample from all the users because displaying all the 2 002 991 users in one graph with 3 dimensions would help us. The result of these actions is visible on the graph in Figure 4.3. The graph shows that there are a few clusters of users that are somehow diver-

[10] https://scikit-learn.org/stable/

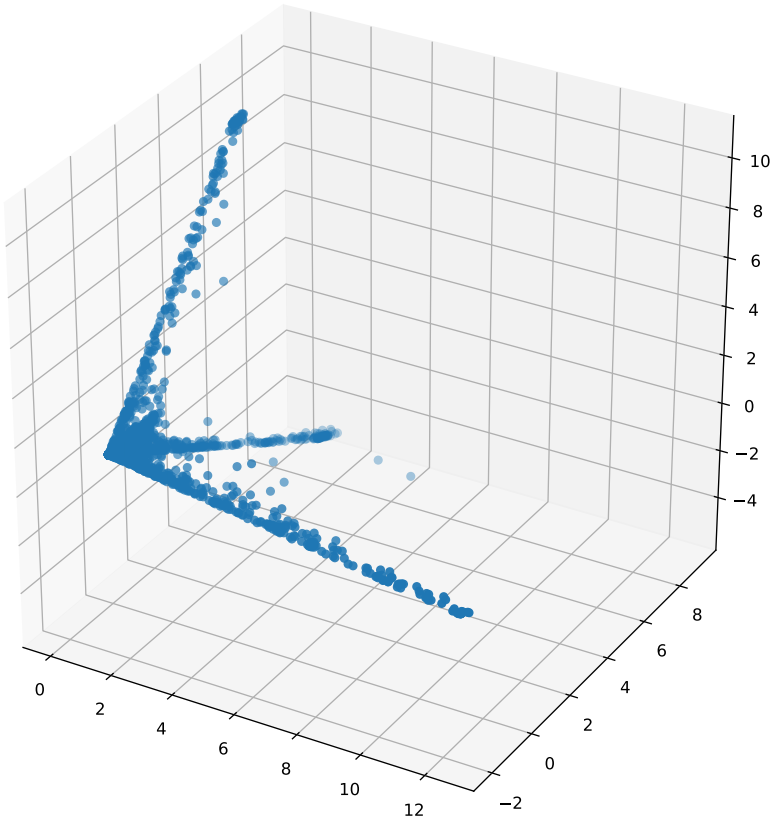


Figure 4.3: Users in 3D space

sified. To easily separate groups of users we have applied *K-Means* algorithm and colored the groups in the graph in Figure 4.4.

We have chosen a few representatives of each group and selected their view-logs from the database, but no breakthrough was made. The only difference between the groups visible by eye is that the users from red group watched in average much less assets then users from other groups. People in the same group watched very varying assets that were in some cases similar to the assets from a different group.

We have reduced the featured space to only 3 dimensions, which supposedly didn't bring any new information than was expected. SVD reduction is not the appropriate approach with a high number of dimensions, especially since it is only a boolean space.

4.3 Algorithms Setup

We have implemented all the designed algorithms and preprocessed provided data to be be in the desired format. Before the execution, it was necessary

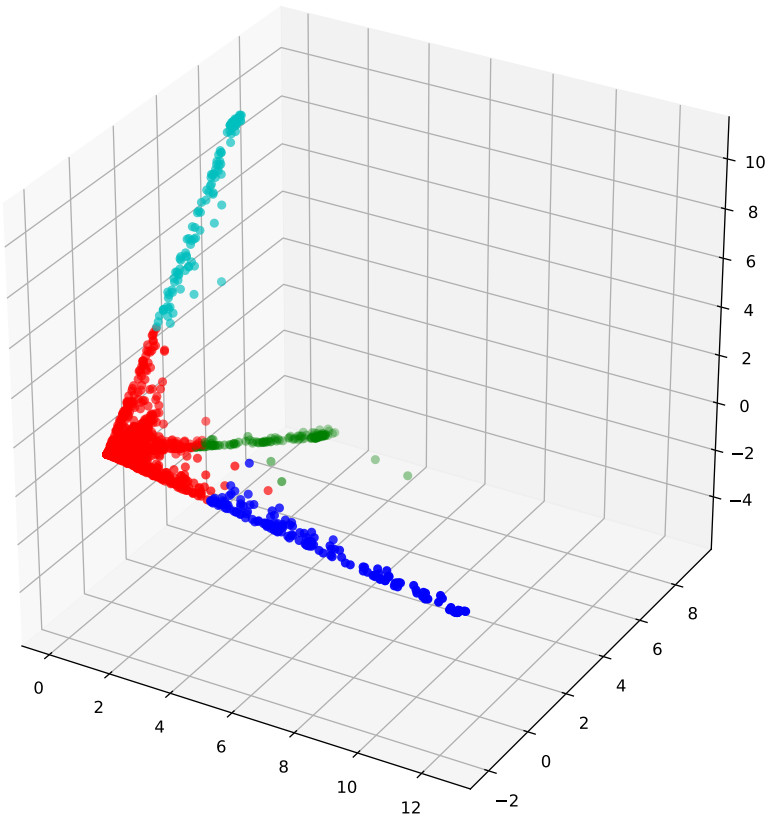


Figure 4.4: Colored users in 3D space

to carefully set hyperparameters, which may significantly affect the results. We have chosen the size of the interval for each modification to be 25 items. If the size is too small, it could cause a excessive decrease in the amount of preferences that a user can have. On the other hand, if the interval is too big, it may include preferences that are no more relevant Secondly, it takes a big part of the user's activity that can't be evaluated.

The second hyperparameter `top-k` had to be set for algorithms *Freeride*, *West Coast* and *Upper East Side*. It's better to set this parameter a little higher, especially in the cases with many assets. But the number of the selected most similar users/assets fundamentally affect the computing time, so we decided to choose a smaller amount.

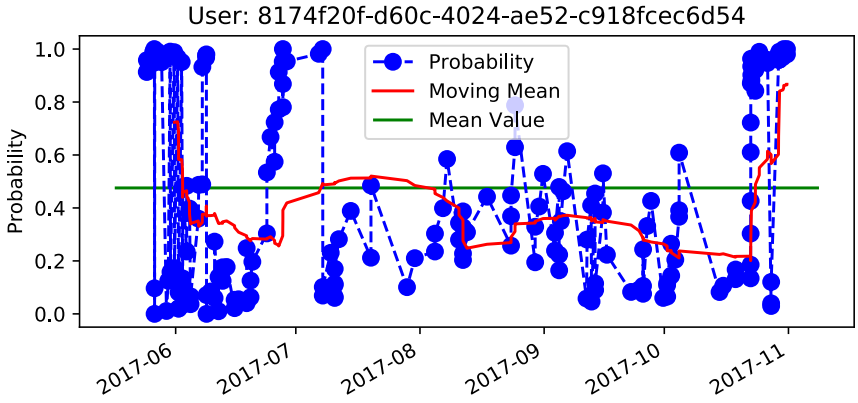
As some executions take a really long time, it was really important to select a user that would clearly demonstrate which assets fulfill his preferences and which would not. Another criterion was that the size of the view-log has useful results. Settings of hyperparameters for every algorithm are visible in Table 4.2.

Table 4.2: Hyperparameters

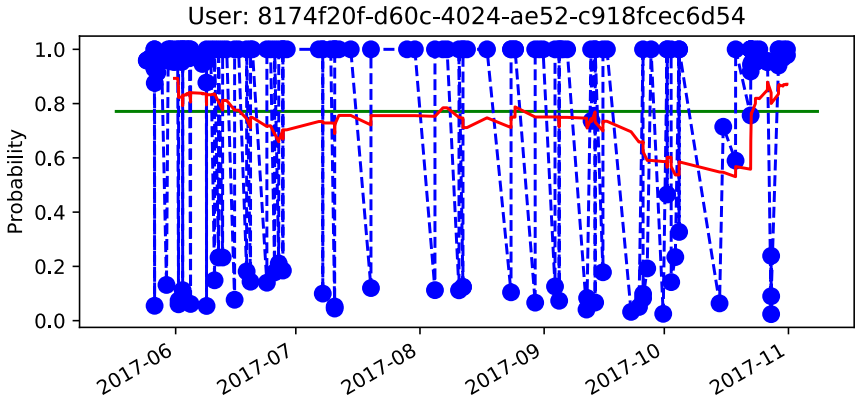
Algorithm	Size of Interval	Top-K
Freeride	25	100
Downtown	25	✗
West Coast	25	15
Upper East Side	25	100

4.4 Results and Discussion

To compare algorithms we have selected a single account from the database. This account has been processed by all the algorithms. We have gained the probabilities of watching every asset from user's view-log based on his historical preferences. The results of the algorithms are visualized in Figure 4.5.

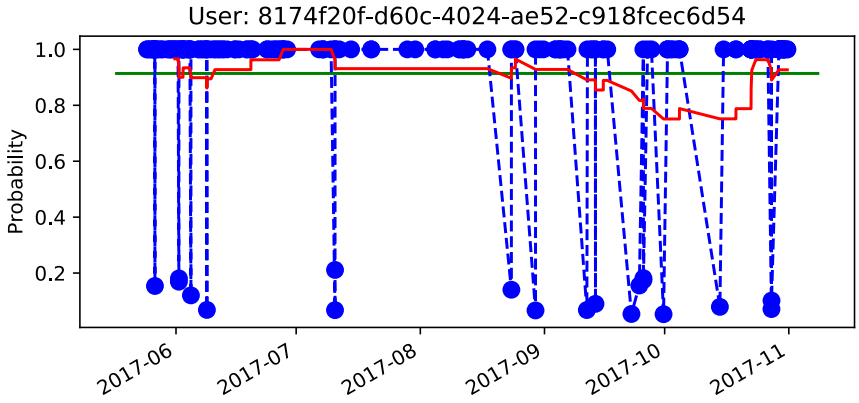


(a) Algorithm Freeride

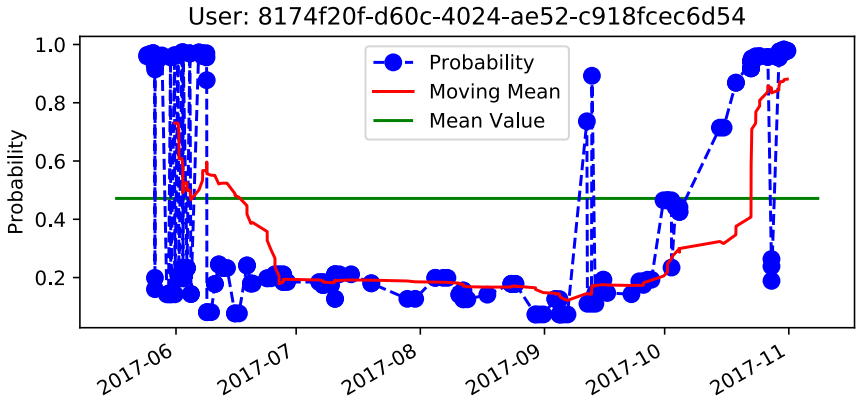


(b) Algorithm Downtown

Figure 4.5: The algorithms comparison – single account



(c) Algorithm West Coast



(d) Algorithm Upper East Side

Figure 4.5: The algorithms comparison – single account

The *Freeride* algorithm has quite a low value of probabilities – 0,48. The values of the predictions are relatively various as is visible in Figure 4.5a. From the view-log, it is visible that watching new assets has lower probability than watching episodes of one TV series. These results indicate that this user account may be shared. Very well are predicted TV series – their probability is constantly high. On the other hand, movies have lower values in general.

The *Downtown* algorithm has a much higher mean value than *Freeride* – 0,77 as is visible in Figure 4.5b. The probability of watching TV series is mastered well. Also learning of new preferences is very fast. In spite of these positives, almost every new asset (which is not in the interval) has a really low probability with no exception. From the results, user account is on the edge of being shared, but it can not be unequivocally proved.

The *West Coast* algorithm provides very good results in comparison to other algorithms. In Figure 4.5c it is visible that mean value of probability is

Table 4.3: Single account processing

Algorithm	Mean Value	Number of suspicious activity	Sharing Rate
Freeride	0,48	78	68 %
Downtown	0,77	45	0 %
West Coast	0,91	19	0 %
Upper East Side	0,47	112	54 %

very high – 0,91. New assets (not in the interval) are evaluated mostly with high probability and low values occur only rarely. Assets with the same genre have perfect rates. This algorithm also learns the new preferences very fast and supports a large range of them. According to the results, the account won't be considered shared.

The last algorithm *Upper East Side* evaluates the account to be shared. The graph in Figure 4.5d has really low mean value (almost the same as the *Freeride* algorithm – 0,47). TV series are rated correctly, but movies have a really low probability, even if it is the same genre. There is almost no learning of new preferences with watching the same asset multiple times. This knowledge suggests that the algorithm poorly evaluates the assets that were watched by only a few users. From the results, we would consider the account to be shared.

We have summarized all these information from single account processing in the Table 4.3. Activity with probability of watching lower than 25 % is considered to be suspicious – watching the given asset is highly improbable according to user's preferences.

To evaluate the account's *sharing rate* we have chosen the *moving mean* (MM). From the list of probabilities, we select a small sliding window and compute the mean value of the window's elements. This method balances the disposable low rated activities, but accounts that multiple personas use will be easily evaluated as the MM drops below a predetermined threshold. We have determined the size of moving window to be 20 items and the threshold for sharing 0,5. Sharing rate is then computed as ratio of activity while the MM is below threshold.

There is a relatively large range of the values as shown in Table 4.3. As *West Coast* indicates, the account is used by single user most of the time or there is really only one persona using this account that likes to experiment with assets that do not exactly fit his preferences. On the other hand, algorithms *Freeride* and *Upper East Side* indicate that this account is definitely shared by multiple personas.

To compare algorithms more objectively we have added fake records to the selected view-log that belongs to other various users. Then we executed all the

Table 4.4: Fake activity detection

Algorithm	Min	Max	Mean	Success Rate
Freeride	0	0,64	0,01	98,7 %
Downtown	0	0,86	0,07	92,3 %
West Coast	0	1	0,11	89,7 %
Upper East Side	0,09	0,99	0,79	11,5 %

Table 4.5: Account's rating

Description	The Best	The Worst
shared Activity	0 %	57,7 %
mean probability	92,4 %	46,6 %
view-log size	242	149
MM below threshold	0	86

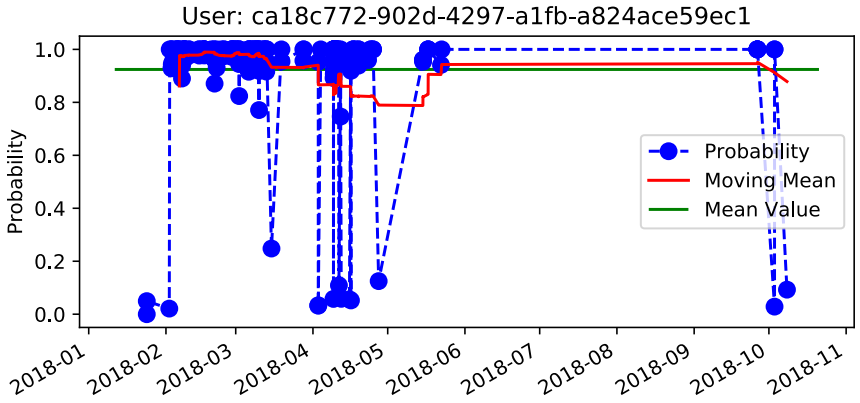
algorithms again to find out whether this activity will be rated with low probability. The Table 4.4, shows the lowest and the highest computed probability of all assets added to the view-log including the mean value and percentage of correctly marked records. The percentage of correctly marked results has been computed by countering the fake activity that has the probability of watching greater than 25 %.

The best algorithm that detects fake activity is *Freeride* with 98,7 % success, which is really good result that *proves the ability to detect account sharing*. The *Downtown* and *West Coast* also have very good results. Nevertheless, the *Upper East Side* algorithm has a surprisingly low rate, although the manual evaluation had quite good results.

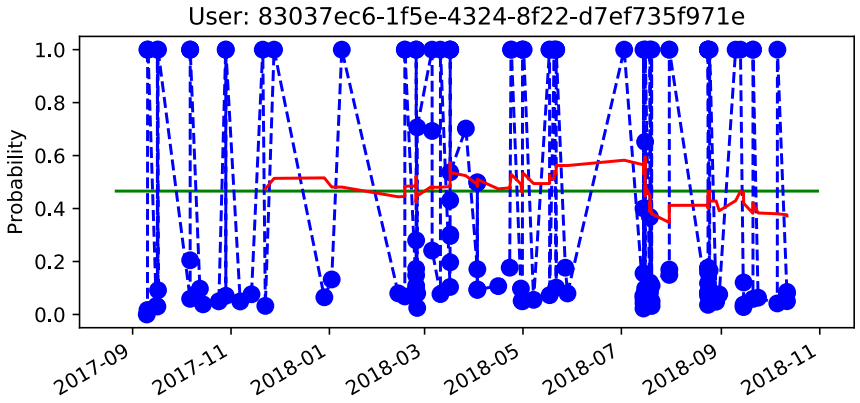
By contrast, there may be an issue that even activity that belongs to the single-person used account can be marked as suspicious with a low probability of watching. As long as it is an unsupervised algorithm, we can't prove, whether the activity is marked correctly. *In spite of these problems, as results of this test suggest, all the algorithms are able to detect activity that does not fit the users preferences.*

We used *Downtown* algorithm to process a sample of users and evaluate their accounts, whether they are shared. The account with the best activity was never evaluated as shared. In contrast to the worst account that was evaluated as shared in the 57,7 % of it's life-time. Going through the view-logs discovered that the best user watches episodes of TV series almost all the time. The worst account mix up almost all the genres available at the Shomax's library.

More accurate results of both accounts are displayed in Figure 4.6a (the best one) and in Figure 4.6b (the worst one). Further statistics from results can be found in the table 4.5.



(a) The best Account



(b) The worst Account

Figure 4.6: Comparison of the best and the worst account

It is really hard to measure the quality of the algorithms. The only demonstrably authoritative way is to evaluate view-logs of the users, that guarantee their accounts have not been used by any other persona. Must be emphasized that also watching a asset with another person influences preferences – the asset watched with children, won't occur in the users view-log, while watching with same-old friends or seniors.

Without any information regarding whether the account is shared or not, the most proving method to measure quality of the algorithms is to go through the view-log and evaluate established criteria:

learning speed Learns the algorithm new preferences fast?

TV series Have the episodes of one TV series high probability?

new preferences Have assets of the different genre low probability?

Table 4.6: Algorithms criteria

Criteria	Freeride	Downtown	West Coast	Upper East Side
learning speed		✓	✓	
TV Series	✓		✓	✓
new preferences	✓	✓	✓	✓
new assets			✓	✓
preferences range	✓		✓	✓

new assets Are there some new assets with high probability?

preferences range Supports wide preferences range?

Based on these criteria, we drew up the Table 4.6 that describes which algorithms fulfill them. As expected, we can see that only the *West Coast* algorithm fulfills all of them, so *we can confirm it provides valid data, that can decide whether the account is shared or not.* We have chosen the account to process very carefully, to be able to classify all these criteria. In accordance with these conditions, we consider the selected account not to be shared.

In addition, there might be some external influences that affected the results. As previously mentioned the data contained many senseless records that we had to remove, but there might be some others that weren't revealed by our analysis, because the data size is 36,92 GiB which is not a size that can be simply verified for errors.

Another aspect that impairs results is that we do not have any information about user accounts – some of them may be testing accounts, superior accounts or accounts that are able to watch on more screens at the time. Lastly, a great influence on the results is the continuous watching of a single asset by a user that is divided into more records by small intervals into viewership.

The algorithms can be used to evaluate whether the user has strict preferences that he does not step out of or has more versatile interests. For these users, recommendations can be more conservative and more successful. On the other hand, some users like experimentation and sometimes watch other genre assets. To these progressive users, assets that do not exactly match their preferences can be recommended.

Based on the data that algorithms provide, the activity with low probability (because it belongs to other persona) can be removed from the records that are used for recommendations. This act improves recommending, since recommendation system gets only relevant data for a given user.

As long as we can compute the mean value of probability or the amount of activity with low probability, we can mark some users‘ account as shared when the value exceeds established bounds. The knowledge about which accounts are shared can be used for commercial interests. Due to subscription fee for online streaming services, users share their accounts to avoid the fee or to split

the amount between more persons. These shared accounts can be restricted or the subscription fee can be increased, based on the results of algorithms.

Conclusion

In this thesis we study the problem that multiple users share a single account in online streaming services. We have researched works related to this problem. Subsequently we have designed and implemented algorithms capable of detecting shared accounts with the intention to increase accuracy of recommending systems. The Showmax company has provided us with real datasets on which we have performed our tests of developed approach.

The current online streaming services do not use shared accounts detection, although Netflix offers an option to split account manually to avoid the problem of recommending system efficiency reduction. Since there is a monthly subscription at these services, providers limit the number of devices to prevent account sharing.

Many papers find the shared accounts detection to be a challenging problem that may significantly influence the quality of recommendations. Most of them aim to identify a set of users that use the given account and identify user of newly incoming session.

Our goal is to detect accounts that are shared by multiple personas. We proposed an algorithm based on collaborative filtering with evaluating user consistency in sliding window and implemented a few modifications of the presented method. Using this method, we evaluate the account's activity, moreover we are able to decide whether is one of the shared accounts.

The results suggest that the algorithms can reveal multiple user's activity with high success rate and afterwards evaluate account sharing. Furthermore the manual evaluation of algorithms shows satisfying results.

The benefit of these algorithms is the implementation simplicity and high accuracy of identifying suspicious activity. Also the ability to preclude compromising the financial interests and performance improvement of recommendation system is crucial to use in production.

In the future works, we plan on testing the improvement rate of the proposed algorithms on accuracy of the recommendations at online streaming services and testing precision on guaranteed single-user accounts.

Bibliography

1. YANG, S.; SARKHEL, S.; MITRA, S.; SWAMINATHAN, V. Personalized Video Recommendations for Shared Accounts. In: *2017 IEEE International Symposium on Multimedia (ISM)*. 2017, pp. 256–259. Available from DOI: 10.1109/ISM.2017.43.
2. JIANG, Jyun-Yu; LI, Cheng-Te; CHEN, Yian; WANG, Wei. Identifying Users Behind Shared Accounts in Online Streaming Services. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. Ann Arbor, MI, USA: ACM, 2018, pp. 65–74. SIGIR '18. ISBN 978-1-4503-5657-2. Available from DOI: 10.1145/3209978.3210054.
3. WANG, Zhijin; YANG, Yan; HE, Liang; GU, Junzhong. User Identification within a Shared Account: Improving IP-TV Recommender Performance. In: MANOLOPOULOS, Yannis; TRAJCEVSKI, Goce; KON-POPOVSKA, Margita (eds.). *Advances in Databases and Information Systems*. Cham: Springer International Publishing, 2014, pp. 219–233. ISBN 978-3-319-10933-6.
4. WANG, Zhijin; HE, Liang. User identification for enhancing IP-TV recommendation. *Knowledge-Based Systems*. 2016, vol. 98, pp. 68–75. ISSN 0950-7051. Available from DOI: https://doi.org/10.1016/j.knosys.2016.01.018.
5. VERSTREPEN, Koen; GOETHALS, Bart. Top-N Recommendation for Shared Accounts. In: *Proceedings of the 9th ACM Conference on Recommender Systems*. Vienna, Austria: ACM, 2015, pp. 59–66. RecSys '15. ISBN 978-1-4503-3692-5. Available from DOI: 10.1145/2792838.2800170.
6. BELLET, Aurélien; HABRARD, Amaury; SEBBAN, Marc. A Survey on Metric Learning for Feature Vectors and Structured Data. *CoRR*. 2013, vol. abs/1306.6709. Available also from: http://dblp.uni-trier.de/db/journals/corr/corr1306.html#BelletHS13.

7. MELVILLE, Prem; SINDHWANI, Vikas. Recommender Systems. In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by SAMMUT, Claude; WEBB, Geoffrey I. Boston, MA: Springer US, 2017, pp. 1056–1066. ISBN 978-1-4899-7687-1. Available from DOI: 10.1007/978-1-4899-7687-1_964.
8. LUO, Shuyu. *Introduction to Recommender System* [online]. 2018 [visited on 2019-05-10]. Available from: https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26.
9. LUO, Shuyu. *Introduction to Recommender System* [online]. 2018 [visited on 2019-05-10]. Available from: https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26.
10. HANDARU. *Build Recommendation Engine Using Graph* [online]. 2016 [visited on 2019-05-15]. Available from: https://medium.com/tiket-com-dev-team/build-recommendation-engine-using-graph-cbd6d8732e46.
11. O'CONNOR, Kevin; ROSSUM, Guido van; PINARD, Franacois; HETTINGER, Raymond. *Heap queue algorithm* [online]. Version 2.3 [visited on 2019-05-10]. Available from: https://docs.python.org/2/library/heapq.html.
12. NO AUTHOR GIVEN. *CSV - Comma Separated Values* [online]. 2019 [visited on 2019-04-29]. Available from: https://datahub.io/docs/data-packages/csv#the-format.
13. ELLINGWOOD, Justin. *The Basics of Using the Sed Stream Editor to Manipulate Text in Linux* [online]. 2013 [visited on 2019-04-29]. Available from: https://www.digitalocean.com/community/tutorials/the-basics-of-using-the-sed-stream-editor-to-manipulate-text-in-linux.
14. NO AUTHOR GIVEN. *Cut Command in Linux with Examples* [online]. 2019 [visited on 2019-04-29]. Available from: https://www.geeksforgeeks.org/cut-command-linux-examples/.
15. KUHN, Markus. *A summary of the international standard date and time notation* [online]. 1995 [visited on 2019-05-06]. Available from: https://www.cl.cam.ac.uk/~mgk25/iso-time.html.
16. GOYAL, Anshika; NEGI, Praveen. *AWK command in Unix/Linux with examples* [online]. 2019 [visited on 2019-04-29]. Available from: https://www.geeksforgeeks.org/awk-command-unixlinux-examples/.
17. JOHNSON, C.; VARMA, J. *Pro Bash Programming, Second Edition: Scripting the GNU/Linux Shell*. Apress, 2015. ISBN 9781484201213. Available also from: https://books.google.cz/books?id=zlMnCgAAQBAJ.

18. MARZ, Nathan; WARREN, James. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. 1st. Greenwich, CT, USA: Manning Publications Co., 2015. ISBN 1617290343, 9781617290343.

19. IBM; ZIKOPOULOS, Paul; EATON, Chris. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. 1st. McGraw-Hill Osborne Media, 2011. ISBN 0071790535, 9780071790536.

20. ROUSE, Margaret. *Hadoop Distributed File System (HDFS)* [online]. 2018 [visited on 2019-05-07]. Available from: `https://searchdatamanagement.techtarget.com/definition/Hadoop-Distributed-File-System-HDFS`.

21. ROUSE, Margaret. *relational database* [online]. 2018 [visited on 2019-05-07]. Available from: `https://searchdatamanagement.techtarget.com/definition/relational-database`.

22. OBE, Regina O.; HSU, Leo S. *PostgreSQL: up and running*. Sebastopol: O'Reilly, 2012. ISBN 1449326331;9781449326333;

Appendix A

Acronyms

CF Collaborative Filtering

CSV Comma Separated Values

HDFS Hadoop Distributed File System

IE Interaction Embedding

ML Machine Learning

MM Moving Mean

RDBMS Relational Database Management System

SQL Structured Query Language

RS Recommender System

SVD Singular Value Decomposition

Appendix B

Contents of Enclosed CD

- `readme.txt` ..... file with contents description
- `src`
  - `impl` ..... implementation source codes
  - `thesis` ..... directory of LaTeX source codes of the thesis
- `BP_Vopat_Tomas_2019.pdf` ..... thesis text in PDF format