University of Padova

Department of Mathematics

Master Thesis in Data Science

Collaborative filtering for recommender systems with implicit feedback

Supervisor
Gian Antonio Susto
University of Padova

Co-supervisors
Chiara Masiero, Luca Brunelli
Statwolf Data Science

Master Candidate
Massimiliano Conte

Academic Year
2021-2022

To Anna, who is always close to me. To my parents, who supported me in my studies. To all the friends with whom I have shared moments of fun.

Abstract

Recommending the right products to the customers can significantly increase the sales of an e-commerce, and the presence of huge amounts of transactional data makes data-driven solutions the best choice for the recommender systems in many circumstances. In this work, a general overview of the recommendation task is given, then several data-driven methods are compared on a real world company data. In particular, the effort is centered around implicit feedback, i.e. binary data such as sales, and collaborative filtering, that is the usage of community behavior in the suggestions computation. Finally, different ways to handle cold starts, that are new customers, are discussed and compared.

Contents

Listing of figures

Listing of tables

Listing of acronyms

ALS . . . . . Alternating least squares

EASE . . . . . Embarassingly shallow autoencoders (reversed)

NMF . . . . . Non negative matrix factorization

SVD . . . . . Singular value decomposition

1

Introduction

Recommender systems intervene in the everyday life of most of the digitized population. Instagram stories, Youtube videos, Netflix series, Facebook friends, Google news, Spotify songs, and Amazon products are all proposed to users through a recommendation system. Such engines can make people discover products they like that otherwise would never be found without an external suggestion. However, the consumption of content proposed by recommender systems can lead to feedback loops, which can degenerate into echo rooms and filter bubbles. The user is offered content concerning only a single area, fueling his interest in the latter, ending up hiding other types of content. This leads the user to consume only this type of content, making the system believe that this user is only interested in this area, fueling this type of recommendations even more [1]. With the growing usage of e-Commerce, large amounts of users, items, behavioral, and interaction data are made easily accessible, leading to a strong economic interest in the technology of recommendations. The presence of such data, available for the majority of companies operating on the Web, has contributed to the arrival and evolution of data-driven recommender systems.

1.1 Related fields

Recommender systems can be seen as an instance of the wide field of information retrieval [2]. Given the importance of web search engines, the most famous information retrieval application, recommender systems inherit some structure from their literature, such as part of the evaluation criteria.
Since most of the methods used in order to build recommendation engines are data driven, other strongly related fields are statistics and machine learning, in particular regression, classification and latent space analysis.
The recommender systems framework is built on top of the machine learning and search engines frameworks. Basically, the task of suggesting items is considered a ranking problem, where the products are ordered by affinity with the user. In the analogy with search engines, the user corresponds to the query, while the items correspond to the documents.

1.2 Context

This work arises from the necessity of building a recommender system for a company specialized in the sale of luxury clothing and accessories. For this reason, the focus is on methods that are well suited to such a context and available data. Such data comprehend user purchases of the last two years, demographic features about the users, and content features that describe the items. Given the low quality of user and item descriptions, due to the strong presence of missing or incorrect entries, the focus of the work is mostly on pure collaborative filtering, which uses only user past purchases to make the recommendations, rather than on using hybrid approaches that also take advantage of user and product features.

2

General framework

2.1 Implicit and explicit feedback

The peculiar data in recommender systems are the interactions between user and item. These interactions can be the of various types: consumption of content, five-stars review, purchased or not, time spent over the item, clicked or not, etc. The most important distinction is between implicit and explicit feedback, dividing whatever the user explicitly provided a rating, or if the feedback is automatically recorded from the user's behavior. For example, a five-stars review of a product is an explicit feedback, because the user consciously decides the rating. In contrast, the fact that a user clicked on a product webpage is an implicit feedback, because it is an indirect manifestation of interest in the product. However, in literature, implicit feedback frequently means the presence of dichotomous interactions, while explicit feedback means the presence of continuous or multimodal interactions. This distinction make a huge difference in how the data is treated, in particular how the missing values are handled. The problem with implicit feedback is that the interactions are in the form of presence or absence of the feedback, so considering only the actually recorded interactions would mean working with

a unary variable, which is incompatible with learning.
Implicit feedback represents an important scenario because product purchases, whose economic value is well known, are the most relevant form of implicit feedback. It can be seen as a positive feedback when a user buys an item, making the assumption of the fact that a user bought one item means that such user actually likes that item.

2.2 Framework

2.2.1 Notation

Let $\mathcal{U}$ be the set of n distinct users, and $\mathcal{I}$ be the set of m items in the catalog. Let $\mathcal{I}_u$ be the set of items bought by the user u, and $\mathcal{U}_i$ be the set of users that bought the item i. In this way, $\mathcal{I}_u \cap \mathcal{I}_v$ denotes the set of items bought by both the users u and v.
The superscript $(train)$ indicates that the referred object is obtained from the train set. For example, $R^{(train)}$ indicates the interaction matrix obtained from the train set of interactions. The same applies to $(test)$. The process of splitting the dataset is discussed in Section 3.2

2.2.2 Rating or Interaction data

The main source of data in recommender systems is usually the set of interactions, which are user-item pairs, possibly weighted: $T = \{(u, i, w) \in \text{interactions}\}$. Such relational data can be described in the rating/interaction matrix $R \in \mathbb{R}^{n \times m}$, defined as:

$$r_{ui} = \begin{cases} w_{ui} \text{ if } (u.i) \in T \\ \text{NA otherwise} \end{cases} \tag{2.1}$$

(NA values are still real values, but unknown).

For implicit feedback data $w_{ui} = 1 \; \forall \; (u, i) \in T$. The unary nature of the rating requires to not ignore the missing values, so for implicit feedback data, the definition of the interaction matrix needs to be sightly different:

$$r_{ui} = \begin{cases} 1 \text{ if } (u.i) \in T \\ 0 \text{ otherwise} \end{cases} \tag{2.2}$$

The relational nature of this kind of data makes untrivial the usage of the well-known machine learning models because it is not possible to assume that the observations, that here are the interactions, are independent and identical distributed, an assumption that is almost everywhere in machine learning and statistical modeling.

2.2.3 Content and demographic data

Content data refer to the features of the item. Demographic data, instead, are the data about users, that usually have the standard ready-to-use matrix structure, but it could not be available due to the fact that collecting data about people is not always feasible. Possible user features can be gender, age, nationality, etc., but of course the actual variables also depend on the domain of application.
Item data usually includes text description, so natural language processing can be an important part of the content-based approach. The other item features heavily depend on the domain application, they can be images, categories, etc.

2.3 Formalization of the problem

2.3.1 Regression/Classification perspective

From a machine learning perspective, the problem can be formalized as the prediction of the unseen interactions weights. For example, with 5-star ratings, the problem becomes to predict the missing ratings, using a regression or a multimodal

classification. With explicit feedback, the performance of the different methods can be accessed by computing one of the many well-known prediction losses, over an unseen test set of observed ratings.
For implicit feedback, such evaluation cannot be done without considering the missing entries. If not, a model that always predicts 1 would achieve a perfect score. By setting the missing interactions as 0, all the unseen entries are described as implicitly describe a. For this reason, computing reconstruction losses in the interactions would lead to a great error, since some unseen positive entries are labeled as negative.
However, in most implicit feedback cases, it is unavoidable to use this representation, at least for the training phase of the method.

2.3.2 Ranking perspective

In the application of recommender systems, what is crucial for the business is the relevance of the first recommended items. Besides the exact prediction of missing ratings, the interest is in the ordering of the items by affinity with the user tastes. Although most of the methods perform learning within the classification/regression framework, there are also some methods that directly attempt to optimize the ranking of relevant items [3][4]. Regardless of how the training phase is conducted, the evaluation process should consider the recommendations as a ranking problem. In this way, the accessed performance of the methods reflects the quality of the recommendations in a production environment. Therefore, the prediction of a system, for one given user, is an ordered list of items, which can be described as a sequence $(item_1, ..., item_m)$, whose first value is the most congenial item to the user, and so on. Each recommendation can be relevant or not, and it is up to the metrics to weight relevance and position in the sequence.

2.4 Challenges

2.4.1 Sparsity

Different levels of sparsity in the interaction matrix make the recommender systems perform differently [5]. Of all possible interactions $(u, i) \in \{0, 1, ..., n\} \times \{0, 1, ..., m\}$, generally only a very small subset is observed. This is true in almost every domain application, but is enhanced with implicit feedback. The reason is because it is likely that an user buys only one or anyway a few more products. For example, if the domain application is an e-Commerce of shoes, most of the users would probably need to buy exactly one pair of shoes (assuming they are buying for necessity).
The presence of sparsity makes it difficult to learn a user profile from few interactions. Moreover, the set of commonly purchased items between two users $\mathcal{I}_u \cap \mathcal{I}_v$ is likely to be empty, making the similarity computation between users also sparse.

2.4.2 Popularity bias

The popularity bias is the phenomenon for which most interactions concern a very small subset of famous items. The catalog can be divided into 3 groups:

- **Short head**, the most popular items;
- **Long tail**, the items with just a few interaction;
- **Distant tail**, the item with almost no interactions, which are usually not considered.

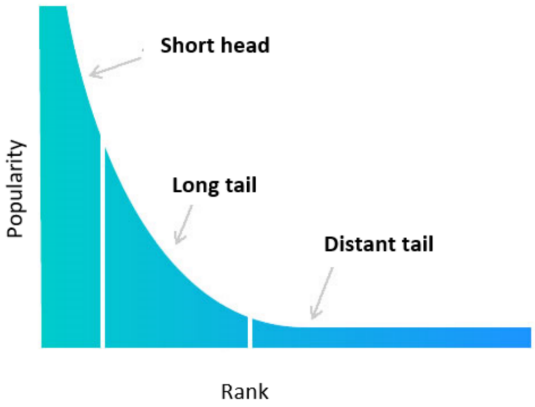

Figure 2.1: Illustration of popularity bias, from [6].

Users are likely to consume only, or at least first, the top products [6]. This makes many methods, which optimize accuracy-related metrics, recommend only such small subset of products. This is because learning from past interaction means that most methods learn to predict past interactions, where there is no influence of the recommendation in the user behavior.
Even if an unpopular item is relevant to one given user, it is probable that such a user never saw that item, hidden by the top products. Learning by the past interaction, with accuracy related metrics, penalizes the methods that try to make that meaningful recommendation, rewarding instead the suggestions of top items, that are likely to be in the past interactions of the user, even if they are less aligned with the user tastes.
Moreover, a system that suggests only the top products could not accomplish to sell more (in case of transactional data), because such items are likely to be known and purchased anyway. However, there can be scenarios where this behavior is welcomed, but this consideration depends on the domain of application, and knowing the existence of this phenomenon leads to greater awareness in the choices.

2.4.3 Cold start

The cold start is a well-known problem in recommender systems [7]. New users, unseen during the training phase, are cold start cases. Such cases imply the absence of collaborative (interaction) data, leaving only the user features as the available information that can help to personalize the recommendations. The same applies to new items in the catalog, where only the item metadata can discriminate the recommendation of that item to one user or another. It is obvious how collaborative filtering suffers more than content-based methods from cold start, in particular cold start users.

2.5 Paradigms

There are many suitable approaches for the construction of a recommender system [8].

- Content-based methods use the data about the items in order to build the suggestions. A machine learning model can be used to predict the probability that a given user buys a given item, or a similarity measure between items can be used to suggest items that are similar to the previously purchased ones [9].

- Demographic filtering uses the data about users, such as gender, age, location, etc. in order to build the suggestions [10].

- Collaborative filtering methods, instead, make use of the interaction data in order to build the suggestions. By doing that, all the users behavior could influence the recommendations for one single user. The easiest example of a collaborative filtering method, for suggesting items to a given user, is to select a neighborhood of similar users (people who have similar tastes with the target user) and suggest what such neighborhood liked the most [11].

- Knowledge-based recommender system are based on a set of logic rules and hard constraints, usually created from user specifications or domain knowl-

edge. Such systems are useful when sufficient data are not available, for example, with very expensive luxury goods [12].

However, most large-scale recommender systems are based on a hybrid system [13], using all the information available, data regard the users, items, transactions, context, and domain knowledge .
More specific approaches do exist, but they are related to a small field of application and are adaptations of the more general paradigms listed above.

3

Recommender Systems Evaluation

This chapter explains the evaluation process, mostly based on implicit feedback and the ranking perspective described in section:2.3.2. The fact that the evaluation is based on the ranking framework does not mean that the methods could not be based on regression or classification, but just that an eventual regression/classification output should be converted into a ranked list, for example ordering the items by descending predicted scores.
The desired characteristics of a recommendation engines are multiple. A set of good general qualities is listed, together with the appropriate technical tools that enable the offline examination of such qualities. Some of them apply to all recommender systems, while some others can be useful depending on the considered scenario.
The possible choices during the test-train split of the dataset are discussed.

3.1 Offline evaluation

Recommender systems work in a dynamic environment. The engine proposes some items, and the user decides whether to consume or not such items. The

active reaction of the customer to the system output enables a fair evaluation of the goodness of the recommendations. In general, A/B testing related procedures are used, and can effectively test the impact of the recommendations on the users. Other than that, testing a wide variety of methods with online procedures may be too time consuming. However, such online evaluation is not used in this work, due to the practical unavailability to do that in the experiments. The selection of few candidate methods for the system is made, thanks to the offline evaluation. The procedure uses a set of historical interactions to test the quality of the suggestions. Each method is trained on a test set, and then it computes a recommendation list for each test user. Finally, such recommendation lists are compared to the unseen historical interaction. This evaluation framework is similar to the classical machine learning one, but it presents some additional unfairness problems.

3.1.1 Problem shift

The procedure tests whether the system is capable of suggesting items that would be consumed without the suggestion, since the data was retrieved without the intervention of the recommender. This means that the chosen quality metrics does not directly measure the true effectiveness of the system, that is the ability to suggest items that the user would bought thanks to the suggestion. The metrics are still useful, but they refer to a different, and related problem. Hopefully, having good performances in the offline evaluation goes hand in hand with having good performances in the fair online evaluation.

3.1.2 Unfairness due to popularity bias

Users interacts only with items they know exist. This means that even a very good recommendation could be negatively evaluated, because the user never saw that item, and thus there is not such interaction in the historical data. This problem is heavily amplified by the popularity bias, because users know only a small subset of famous items, and so the recommendation of less popular items is discouraged

by the offline evaluation. This consideration is important because the recommendation of unpopular products is a key task in many business domains.

3.2 Test train split

There are more ways to split the data set. One common way is to pick one interaction per user, possibly the last one, and set those interactions as the test set. If cold start cases are unwanted in the test set, they can be avoided by selecting only the users with enough purchases. This approach has the limitation of one test interaction per user, which causes some metrics to collapse to others, such as the recall that becomes the hit to ratio.
Another strategy is to split by timestamp, i.e. setting the last observed interactions as the test set. The idea is that, in a real environment, only past interactions are known. The problem with this approach is that, in many recommender system application domains, users are likely to consume all their lifetime consumed items in a short period of time. This means that the chronological split would set, for most users, all their interactions together in the train or the test set, depending on the time at which such a user was active.
To prevent such phenomenon, in the domains where it could happen, a completely randomly chosen test set of interactions is preferable. Time information is lost, but a more complete set of situations is represented, so more aspects of the recommender system can be tested.

3.3 Relevance

Suggesting relevant items is indispensable; a system that does not satisfy this property is useless. The relevance can be measured in many different ways; in fact, all suitable machine learning and information retrieval losses are different ways of inspecting the relevance of the recommendations.

3.3.1 Precision related measures

The precision is the accuracy of the first items recommended. For a given user, the inspected method produces the ranked list of items. Only the first k are considered. Usually, the choice of k coincides with the number of items proposed in the actual usage of the system. If this information is not known, the standard choice is $k = 10$.
For one user and the corresponding predicted rank list, the precision at k if defined as:

$$P@k(u) = \frac{1}{k} \sum_{i=1}^{k} rel_u(i) \tag{3.1}$$

Where $rel_u(i) = 1$ if user u bought (in the test set) the item recommended in position i, and $rel_u(i) = 0$ otherwise.

Averaging this for all test users, the (mean) precision along all the users is obtained:

$$P@k = \frac{1}{n} \sum_{u \in \mathcal{U}^{(test)}} \left(\frac{1}{k} \sum_{i=1}^{k} rel_u(i) \right) \tag{3.2}$$

This metric gives the same importance to all the top k suggestions, regardless of their position within the top k. An extension that weights more the first items is the average precision:

$$AP@k(u) = \frac{1}{k} \sum_{i=1}^{k} P@i(u) \cdot rel_u(i) \tag{3.3}$$

Averaging this for all the test users, is obtained the mean average precision:

$$mAP@k = \frac{1}{n} \sum_{u \in \mathcal{U}^{(test)}} \left(\frac{1}{k} \sum_{i=1}^{k} P@i(u) \cdot rel_u(i) \right) \tag{3.4}$$

Truncation

Since most users interacted with a very small set of items, it is likely that the precision at 10 cannot achieve high values. For example, if a user has only one test interaction, a very frequent case, the best possible $P@10(u)$ is 0.1. To make it possible for $P@k$ to reach 1, an adaptive number of recommendations is considered, where k becomes the cap. In particular, we consider $min(|I_u^{(test)}|, k)$ recommendations, so a number of recommendations equals to the number of test interactions, at most k. Given the usual distribution of implicit interactions, it could be expected that most of the time such correction considers only the first 1 or 2 recommendations. This kind of correction is proposed, for example, in a well-known implicit feedback recommender system competition*.
The truncated precision, for one user, is:

$$P@k(u)_{corrected} = \frac{1}{min(|\mathcal{I}_u^{(test)}|, k)} \sum_{i=1}^{min(|\mathcal{I}_u^{(test)}|,k)} rel_u(i) \tag{3.5}$$

The truncated mean precision is obtained by averaging over the test users:

$$P@k_{corrected} = \frac{1}{n} \sum_{u \in U^{(test)}} \left(\frac{1}{min(|\mathcal{I}_u^{(test)}|, k)} \sum_{i=1}^{min(|\mathcal{I}_u^{(test)}|,k)} rel_u(i) \right) \tag{3.6}$$

The truncated average precision, for one user, is:

*https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations/overview/evaluation.

$$AP@k(u)_{corrected} = \frac{1}{min(|\mathcal{I}_u^{(test)}|, k)} \sum_{i=1}^{min(|\mathcal{I}_u^{(test)}|,k)} P@i(u) \cdot rel_u(i) \tag{3.7}$$

Finally, by averaging the $AP@k(u)_{corrected}$ over all the test users, the truncated mean average precision is obtained:

$$mAP@k_{corrected} = \frac{1}{n} \sum_{u \in \mathcal{U}^{(test)}} \left(\frac{1}{min(|\mathcal{I}_u^{(test)}|, k)} \sum_{i=1}^{min(|\mathcal{I}_u^{(test)}|,k)} P@i(u) \cdot rel_u(i) \right) \tag{3.8}$$

3.3.2 Recall

The recall measures the proportion of user interactions, in the test set, that the system correctly suggests within the top k recommendations. The recall, for one user, is defined as:

$$Rec@k(u) = \frac{1}{|\mathcal{I}_u^{(test)}|} \sum_{i=1}^{k} rel_u(i) \tag{3.9}$$

As for precision, the average along all the test users is:

$$Rec@k = \frac{1}{n} \sum_{u \in \mathcal{U}^{(test)}} \left(\frac{1}{|\mathcal{I}_u^{(test)}|} \sum_{i=1}^{k} rel_u(i) \right) \tag{3.10}$$

Truncation

As in precision, a similar argument can be made with recall, changing the number of considered recommendations to $max(|\mathcal{I}_u^{(test)}|, k)$, in order to allow each ranked list to cover all the interaction. However, with the typical number of test

interactions per user, which is very small, $max(|\mathcal{I}_u^{(test)}|, k)$ would be k for almost every user, with a standard choice of k. For this reason, the corrected version is not used.

3.3.3 Precision-recall plot

Precision measures the accuracy of the recommendations. Recall measures the ability to suggest all the relevant items. There is a trade-off between precision and recall, with smaller recommended lists (small values of k), the precision tends to grow and the recall tends to drop, while for larger recommended lists (large values of k), the recall tends to grow and the precision tends to drop. A broad relevance investigation can be performed with the precision-recall plot. For every recall point, i.e. $k = 1, \ldots, m$, $P@k$ and $rec@k$ are computed, and then the trend is displayed on a scatterplot.

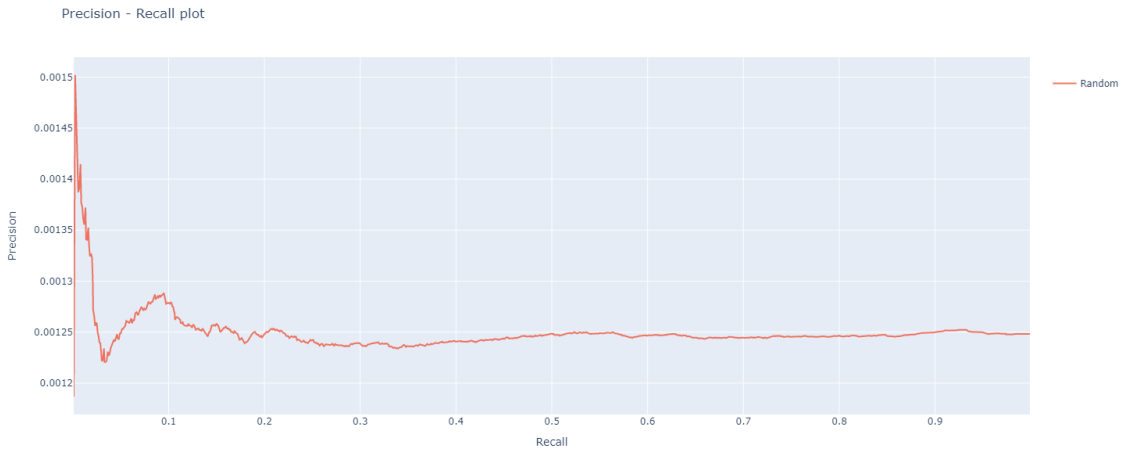

Figure 3.1: Precision-recall plot for a random predictor.

A possible metric that summarizes this plot is the AUC (area under the precision recall plot). However, using the AUC results in the loss of information on where the models are performing, since this metric considers all of each recommended list.

3.3.4 Hit ratio

This metric is the proportion of users for whom at least one relevant recommendation is given, within the top k. Originally, this metric was developed for the test splitting, where there is exactly one test interaction per user. In that case, the name refers to the proportion of "hitted" users. With the completely random split chosen in this work, these metrics are still useful, since for an e-commerce recommendation system it is interesting to known for how many people at least one potential selling item is proposed. This is especially true in domains where users tend to buy only one item at a time.

$$HR@k = \frac{1}{n} \sum_{u \in \mathcal{U}^{(test)}} min\left(\sum_{i=1}^{k} rel_u(i), 1 \right) \tag{3.11}$$

3.3.5 Expected percentile ranking

Sometimes called mean percentile ranking, it measures how far relevant recommendations are from the top of the ranked list. This metric, different from the previous ones seen, uses all the ranked list, without stopping at k. This makes it show how the system is performing after the top recommendations, but it is also more unstable due to the fact that one bad ranking for a relevant item has a strong impact on the score. Moreover, we are usually interested in only the first recommendations.

$$\overline{rank} = \frac{\sum_{ui} rel_u(i) \cdot rank_{ui}}{\sum_{ui} rel_u(i)} = \overline{\{rank_{ui} :\ rel_u(i) = 1;\ \forall u \in \mathcal{U}^{(test)}, \forall i \in \mathcal{I}\}} \tag{3.12}$$

Where $rank_{ui}$ is the quantile of the item i in the recommended list for user u.

One good property is that a random predictor is expected to have a $\overline{rank} = 0.5$,

since relevant recommendations are distributed uniformly in the ranked list.

3.4 Coverage

A good recommender system should be able to suggest all the items of the catalog. The coverage metric is the proportion of items that are suggested at least once, for a given set of users.
Let T_u represent the top k recommended list of items for user u:

$$CC@k = \frac{|\bigcup_{u \in U^{(test)}} T_u|}{m} \tag{3.13}$$

3.4.1 Frequency of recommendations plot

Having a very large set of test users is likely to make the coverage metric converge to 1. For this reason, a deeper understanding of system coverage can be achieved by plotting the frequency of recommendation for each item, within the top k recommendations, along the test users. Note that items are ordered by popularity.

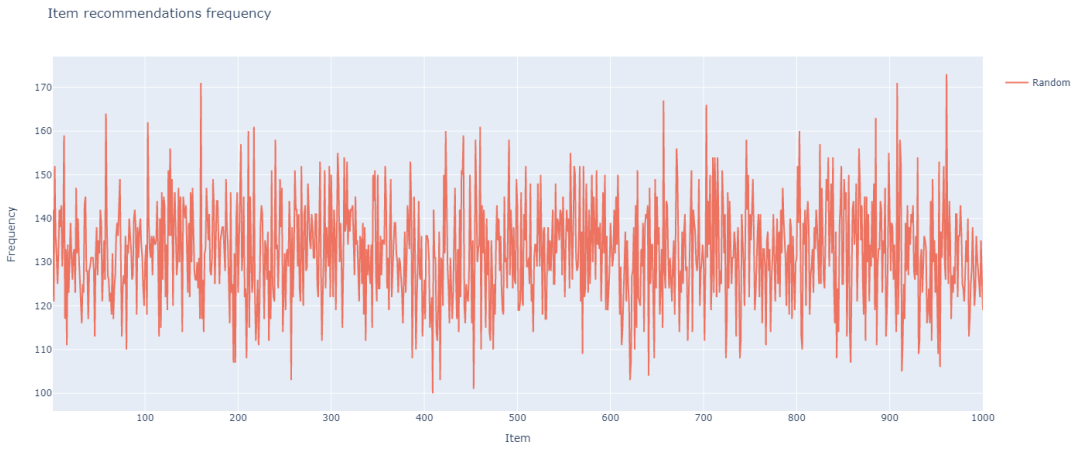

Figure 3.2: Frequency of recommendations per item, for a random predictor.

An important consideration about this plot is that it shows how much the

items are recommended, without any information about the relevance. To understand whether some recommendations are useful, this plot should be paired with the precision, stratified by popularity. This additional piece of information could be used to compare the performances between the recommendations in the short head and in the long tail.

3.5 Novelty

The novelty of a recommender system is the ability to recommend items that the user is not aware of [14]. In some environments this can be an important characteristic, in order to keep the user interested. The evaluation of such quality can be done by comparing the interaction timestamps of the test set with the interaction timestamps of the training set. A more accurate investigation can be done if detailed behavioral data are available, for example, the set of visited webpages by each user.

3.6 Other aspects relating to the performance

3.6.1 Serendipity

The serendipity is the ability to make users discover new items that are appreciated. In some scenarios, this can be the end goal of the recommender system. Can be checked by evaluating both relevance and novelty. Therefore, one way to view serendipity is the relevance of non-obvious suggestions [14].

3.6.2 Diversity

The diversity of a recommended list is the dissimilarity between the items suggested. The notion of item similarity can depend on content or interaction data, that is, if the item characteristics are similar or if the items tend to be bought from the same users. Diversity is important in environments where similar items

are interchangeable, for example, in a movie recommender system, suggesting 5 horror movies can cause a user to leave the platform, with respect to suggesting 5 movies of different genre, even if such a user likes horror movies. Diversity can be enforced by reordering the predicted ranked list, with approaches based on item features [15] (from information retrieval), or based on item popularity [6].

3.6.3 Trust

Trust measures the faith of users in the system recommendations. If an user does not trust the system, it is likely that such a user ignores the suggestions, even if they are relevant. This is important for items such that the user cannot immediately understand the relevance of the proposed items. When the recommender system provides explanations, the user is more likely to trust the system, especially if the explanations are logical [14]. In settings where trust is important, interpretable methods should be preferred.

3.6.4 Robustness

A recommender system is robust when attacks, such as fake ratings, or abnormal patterns in the data, do not significantly affect the rankings. Significant profit-driven motivations under attacks do exist; for example, a third part seller on Amazon could generate positive fake news, in order to make their products suggested to as many users as possible. If the interactions are purchases, such attacks are unlikely, because they would require an expensive set of purchases in order to affect the system.

3.6.5 Scalability

The mass of users of a service, and the size of the catalog, can lead to a huge set of possible interactions. For this reason, each method should also be considered for its requirements in terms of computational resources. While an e-mail ad-

vertisement campaign does not require significant requirements in terms of time efficiency, real-time services need fast recommendations. Even without real-time applications, the size of the data can be such that training time, prediction time, and memory consumption should be taken into account.

4

Methods for implicit feedback

In this chapter, suitable methods that can perform the recommendations are listed, including baselines, content-based methods, pure collaborative filtering methods, and hybrid methods. Most of the effort is on pure collaborative filtering, i.e. methods that base their recommendations exclusively on historical interactions. An algorithm falls into the collaborative filtering category if the recommendations for one given user are obtained considering the past interactions of other users, too.

4.1 How recommendations are computed

Recommendations are made for each user in the test set. An affinity score is computed between each user and, one by one, all items in the catalog. Such scores, also called predicted ratings in the explicit feedback framework, are denoted by $\hat{r}_{ui}$, where u is the target user and i is a chosen item. The r notation is because scores are considered an estimate of the missing entries of R. Once the scores are obtained, the recommendations are calculated as the list of items in the catalog, ordered by decreasing affinity.

In the considered use case, however, it is not desirable to suggest an item to a user who has already purchased it. For this reason, these items are removed from the suggestions.

4.1.1 Cold starts

Most methods are not able to perform the recommendations for a cold start user. A basic way to overcome this problem, when recommendations are needed for these users, is to use unpersonalized methods, such as the popularity-based recommender (4.2.1). An alternative, based on previous behavioral data, is proposed in Section 6.

4.2 Baselines

4.2.1 Popularity-based

This baseline is an unpersonalized method, where suggestions are the most popular items, regardless of the user.

$$\hat{r}_{ui} = \sum_{v=1}^{n} R_{vi}$$

Given the strong popularity bias that is typically present, it is often very challenging to perform better than recommending the most popular items.

4.2.2 Random predictor

The recommendation lists provided by a random predictor are just the result of a random shuffle of the catalog.

4.3 Content-based recommendations

The content-based method does not learn from the interaction matrix R. This method works by suggesting, to a given user u, the items most similar to those consumed previously. The similarity between items is computed on the item features. Let $X \in \mathbb{R}^{m \times p}$ be the content data matrix, with p recorded features. Let X_i be the feature vector of the item i, a row of X. A possible cosine-based similarity between items is defined as:

$$B_{ij} = sim(i, j) = \frac{X_i^T X_j}{||X_i|| \cdot ||X_j||} \tag{4.1}$$

The score matrix can be obtained as:

$$\hat{R} = RB \tag{4.2}$$

4.3.1 Feature extraction

In the case study of this work (1.2), the features that describe each items are clothing category, gender of the product, and a bag of words vector representation of the product description. A well-crafted feature extraction is crucial for the good functioning of the model, in particular, the usage of the best practices in natural language processing. During the extraction of a bag of words from the product description, stop words removal and stemming are implemented. Moreover, only a carefully selected subset of keywords, relevant to the domain context, is kept. Words with the same meaning in the clothing industry are merged, in order to remove noise. At the end of the process, each item is described by $p = 91$ variables.

4.4 Memory-based methods

Memory based methods, also called neighborhood-based methods, are collaborative filtering techniques in which recommendations are obtained by means of an aggregation of a neighborhood, using memorized feedback, without an explicit model of behavior. In general, a neighborhood of a user (or item) is a set of similar users (or items), that are used to infer the user-item score. Such methods can be seen as a generalization of the widely known K Nearest Neighbors classifier/regressor [14].
Those methods are divided in two categories: user-based methods and item-based methods. In user-based methods, the implicit assumption is that a user behaves as its similar users.
In item-based methods, instead, the underlying hypothesis is that similar items are rated in a similar way by the same user. The general scheme is:

1. Find a set of similar users (or items)
2. Aggregate the ratings of the neighborhood

However, the context of implicit and sparse feedback makes item-based procedures not viable, since there are too few interactions per user, making the aggregation unreliable.

4.4.1 Similarity

A key component of memory based models is the concept of similarity. In a content-based approach, two items are considered similar if their features match. In a demographic filtering scenario, two users are similar if they share the same demographic characteristics. However, in collaborative filtering, the concept of similarity does not depend on the features of the user or the item, but on the behavior of the interactions. This means that two users are similar if they have consumed the same items, regardless of their demographic features, and two items

are similar if they tend to be consumed by the same users, without considering the features of the items.
A commonly used similarity function between users is the cosine:

$$cos(u, v) = \frac{R_u^T R_v}{||R_u|| \cdot ||R_v||} \tag{4.3}$$

Where R_u is the row of R that corresponds to the user u.
Given that $r_{ui} \in \{0, 1\}\ \forall u \in \mathcal{U},\ \forall i \in \mathcal{I}$, the same similarity function can be written using the set notation:

$$cos(u, v) = \frac{|\mathcal{I}_u \cap \mathcal{I}_v|}{\sqrt{|\mathcal{I}_u|} \cdot \sqrt{|\mathcal{I}_v|}} \tag{4.4}$$

This function measures the number of commonly consumed items, normalized by the amount of items that each user purchased.
An extension that can weight in a different way the number of purchases of the two users is the asymmetric cosine [16]. In the original work, the formula is derived by writing the cosine distance as the product between two conditioned probabilities, then making it asymmetric by adding a parameter. In the resulting distance function, the exponents of the denominator are parametrized such that, by varying α, the right normalization balance, between the target user and the others, can be fine-tuned.

$$asymC(u, v) = \frac{|\mathcal{I}_u \cap \mathcal{I}_v|}{|\mathcal{I}_u|^{\alpha} \cdot |\mathcal{I}_v|^{1-\alpha}} \tag{4.5}$$

Where u is the target user, i.e. the user for which the recommendations shall be done. With this formulation, it is no longer true that $sim(u, v) = sim(v, u)$, and for this reason the specification of which user is the target is needed.

4.4.2 User-based KNN

This simple memory-based method works by averaging the ratings of the K most similar users to the target user. Let $sim(\cdot, \cdot)$ be the chosen similarity function. The neighborhood of u, that is the set of the K most similar users, can be defined as:

$$N_K(u) = \{(v_1, v_2, ..., v_K) \in \mathcal{U} \backslash \{u\} : sim(u, v') \leq sim(u, v); \\ \forall v \in (v_1, v_2, ..., v_K), v' \in \mathcal{U} \backslash \{u\}\} \tag{4.6}$$

Then the aggregation is just the average:

$$\hat{r}_{ui} = \frac{1}{K} \sum_{v \in N_K(u)} r_{vi} \tag{4.7}$$

4.4.3 User-based weighted KNN

The aggregation can be modified to give more weight to more similar users in the neighborhood. Let $w_{uv} = sim(u, v)$:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_K(u)} w_{uv} r_{vi}}{\sum_{v \in N_K(u)} w_{uv}} \tag{4.8}$$

With this adjustment, it could also be considered the removal of the neighborhood, that is equivalent to set $K = m$, since the similarity in yet taken into account in the aggregation. In this way, all the information is used.

$$\hat{r}_{ui} = \frac{\sum_{v \in \mathcal{U}} w_{uv} r_{vi}}{\sum_{v \in \mathcal{U}} w_{uv}} \tag{4.9}$$

4.5 Using standard machine learning

Off-the-shelf machine learning models cannot be used directly in recommending systems [14]. This is because the response variable y is the matrix R and needs to be predicted with itself as input. However, it is possible to adapt standard machine learning by predicting one column of R at the time, using all the other columns as features in input. In this way, the chosen model can learn patterns of the type: "user u consumed items i_x, i_y" $\longrightarrow$ "user u may consume item i_z", because the prediction of r_{uj} is based on the other purchases of u. This method is collaborative filtering because the associations are learned using all the users interactions, and the recommendations for one user are based on the other users behavior.

4.5.1 Column-wise logistic regression

A suitable model is the logistic regression. Given the sparsity in the data, simple models are preferable. Let $\hat{R}$ be the matrix with the predicted ratings. Let $R^{[i]}$ be the column of R corresponding to the item i, and let $R^{[-i]}$ be the matrix obtained with all the columns of R except for $R^{[i]}$. Let $LR(y|X)$ be the maximum likelihood estimation $(\hat{\beta}_0, \hat{\beta})$ of the logistic regression parameters, with y as target variable and X as data matrix. Finally, let $\sigma(\cdot)$ be the element-wise sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. A procedure to efficiently estimate R is described in Algorithm 4.1.

Algorithm 4.1 Logistic regression for collaborative filtering

for $i \in 1, ..., m$:
  $y \leftarrow R^{[i]}$
  $X \leftarrow R^{[-i]}$
  $\hat{\beta}_0^{(i)}, \hat{\beta}^{(i)} \leftarrow LR(y|X)$
  $\hat{R}^{[i]} \leftarrow \sigma(\hat{\beta}_0^{(i)} + X\hat{\beta}^{(i)})$
end for

At the end of this procedure, all the scores $\{r_{ui}\}$ are estimated in $\hat{R}$. Notice that the predictions are made on the train instances, so using a complex method instead of logistic regression can easily cause overfitting issues.

Interpretability

This method is suitable for interpretation: the parameters $\hat{\beta}^{(i)}$, contain the information of which consumed items caused the suggestions. In fact, in the prediction of r_{ui}, each consumed item by u results in a contribution whose intensity is indicated by the corresponding entry of $\hat{\beta}^{(i)}$, denoted by $\hat{\beta}^{(i)}_j$.

$$\hat{r}_{ui} = \sigma(\hat{\beta}^{(i)}_0 + X\hat{\beta}^{(i)})$$

$$\hat{r}_{ui} = \sigma(\hat{\beta}^{(i)}_0 + \sum_{j \in \mathcal{I}_u \setminus \{i\}} \hat{\beta}^{(i)}_j)$$

Since the final ordering is invariant with respect to monotone functions, the sigmoid function can be removed from the prediction function:

$$\hat{r}'_{ui} = \hat{\beta}^{(i)}_0 + \sum_{j \in \mathcal{I}_u \setminus \{i\}} \hat{\beta}^{(i)}_j$$

4.5.2 Hybrid extension

This method can use demographic data. User feature columns are concatenated to $R^{[-i]}$, so that the model incorporates the demographic data in the predicting function. Let $Q \in \mathbb{R}^{n \times p}$ be the demographic data matrix, where each row is the p-dimensional feature vector of an user. At each iteration i, the data matrix used as input for the logistic regression is:

$$X = (R^{[-i]}|Q) \tag{4.10}$$

In this way, the parameter vector contains one collaborative part β and one demographic part γ, making it an hybrid method:

$$MLE(\beta_0^{(i)}, \beta^{(i)}, \gamma^{(i)}) = (\hat{\beta}_0^{(i)}, \hat{\beta}^{(i)}, \hat{\gamma}^{(i)}) = LR(y|X) \tag{4.11}$$

$$\hat{r}_{ui} = \sigma(\hat{\beta}_0^{(i)} + \sum_{j \in \mathcal{I}_u \setminus \{i\}} \hat{\beta}_j^{(i)} + Q_u^T \hat{\gamma}^{(i)})$$

The procedure is detailed in Algorithm 4.2.

Algorithm 4.2 Logistic regression for hybrid filtering

for $i \in 1, ..., m$:
  $y \leftarrow R^{[i]}$
  $X \leftarrow (R^{[-i]}|Q)$
  $\hat{\beta}_0^{(i)}, \hat{\beta}^{(i)}, \hat{\gamma}^{(i)} \leftarrow LR(y|X)$
  $\hat{R}^{[i]} \leftarrow \sigma(\hat{\beta}_0^{(i)} + X\hat{\beta}^{(i)} + Q\hat{\gamma}^{(i)})$
end for

4.6 Latent factors models

The category of models based on latent factors, also called factorization machine models, is a widely studied topic in collaborative filtering [17]. The general model is well known, and several ways to estimate it have been proposed. The idea is to learn a latent representation for users and items, where each hidden factor corresponds to one characteristic of both the items and the users profiles. The recommendations are computed by suggesting items that match the target user's profile. For example, in a book recommender system, an user could be interested in math, and if the estimation process built good profiles, these models are able to represent such interest in the latent space, and thus recommend math books to the user.

The scores are predicting by the inner product, which sums all the matches be-

tween the user and the item profiles. Let U_u be the hidden representation of the user u, and V_i the hidden representation of the item i.

$$\hat{r}_{ui} = < U_u, V_i > = U_u^T \cdot V_i \tag{4.12}$$

By stacking all the predicted scores into the matrix $\hat{R}$, the matrix factorization problem is derived:

$$\hat{R} = U \cdot V^T \tag{4.13}$$

Where $U \in \mathbb{R}^{n \times k}$ is the matrix whose rows are the k-dimensional hidden representations of the users, and $V \in \mathbb{R}^{m \times k}$, is the matrix whose rows are the k-dimensional hidden representations of the items.

Example

Assume the interaction matrix encodes a clothing company historical purchases. There are two users and three items, and a two-dimensional reasonable hidden representation is available. The first factor represents the gender, the second encodes the color attitude, high values for brightness attitude and low values for darkness attitude. The first user is a man that loves bright clothing. The second user a woman that does not have color preferences. All the items are for man, one is bright, one gray, one dark. The matrix reconstruction could work like this:

$$\hat{R} = \begin{pmatrix} 1 & 1 \\ -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix}$$

$$\hat{R} = \begin{pmatrix} 2 & 1 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

For the first user, it is easy to recommend the bright dress, while for the second there are not good matching items.
However, good representations are not given and should be learned, starting from R.

4.6.1 SVD

The estimation of latent factor models with the singular value decomposition was firstly proposed by [18]. The singular value decomposition factorizes a matrix $M \in \mathbb{R}^{n \times m}$ as:

$$M_{n \times m} = U_{n \times n} \cdot \Sigma_{n \times m} \cdot Q^T_{m \times m} \tag{4.14}$$

Where U and Q are unitary*, Σ is rectangular diagonal with non-negative values on the diagonal, called singular values.
A low rank approximation of M can be obtained by considering only the higher $k < min(n, m)$ singular values in Σ, and setting the others at 0. This is also called truncated SVD:

$$\hat{M}_{n \times m} = U_{n \times k} \cdot \Sigma_{k \times k} \cdot Q^T_{k \times m} \tag{4.15}$$

In the recommender system notation, this is:

$$\hat{R}_{n \times m} = U_{n \times k} \cdot \Sigma_{k \times k} \cdot Q^T_{k \times m} \tag{4.16}$$

Setting $V = \Sigma \cdot Q^T$ results in:

$$\hat{R} = U \cdot V^T \tag{4.17}$$

With $U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{m \times k}$.

*$X \in \mathbb{R}^{n \times n}$ is unitary if $XX^{-1} = X^{-1}X = I_n$. The real analogue is the orthogonal matrix

The estimation of the latent factor model is obtained. Such procedure implicitly solves the following optimization problem:

$$U, V = \underset{\substack{U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{m \times k}: \\ U^T U = V^T V = I_k}}{\operatorname{argmin}} ||R - UV^T||^2 \tag{4.18}$$

Where $|| \cdot ||$ indicates the Frobenius norm. Such method aims to find the best matrices, with orthogonal columns, that minimize the reconstruction loss. The actual algorithm used for the estimation is described in [19].

4.6.2 Alternating Least Squares

Another widely used estimation technique is the alternating least squares, proposed with implicit feedback in [20]. The difference with respect to the SVD estimation is that there is a relaxation of the orthogonality constraint present in 4.18, in the optimization problem:

$$U, V = \underset{U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{m \times k}}{\operatorname{argmin}} ||R - UV^T||^2 \tag{4.19}$$

Given the freedom of U and V, in practice a regularization term is needed:

$$U, V = \underset{U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{m \times k}}{\operatorname{argmin}} \left(||R - UV^T||^2 + \lambda ||U||^2 + \lambda ||V||^2 \right) \tag{4.20}$$

The optimization problem formalized in [20] also includes a confidence score c_{ui} for each observed interaction, but in the case study of this work 1.2, this information is not present, and setting $c_{ui} = 1 \; \forall u, i$ is reasonable, removing the confidence from the model.
The algorithm that solves the problem is the alternating least squares. The general

idea is that, if U is considered fixed:

$$\hat{R} = UV^T$$

Becomes a multiple least squares problem

$$\hat{Y} = X\mathrm{B}$$

With $Y = R, X = U, \mathrm{B} = V^T$. The same reasoning can be made with V fixed:

$$\hat{R} = UV^T$$

$$\hat{R}^T = VU^T$$

The algorithm uses the least squares formula: $\hat{\mathrm{B}} = (X^T X)^{-1} X^T Y$, or the l_2 regularized version [21]: $\hat{\mathrm{B}} = (X^T X + \lambda I)^{-1} X^T Y$, in order to update one matrix at a time, keeping the other fixed.

Algorithm 4.3 Alternating Least Squares

for $i \in \{1, ..., \mathrm{N\_ITER}\}$:
  $V \leftarrow \left((U^T U + \lambda I_k)^{-1} U^T R\right)^T$
  $U \leftarrow \left((V^T V + \lambda I_k)^{-1} V^T R^T\right)^T$
end for

The algorithm requires to invert $k \times k$ matrices, but given that the usual values for the number of latent factors are between $k = 50$ and $k = 500$, the procedure can run efficiently.

4.6.3 Non Negative Matrix Factorization

The possibility for U and V to have negative entries makes the interpretation of factors more difficult. For this reason, non-negative constraints are introduced:

$$U, V = \underset{\substack{U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{m \times k}: \\ U, V \geq 0}}{\operatorname{argmin}} ||R - UV^T||^2 \tag{4.21}$$

The actual algorithm, with all the regularizations and details, is described in [22]. In practice, this method is often slower than SVD and ALS.

4.7 EASE$^{\mathrm{R}}$

Embarrassingly Shallow AutoAncoders (in Reverse order, EASE$^{\mathrm{R}}$) is a collaborative filtering model that suggests items on the basis of similarity between items [23]. Each past interaction j of the user u contributes to the scores r_{ui} by an addend, which is the similarity between the item i and j, B_{ij}. This method is collaborative because the similarities are not based on the features of the items, but are learned from the interaction matrix. EASE$^{\mathrm{R}}$ works by solving the following problem:

$$\hat{B} = \underset{B \in \mathbb{R}^{m \times m};\ diag(B) = 0}{\operatorname{argmin}} \left(||R - RB||^2 + \lambda ||B||^2 \right) \tag{4.22}$$

The reconstructed matrix with the scores r_{ui} is:

$$\hat{R} = R\hat{B} \tag{4.23}$$

In this way, the scores are obtained by:

$$\hat{r}_{ui} = \sum_{j=1}^{m} r_{uj} \hat{B}_{ij} = \sum_{j \in I_u} \hat{B}_{ij} \tag{4.24}$$

An explicit solution to the optimization problem is available, setting:

$$\hat{P} = (R^T R + \lambda I_m)^{-1} \tag{4.25}$$

$$\hat{B} = I_m - \hat{P} \cdot \text{diagMat}(1 \oslash \text{diag}(\hat{P})) \tag{4.26}$$

Where $\oslash$ is the element-wise division. That is:

$$\hat{B}_{ij} = \begin{cases} 0 & \text{if } i = j \\ -\frac{\hat{P}_{ij}}{\hat{P}_{jj}} & \text{if } i \neq j \end{cases} \tag{4.27}$$

4.7.1 Interpretation

If R is standardized, and is assumes that each row of the standardized R comes from a vector of gaussian random variables $X \sim N(0, \Sigma)$, the elements outside of the diagonal of $\hat{B}$ are the partial correlations. This means that the similarity estimated by EASE$^{\text{R}}$ is related to the partial correlations, i.e., the correlations between items without the effect of all the other items. When $B_{ij} = 0$ with $i \neq j$, it means that the item i and the item j are conditionally independent. Practically, given a user u, this method suggests items that are frequently consumed by users who have interacted with a subset of I_u.

5 Experimental results

5.1 Experimental setup

5.1.1 Data

As described in Section 1.2, the data comes from a real company that sells luxury clothing and accessories. The main source of available data is the transactions history, an implicit feedback. Content data is present, in particular the category of the product and the description of the items. Some demographic features are present, but they can be available only after the first purchase of the user, due to the registration process. Finally, behavioral data is available, in particular the set of interactions of the form "user u visited the webpage of the item i".

5.1.2 Selection and splitting

The full dataset is composed of the past purchases, selected along with users who bought at least 2 items and the top $m = 1000$ sold items. 15% of the interactions, randomly selected, are set as the test set. Recommendations are made to users for whom there is at least one purchase in the test set.

5.1.3 Descriptive statistics

There are $n = |\mathcal{U}| = 94064$ users, of those $|\mathcal{U}^{(train)}| = 90072$ have at least one interaction in the training set, and $|\mathcal{U}^{(test)}| = 29710$ have at least one interaction in the test set.
The sparsity of R, i.e. the proportion of zero entries, is 99.7%.
Users have a very small number of interactions, in mean, only 2.66 bought items per user. The amount of interactions per item is larger, there are 250.7 mean sales per item.
However, the amount of sales per item is not equally distributed, the popularity bias is strongly present, having that the most sold items account for the majority of the total sales as depicted in Figure 5.1:

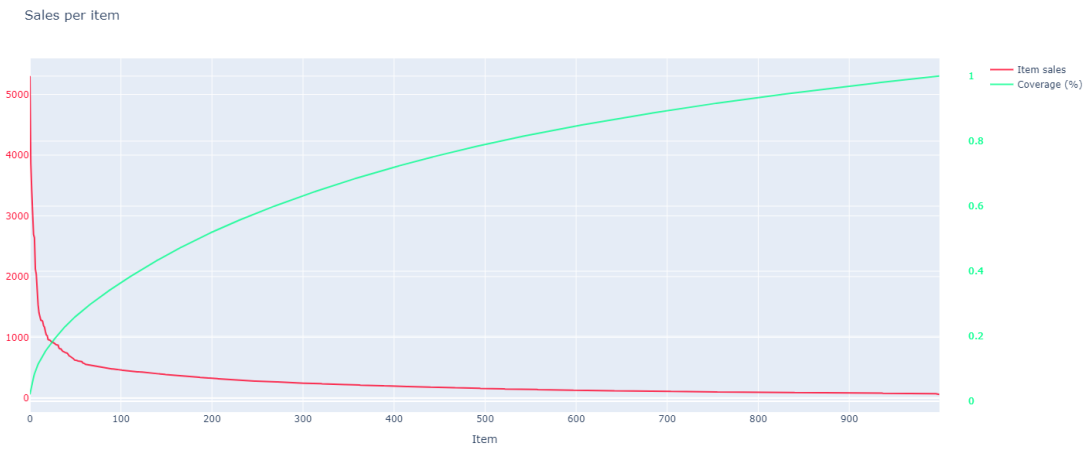

Figure 5.1: Popularity bias and sales coverage.

There are users for whom many interactions are known, and hence the recommendations for them should be more accurate. The distribution of the number of train purchases for test user is illustrated in Figure 5.2.

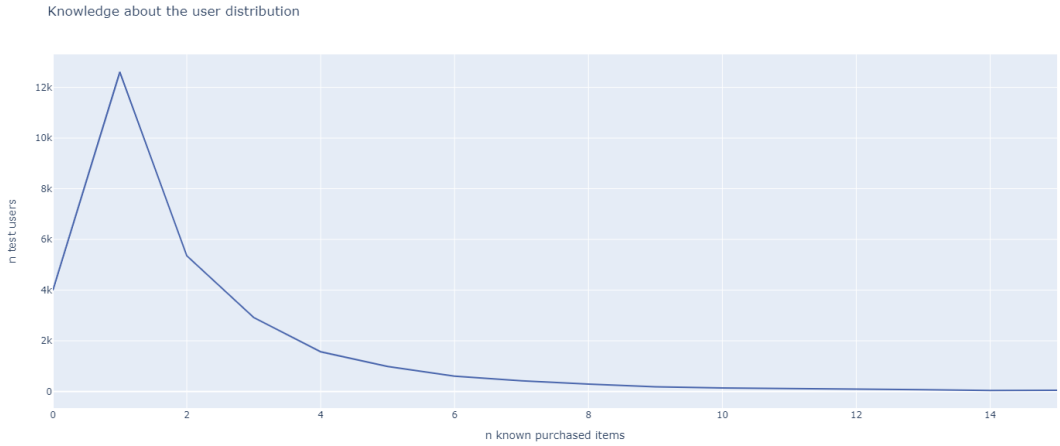

Figure 5.2: Number of test users stratified by train purchases.

For most users, there are at most two purchased items. This kind of information sparsity could make simpler models perform the best. There are $|\mathcal{U}| - |\mathcal{U}^{(train)}| = 3992$ cold start users. Notice that the true proportion of cold starts, in the production environment, is much greater: Every user, before its first purchase, is a cold start; Furthermore, the way the dataset is retrieved, by picking interactions only if belong to users who bought at least two items, greatly reduces the amount of cold starts. For this reason, a special treatment for such users is reasonable. Approaches to address cold start are discussed in Chapter 6.

5.2 Latent factor models and popularity bias

The popularity bias present in the data makes factorization models perform poorly. This type of behavior seems not to be well studied in the literature. For this reason, this behavior is deeply investigated, demonstrating how the standardization of R can help this family of models.
Top items are not recommended by the latent factors models. The expected behavior of a method is to recommend each item a number of times that is proportional to its popularity. All the proposed methods show the expected recommendation frequencies, except for the latent factors models.

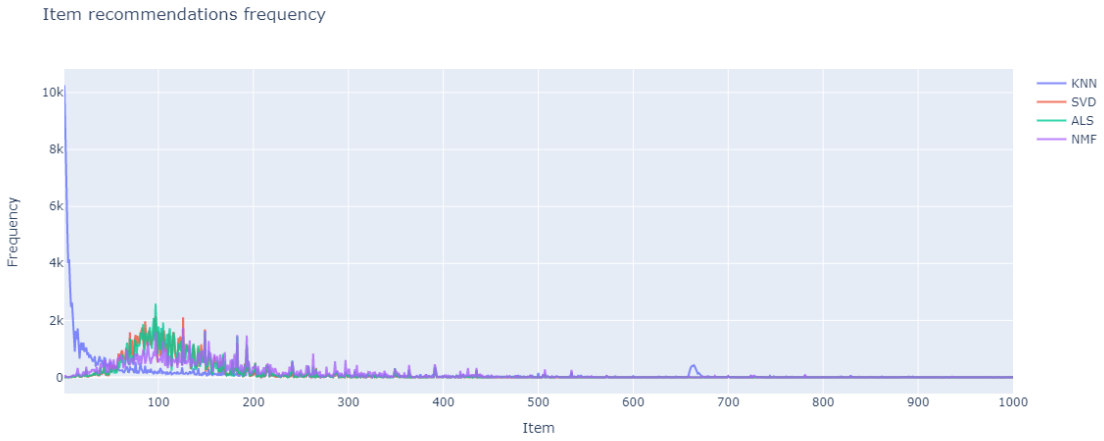

Figure 5.3: Recommendation frequency of latent factors models, against KNN.

In Figure 5.3, the latent factors model, trained with SVD, ALS or NMF, with $k = 100$ latent factors, is compared to the simple KNN, with neighbors of 500 users. It can be seen how factorization models do not recommend the top products, as KNN does. The most recommended items are around the 100-th most popular product, with the same number of latent factors. In fact, by changing the number of latent factors, the most recommended items shift accordingly, becoming the ones around the k-th most popular product. Such a "moving hump" can be seen

by showing the recommendation frequencies of the latent factors models with $k \in \{10, 100, 500\}$ (Figure 5.4).

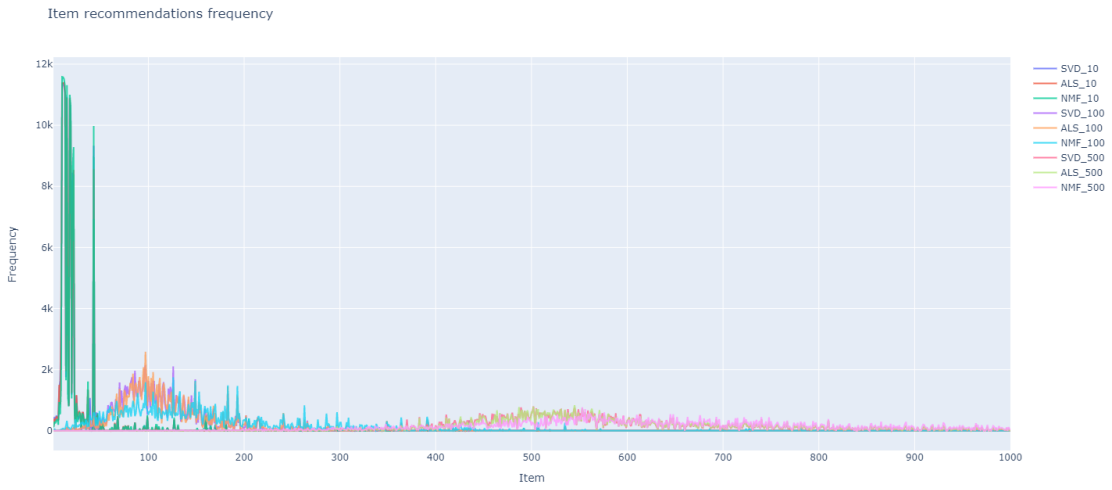

Figure 5.4: Recommendation frequency of latent factors models, varying the number of factors.

The performances are a lot worse than the KNN and the popularity-based baseline (Table 5.1).

Model	mAP@10
Popularity based	0.0466
Knn, Neighbor size: 500	0.0624
SVD, 100 latent factors	0.0269
ALS, 100 latent factors	0.0273
NMF, 100 latent factors	0.0283

Table 5.1: Latent factors models performances.

5.2.1 Generated data

In order to show that the popularity bias is responsible for the factorization models problem, the behavior is tested on two different randomly generated datasets. The first one, is generated in such a way that does not contain the popularity bias,

while the second one emphasizes the bias.
The generation of the first dataset is easy: Random coordinates $(u, i) \in \{1, ..., n\} \times \{1, ..., m\}$ are sampled until the desired total number of sales is reached, obtaining $T^{(unif)}$, and so $R^{(unif)}$. In Figure 5.5, it can be seen how the popularity bias

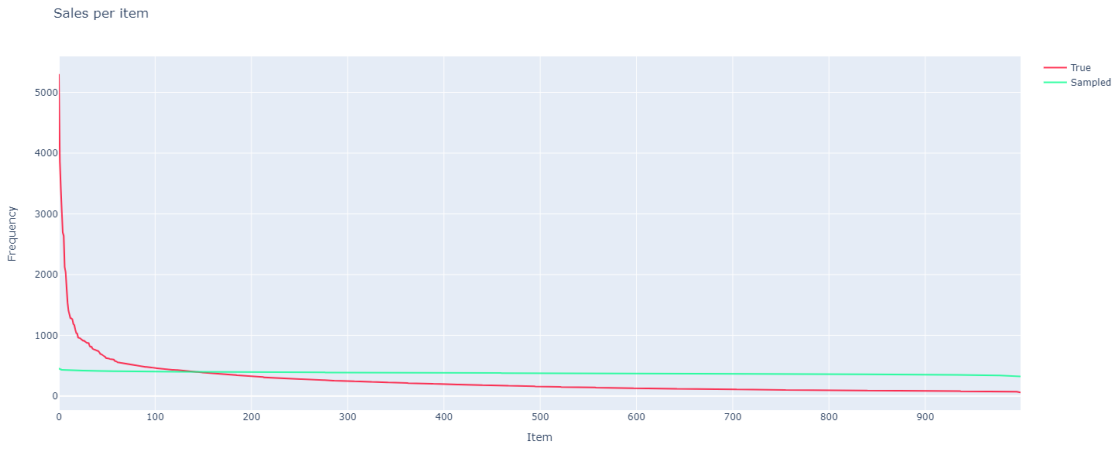

Figure 5.5: Popularity bias in the uniformly generated dataset.

is not present in the generated dataset.
For the generation of the second dataset, a matrix with the same shape as the real one is considered. For each item that corresponds to one column of $R^{(zeta)}$, the generation is carried out by:

1. Sample the popularity bias, that is the number of sales of that item, that is also the sum of the considered column of $R^{(zeta)}$;

2. Randomly distribute such sales along the fake users.

In the first step, the popularity of each item is sampled from the zeta distribution [24], an extension of the Zipf's law [25]. The probability distribution is:

$$f_{zeta}(x) = \frac{x^{-s}}{\zeta(s)} \quad \forall x \in \{1, 2, ...\} \tag{5.1}$$

Where $\zeta(s) = \sum_{i=1}^{+\infty} \frac{1}{n^s}$ is the Riemann zeta function, with $s > 1$.

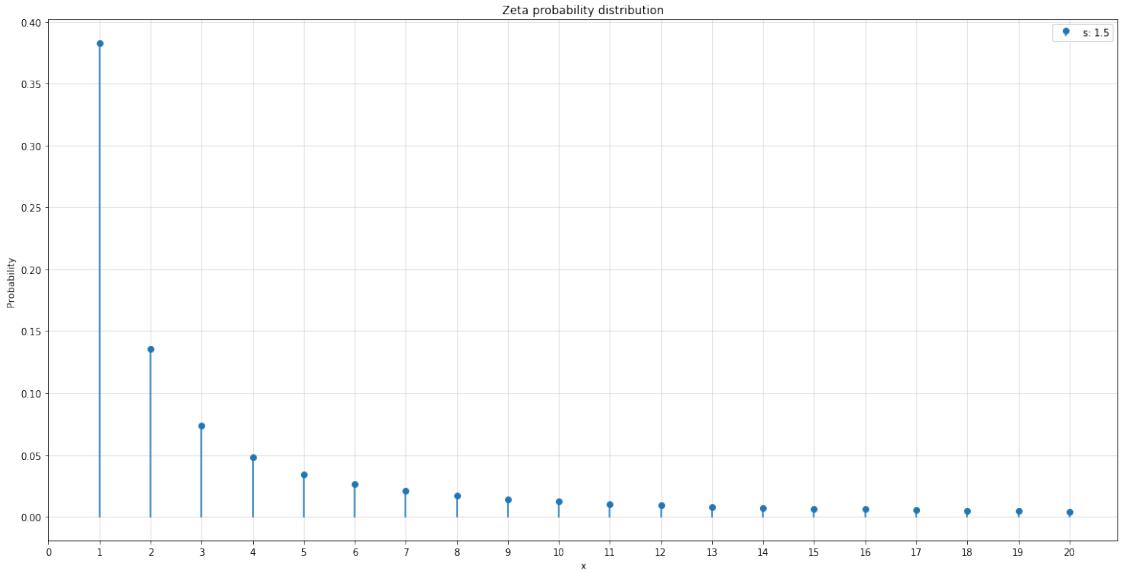

Figure 5.6: Zeta probability distribution, with $s = \frac{3}{2}$.

This distribution has a very long tail, so it is likely to sample some items with huge popularity.

In the second step, each entry of the considered column can be 1 or 0, and the sum of the column is guaranteed to be equal to the popularity. This is achieved by applying the steps described in Algorithm 5.1.
The parameter s is tuned and the sampling is repeated, until a $R^{(zeta)}$ with a sparsity similar to the original is obtained. In Figure 5.7 it can be seen how the popularity bias is emphasized in the zeta generated dataset.

5.2.2 Behavior in the generated datasets

The hump does not appear in the dataset without the popularity bias (Figure 5.8), contrary to the zeta generated dataset, where the hump is stronger than the one in the real dataset (Figure 5.9).

Algorithm 5.1 Uniform assignment of fake sales

Input: p = Number of sales of the considered item i
Output: $R^{[i](zeta)}$

$R^{[i](zeta)} = 0_n$
$sales = 0$
while $sales < p$:
  Uniformly sample an index $u \in \{1, ..., n\}$
  if $R_u^{[i](zeta)} = 0$:
    $R_u^{[i](zeta)} = 1$
    $sales = sales + 1$
  end if
end while

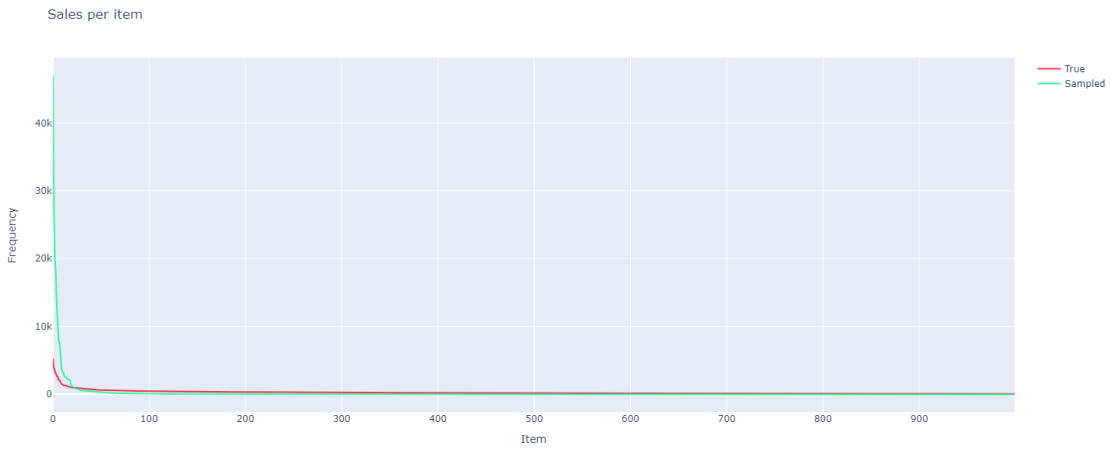

Figure 5.7: Popularity bias in the zeta-generated dataset.

Without any other type of relationship in the data, it was shown how the presence of the popularity bias affects the latent factors models.

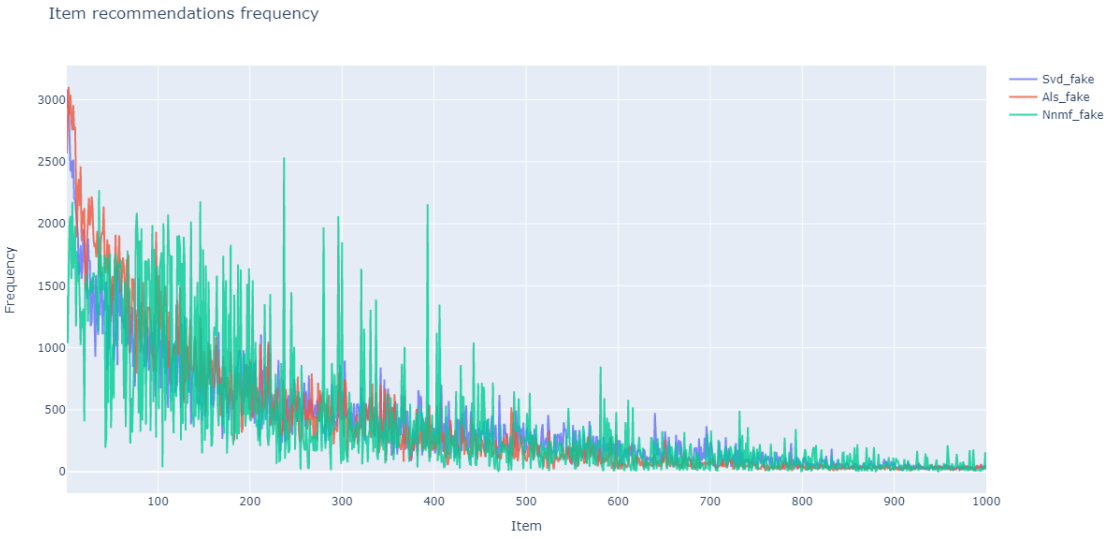

Figure 5.8: Recommendation frequency of latent factors models in the uniformly generated dataset.

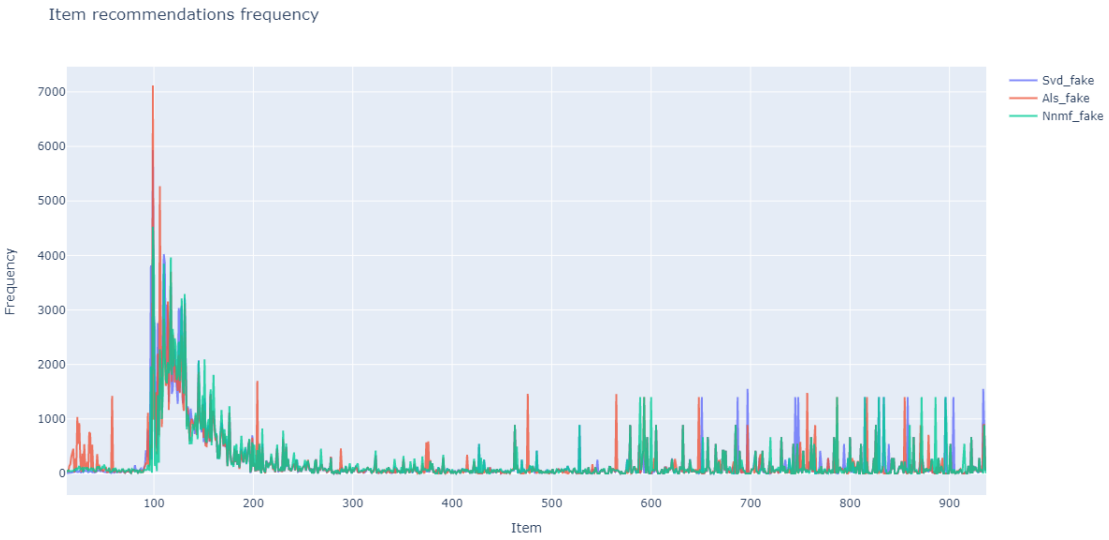

Figure 5.9: Recommendation frequency of latent factors models in the zeta-generated dataset.

5.2.3 SVD DECOMPOSITION INSPECTION

In this section, is considered the SVD training for the matrix factorization mode, in order to understand why the hump is appearing. The reason why SVD was chosen is that it has a set of known good properties. We can decompose and plot the effect of each individual factor:

$$\hat{R} = \sum_{i=1}^{k} M_i \tag{5.2}$$

$$\hat{R} = \sum_{i=1}^{k} \sigma_i U_i \otimes V_i \tag{5.3}$$

Where $\otimes$ is the outer product. The first factors account for the majority of the reconstruction, and by looking at the contributions of the first 4 factors, it can be seen how such factors contribute only to the top popular items (Figure 5.10).

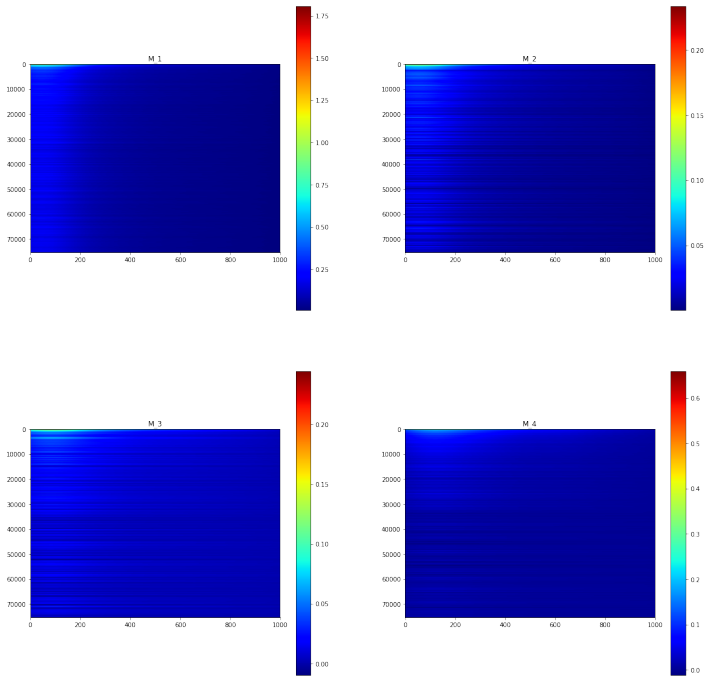

Figure 5.10: Contribution of the first 4 factors in the SVD reconstruction of R.

The most popular columns are perfectly reconstructed (Figure 5.11), because they are more relevant in the optimization problem (4.18), having more information to represent. This causes the fact that $R^{(train)}$, which are the columns of the top popular items, the zero entries are perfectly reconstructed in $\hat{R}^{(train)}$. The scores of the top items not yet purchased, so with $R_{ij} = 0$, are $\hat{r}_{ui} \approx 0$, and thus the top items are never recommended, explaining the strange behavior. The hump appears because, differently from the uniformly generated fake dataset, the real data have some patterns in it. For this reason, there are some useful latent factors that can reconstruct the matrix. The result is that only the first columns are perfectly reconstructed, while the columns before the k-th are a mixture between overfitting representations and useful factors.

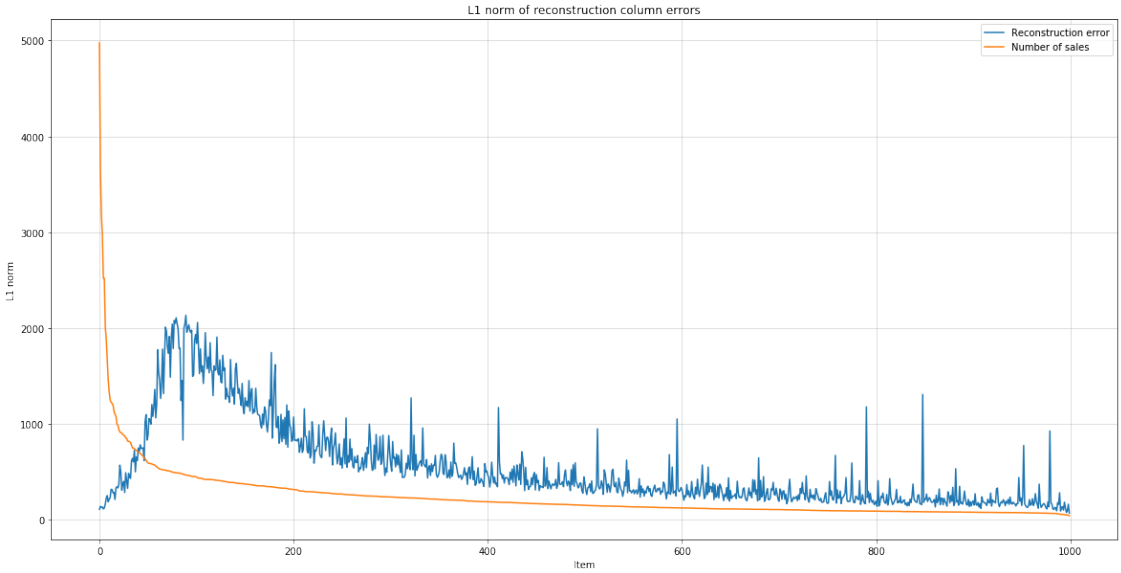

Figure 5.11: SVD reconstruction error of R.

5.2.4 Simplified example of the problem

The hump follows the number of latent factors because the decomposition can perfectly reconstruct one column per latent factor. Assuming the directions of maximum variability in the item space are the most popular items themselves, that

is, the direction of maximum variance is e_1, the second one is e_2, etc., that is, the extremization of the popularity bias, the SVD factorization would resolve as in the following example: V is divided between two parts, the one where the sort of item overfitting takes place, $V^{(a)}$, and the rest of the matrix, $V^{(b)}$. Here $V^{(a)}$ is set to I_k, but is not mandatory for the behavior appearance, it is just to keep the example as simple as possible. For simplicity, assume that the item indexes are ordered in such a way that the most sold items are fist, so item 1 is the most sold one, and item m the least sold one.

$$V_{m\times k} = \begin{pmatrix} V^{(a)}_{k\times k} \\ \\ V^{(b)}_{m-k\ \times\ k} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ \\ & & V^{(b)} & & \\ \\ \end{pmatrix}$$

Assume user u bought items $\{1, 3\}$. Its R column can be perfectly reconstructed by setting $U_i = (1\,0\,1\,0\,0\ \dots\ 0)$. Now, say that user i bought items $\{1, 3, k+2\}$. Its embdedding can easily reconstruct its purchases of 1 and 3, but not $k+2$. This is because with this scheme the decomposition can overfit up to k columns. So the first k entries of its R column can be accurately reconstructed, while the others will not, by setting the first k entries of its embedding as: $U_{i,1:k} = (1\,0\,1\,0\,0\ \dots\ 0)$. The other $m-k$ entries should be the best linear combination of the latent factors which minimize the reconstruction loss of the columns $\{k+1, k+2, \dots m\}$. This is why the scores for popular items are so low that they are not recommended.

5.2.5 Standardization

The popularity bias can be removed from the R matrix by standardizing it. Each item has the same mean and variance, making them weight the same in the optimization problem. This simple change makes the hump disappear. Moreover, the recommendation frequencies are flat, instead of encouraging the top products (Figure 5.12).

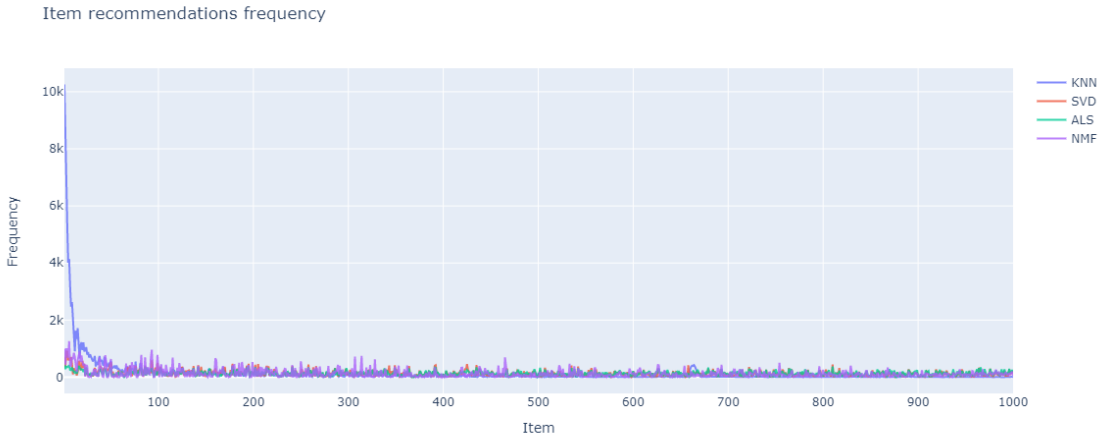

Figure 5.12: Recommendation frequency of standardized latent factors models, against KNN.

5.3 Results

5.3.1 Compared methods

In this section, the set of chosen methods is listed:

- Baselines: A random baseline and the popularity-based predictor are implemented, mainly for the comparison purpose.

- Content-based: A method based on the similarity between items. In particular, a carefully extracted bag-of-words representation of the item description, together with other product features, is used in order to define the distances.

- Collaborative filtering:

  - Two user-based neighborhood methods have been tested, one simple KNN and one weighted KNN with asymmetric cosine distance.

  - Column-wise logistic regression is implemented, together with its hybrid extension, that includes demographic data.

  - The latent factors model is tested, trained with three different approaches: SVD, ALS, and NMF.

  - EASE^{R} is also compared.

Name	Model	Category	Section
MostPop	Popularity-based recommender	Baseline	4.2.1
Random	Random predictor	Baseline	4.2.2
MostSimilar	Content-based recommender	Content-based	4.3
Knn	User based KNN	Collaborative filtering	4.4.2
AsymWKnn	User based weighted KNN with asymmetric cosine distance	Collaborative filtering	4.4.3
LogReg	Colums-wise logistic regression	Collaborative filtering	4.5.1
HybridLogReg	Colums-wise logistic regression with demographic data	Hybrid	4.5.2
SVD	Latent factors model with SVD training	Collaborative filtering	4.6.1
ALS	Latent factors model with ALS training	Collaborative filtering	4.6.2
NMF	Latent factors model with NMF training	Collaborative filtering	4.6.3
$EASE^R$	Embarrassingly shallow autoencoders	Collaborative filtering	4.7

Table 5.2: Compared methods.

5.3.2 Models configuration

In this section, the configuration of each compared method, for example, the hyperparameters settings, is listed. Each specific setting is the result of a grid search like experiment on a small validation set, where each best performing configuration is selected.

Name	Settings
MostPop	·
Random	·
MostSimilar	·
Knn	$k = 500$
AsymWKnn	$\alpha = 0.4$
LogReg	No regularization ($\lambda = 0$)
HybridLogReg	No regularization ($\lambda = 0$)
SVD	$k = 100$, R standardized
ALS	$k = 100$, R standardized, $\lambda = 0.01$
NMF	$k = 100$, R standardized
EASE^{R}	$\lambda = 0.01$

Table 5.3: Configuration of the methods.

5.3.3 Metrics

In order to compare the methods, the following metrics are used:

- *mAP*: These metric measures the relevance of the first recommendations, rewarding the relevance of the very first recommended items (Section 3.4);
- *Recall*: The recall measures the ability of the method to include the items already consumed within the first recommendations (Section 3.10);
- *Hit ratio*: This is the proportion of users for which at least one relevant item is proposed in the first recommendations. The hit ratio is highly interpretable from a business point of view, since it can be seen as the proportion of users who are buying (Section 3.3.4);
- *Coverage*: The proportion of items that are proposed at least once. Ideally, a method can recommend all the catalog, i.e. $CC@k = 1$ for a small k (Section 3.13).

The considered length of the recommendation list, for the metrics, is 10, that is a reasonable amount of items to suggest, in an e-commerce environment.

The corrected versions of *mAP* and *recall* (Section 3.3.1,3.3.2) are not used, because for a large proportion of test users, there is only one interaction within the test set. This would mean, for such users, that only the first recommendation is used in the metrics computation. This is something to avoid if the idea is to propose more than one item, at production time, as in the considered use case.

5.3.4 Test set results

The methods are compared on the full test set.

	mAP@10	Rec@10	HR@10	CC@10
MostPop	0.0466	0.1186	13.35%	1.0%
Random	0.0032	0.0100	1.21%	100.0%
MostSimilar	0.0348	0.0786	9.03%	99.4%
Knn	0.0624	0.1324	15.03%	100.0%
AsymWKnn	0.0732	0.1613	18.17%	90.8%
LogReg	0.0611	0.1345	15.2%	100.0%
HybridLogReg	0.0611	0.1345	15.2%	100.0%
SVD	0.0572	0.1130	13.1%	100.0%
ALS	0.0569	0.1161	13.38%	100.0%
NMF	0.0517	0.1079	12.57%	99.8%
$EASE^R$	0.0712	0.1522	17.47%	97.8%

Table 5.4: Test set results.

In Table 5.5 it can be seen that:

- The best performance, in terms of relevance, is obtained by AsymWKnn. However, such method has a very low coverage, meaning that its relevance could be high because it takes more advantage of the popularity bias, with respect to the other methods, that have a larger coverage.

- $EASE^R$ is the second-best performing method for its relevance, but it has a better coverage.
- The latent factors model performs worst when fitted with NMF.
- The content-based method, MostSimilar, performs worse than any collaborative filtering method.
- Demographic data does not improve the results of the logistic regression, as logistic regression with and without such data performs almost the same (metrics are not exactly the same, as seems in the Table 5.5 due to rounding).

It is important to keep in mind that these metrics measure the ability of the methods to suggest items that the user would buy without the suggestion. This means that methods that mostly propose popular items are expected to have a higher relevance, even if the quality of their suggestions is not better. For this reason, the recommendation frequencies are compared. In fact, the methods that have the best metrics are the same that recommend the top products the most (Figure 5.13, Figure 5.14).

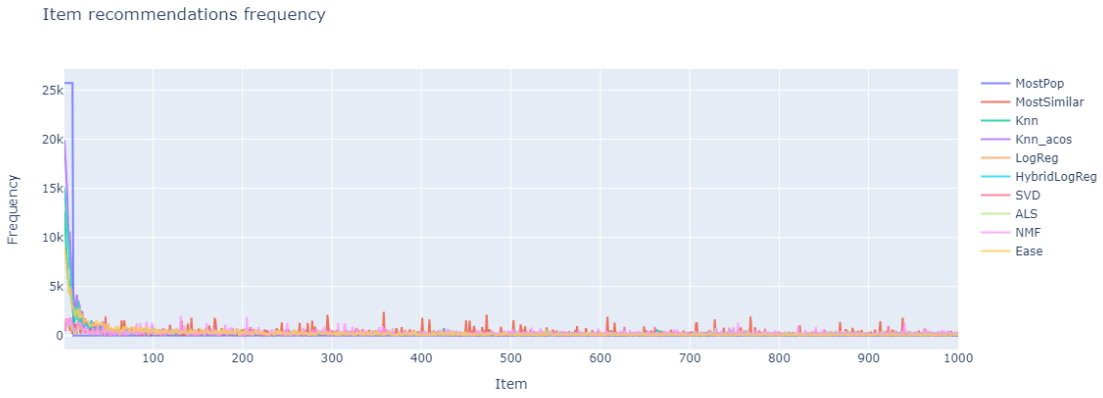

Figure 5.13: Recommendation frequencies comparison.

For this reason, the accuracy of the predictions is compared by stratifying the recommendations by popularity rages. In Figure 5.15, it can be seen that, in general, a given method has better precisions than others in the popularity bands it

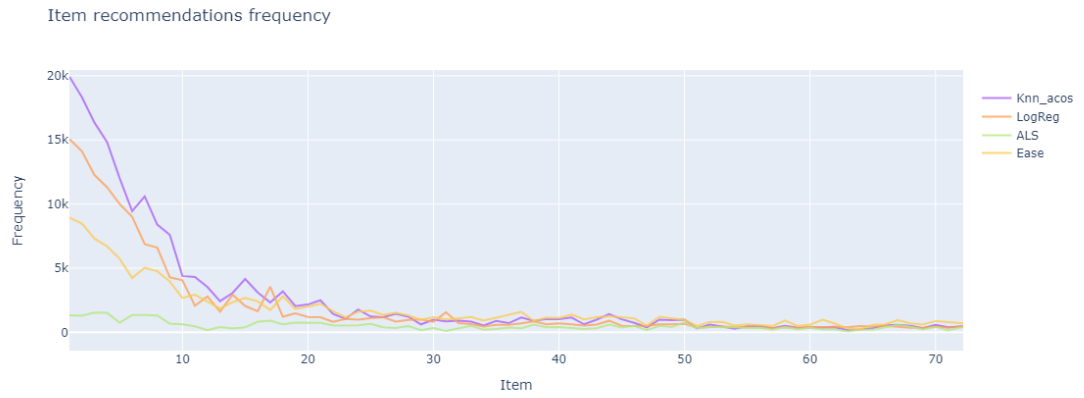

Figure 5.14: Recommendation frequencies comparison, zoomed in.

recommends less. This is because, if a method recommends very rarely unpopular items, the times that this happens are due to a high confidence of the relevance of such items. At least, this plot gives the confirmation that the best-performing methods are not achieving such metrics just because of the popularity bias, but they are relevant recommendations across all the catalog.

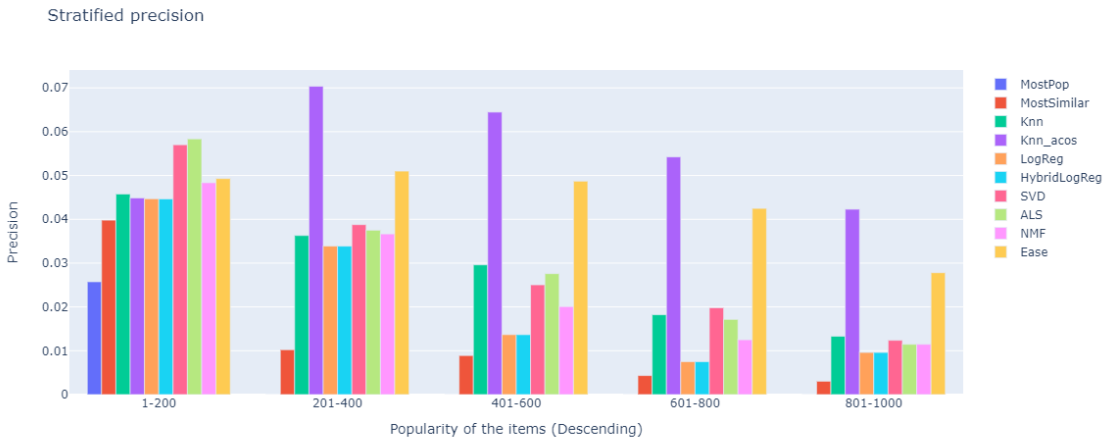

Figure 5.15: Stratified comparison.

5.3.5 Stratified test set

In order to investigate how the amount of known interactions influences the recommendations performance, the metrics are computed on some subsets of the test dataset. In particular, the methods are tested on:

- *1-item known dataset*: All the test users with exactly 1 interaction known;
- *Known users dataset*: All the test users with at least 2 interaction known;
- *Well known users dataset*: All the test users with at least 5 interaction known.

The performances in the subsets of the test set can be seen in Table 5.5. The amount of known purchases influences each method in a different way. EASE^{R} and latent factors models benefit from an extended knowledge of the user, while memory-based models and logistic regression do not. This is because EASE^{R} and latent factors models improve their performances with more knowledge, in particular the *hit ratio*, while the other methods *hit ratio* get worse. This is due to the fact that, if an user consumed many items, it is likely that such items are the most popular ones, and thus, methods that exploit the popularity bias cannot recommend those items, because they have already been consumed. In contrast, latent factors models build better user and item embeddings, thanks to the increased amount of knowledge. In the same way, item-item similarity is improved in EASE.
This comparison shows how different methods perform in different situations, and so how a different model can be used for each user, depending on the available knowledge.

Method	Users Subset	mAP@10	Rec@10	HR@10	CC@10
MostPop	One item	0.0602	0.1500	15.72%	1.0%
	Known	0.0306	0.0833	10.17%	1.0%
	Well Known	0.0202	0.0554	10.21%	1.0%
MostSimilar	One item	0.0418	0.0806	8.51%	98.1%
	Known	0.0214	0.0593	7.31%	98.7%
	Well Known	0.0165	0.0473	7.99%	76.4%
Knn	One item	0.0792	0.1491	15.57%	99.5%
	Known	0.0479	0.1130	13.81%	98.3%
	Well Known	0.0360	0.0879	15.36%	94.9%
Knn_acos	One item	0.0944	0.1938	20.24%	90.8%
	Known	0.0579	0.1379	16.73%	76.5%
	Well Known	0.0444	0.1129	19.16%	28.3%
LogReg	One item	0.0869	0.1774	18.55%	85.1%
	Known	0.0377	0.0931	11.64%	99.2%
	Well Known	0.0295	0.0778	14.58%	81.0%
HybridLogReg	One item	0.0869	0.1774	18.55%	85.1%
	Known	0.0377	0.0930	11.64%	99.2%
	Well Known	0.0294	0.0777	14.58%	80.9%
SVD	One item	0.0686	0.1174	12.36%	100.0%
	Known	0.0463	0.1019	12.84%	100.0%
	Well Known	0.0486	0.1111	19.12%	97.7%
ALS	One item	0.0710	0.1274	13.36%	100.0%
	Known	0.0435	0.0993	12.51%	99.8%
	Well Known	0.0439	0.1057	18.46%	96.3%
NMF	One item	0.0594	0.1101	11.58%	99.4%
	Known	0.0427	0.0974	12.39%	96.9%
	Well Known	0.0410	0.1005	17.85%	90.3%
$EASE^R$	One item	0.0877	0.1713	17.95%	96.4%
	Known	0.0599	0.1389	17.36%	96.8%
	Well Known	0.0634	0.1402	24.27%	77.7%

Table 5.5: Stratified test set results. Best results over each subset are highlighted.

6

Handling user cold starts

The majority of the customers who visit the website are cold starts. Every possible user is or was a cold start, before its first purchase. Given these facts, the development of a specific method for cold starts can significantly improve the business impact of the recommender system.

6.1 Behavioral data

Since most of the users register on the website only to complete a purchase, demographic data is not available for cold starts at production time. Without past interactions and without user features, the only data left is the behavior of the user on the website. In particular, every visit to a webpage specific to an item is recorded as implicit feedback. In this way, a different and more noisy form of interactions is obtained: $T = \{(u, i, t) \in \text{visits}\}$, where u is the user, i the item, and t the time at which the user u visited the item i. There is a significant difference in the usage of this kind of feedback with respect to the purchase interactions: If a cold start user saw only a subset of the catalog before buying, its next purchasing is guaranteed to be one of the visited items. For this reason, the system should

propose only previously consumed items, which is completely different from the standard case, where already purchased items cannot be suggested (ref:4.1).

6.2 Evaluation methodology

The recommendations for cold starts can be tested on the first purchase of each user. This is because each user, before its first purchase, was a cold start. The general idea is to consider the first purchase of each user as the test set. The training set is the set obtained with all the user-item visits until the day before the first purchase, for each user. In this framework, considerations about coverage and recommendation frequencies do not apply. Collaborative filtering techniques cannot be applied without specific adaptations. The relevance can be measured with the hit ratio, which is a standard accuracy-related metric when there is one test interaction per user, like in this framework.

6.3 Methods

In this chapter, R is the rating matrix built on the interactions between the user and the item. A couple of cold starts specific heuristics are proposed.
The first one, given a target user, suggests the items that that user visited the most. It is further referred as MostView.

$$\hat{r}_{ui} = |\{(v, j, t) \in T : v = u, i = j\}| \tag{6.1}$$

The other heuristic, LastView, suggests the items that the target user has visited, ordered from the most recently visited to the oldest visited item. If an item is visited more times, the most recent visit is considered.

$$\hat{r}_{ui} = \max \{t : \exists (u, i, t) \in T\} \tag{6.2}$$

6.3.1 EASE$^{\mathrm{R}}$

A simple collaborative filtering method that can be adapted to this context is EASE$^{\mathrm{R}}$. In order to allow it to suggest already consumed items, the step where the diagonal of B is set to 0 (Equation 4.26) is removed, allowing item self-similarity. Instead, since the most similar item to a specific item is itself, the diagonal is set to the maximum value in B:

$$\hat{B}_{ij} = \begin{cases} \max\limits_{(i,j)\in\{1,\dots,m\}^2} B_{ij} & \text{if } i = j \\ -\frac{\hat{P}_{ij}}{\hat{P}_{jj}} & \text{if } i \neq j \end{cases} \tag{6.3}$$

6.4 Results

Here the methods are compared. It is important to underline that the dataset used here is heavily different from the standard one, so the results should not be compared. Since the visited items are guaranteed to include the first purchase, the hitted proportion of users is expected to be much higher.

	HR@1	HR@10
MostPop	1.9%	6.4%
MostView	19.1%	35.4%
LastView	12.1%	32.8%
EASE$^{\mathrm{R}}$	13.0%	34.1%

Table 6.1: Cold starts results.

As reported in Table 6.1 good improvement is achieved by using behavioral data, instead of the popularity-based baseline. Suggesting the item most visited by the user seems to be the best strategy. However, more than in the standard case, such an offline evaluation does not reflect the true performances, because there

is a strong leakage between the train and the test set. A collaborative approach, $EASE^R$, outperforms the popularity baseline. In a production environment, the ability to recommend unseen items can make a difference, so an online evaluation should be considered in this task.

7
Conclusions

This work addresses the problem of making good recommendation when only implicit feedback is available. It proposes several collaborative filtering techniques and evaluate them on a real-world transactions dataset. This field is challenging, due to issues such as popularity bias, sparsity and fairness in the evaluation process. We saw on a real task that the popularity bias and the cold starts are strongly present and how they affect in an important way the system. In fact, at the end of the analysis, there is not a clear best method. Depending on the business application, one can argue in favour of one specific method on the basis of the domain necessities, such as explainability, coverage or relevance.

The final message that we would like the reader to understand, is that recommendation systems are not a fully automatable field, in which it is sufficient to choose the method with the best metric. On the contrary, a deep understanding of the field is required by the data scientist, who knows how to read the metrics, taking into account the present biases, and finally, who knows how to decide which is the best configuration of the system, taking into consideration all the needs of the application domain.

References

[1] M. Mansoury, H. Abdollahpouri, M. Pechenizkiy, B. Mobasher, and R. Burke, "Feedback loop and bias amplification in recommender systems," 2020. [Online]. Available: https://arxiv.org/abs/2007.13019

[2] N. J. Belkin and W. B. Croft, "Information filtering and information retrieval: Two sides of the same coin?" *Commun. ACM*, vol. 35, no. 12, p. 29–38, dec 1992. [Online]. Available: https://doi.org/10.1145/138859.138861

[3] J. Weston, H. Yee, and R. J. Weiss, "Learning to rank recommendations with the k-order statistic loss," in *Proceedings of the 7th ACM Conference on Recommender Systems*, 2013, pp. 245–248.

[4] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," 2012. [Online]. Available: https://arxiv.org/abs/1205.2618

[5] J. F. Silva, N. Moura Junior, and L. Caloba, "Effects of data sparsity on recommender systems based on collaborative filtering," 07 2018, pp. 1–8.

[6] H. Abdollahpouri, R. Burke, and B. Mobasher, "Managing popularity bias in recommender systems with personalized re-ranking," 2019. [Online]. Available: https://arxiv.org/abs/1901.07555

[7] B. Lika, K. Kolomvatsos, and S. Hadjiefthymiades, "Facing the cold start problem in recommender systems," *Expert Systems with Applications*, vol. 41, no. 4, Part 2, pp. 2065–2073, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417413007240

[8] D. Das, L. Sahoo, and S. Datta, "A survey on recommendation system," *International Journal of Computer Applications*, vol. 160, pp. 6–10, 02 2017.

[9] P. Lops, M. de Gemmis, and G. Semeraro, *Content-based Recommender Systems: State of the Art and Trends*, 01 2011, pp. 73–105.

[10] M. Y. H. Al-Shamri, "User profiling approaches for demographic recommender systems," *Knowledge-Based Systems*, vol. 100, pp. 175–187, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705116001192

[11] R. Zhang, Q.-d. Liu, Chun-Gui, J.-X. Wei, and Huiyi-Ma, "Collaborative filtering for recommender systems," in *2014 Second International Conference on Advanced Cloud and Big Data*, 2014, pp. 301–308.

[12] R. Burke, "Knowledge-based recommender systems," *Encyclopedia of library and information systems*, vol. 69, 05 2000.

[13] E. Ç ano and M. Morisio, "Hybrid recommender systems: A systematic literature review," *Intelligent Data Analysis*, vol. 21, no. 6, pp. 1487–1524, nov 2017. [Online]. Available: https://doi.org/10.3233%2Fida-163209

[14] C. C. Aggarwal, *Recommender Systems - The Textbook*. Springer, 2016.

[15] J. Carbonell and J. Goldstein, "The use of mmr, diversity-based reranking for reordering documents and producing summaries," in *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 335–336. [Online]. Available: https://doi.org/10.1145/290941.291025

[16] F. Aiolli, "Efficient top-n recommendation for very large scale binary rated datasets," in *Seventh ACM Conference on Recommender Systems, RecSys*

'13, Hong Kong, China, October 12-16, 2013, Q. Yang, I. King, Q. Li, P. Pu, and G. Karypis, Eds. ACM, 2013, pp. 273–280. [Online]. Available: http://doi.acm.org/10.1145/2507157.2507189

[17] Y. Zhang, "An introduction to matrix factorization and factorization machines in recommendation system, and beyond," 2022. [Online]. Available: https://arxiv.org/abs/2203.11026

[18] D. Billsus and M. J. Pazzani, "Learning collaborative information filters," in *Proceedings of the Fifteenth International Conference on Machine Learning*, ser. ICML '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, p. 46–54.

[19] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," 2009. [Online]. Available: https://arxiv.org/abs/0909.4061

[20] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," 12 2008, pp. 263–272.

[21] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 42, no. 1, pp. 80–86, 2000. [Online]. Available: http://www.jstor.org/stable/1271436

[22] A. CICHOCKI and A.-H. PHAN, "Fast local algorithms for large scale nonnegative matrix and tensor factorizations," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E92.A, no. 3, pp. 708–721, 2009.

[23] H. Steck, "Embarrassingly shallow autoencoders for sparse data," in *The World Wide Web Conference on - WWW '19*. ACM Press, 2019. [Online]. Available: https://doi.org/10.1145%2F3308558.3313710

[24] P. Borwein, "An efficient algorithm for the riemann zeta function," 1995, pp. 29–34.

[25] G. K. Zipf, *The psycho-biology of language: An introduction to dynamic philology*. Routledge, 2013.

Acknowledgments

I would like to express my gratitute to all the member of the Statwolf team, the company where I did my internship on which this thesis is based. In particular, I thank Chiara and Luca, who closely followed me in the development of this project.