FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

# Assignment of master's thesis

| | |
|---|---|
| **Title:** | Visual evaluation of recommender systems |
| **Student:** | Bc. Jan Šafařík |
| **Supervisor:** | Ing. Vojtěch Vančura |
| **Study program:** | Informatics |
| **Branch / specialization:** | Knowledge Engineering |
| **Department:** | Department of Applied Mathematics |
| **Validity:** | until the end of summer semester 2022/2023 |

## Instructions

Survey recommender system algorithms and evaluation of their performance, including types of biases. Design and create an environment that facilitates developing and testing recommender system models. The solution will include a Python library with interfaces for implementing custom models and a web application that provides previews of recommendations, metrics obtained by model evaluation, and information extracted from the interaction matrix. Create dashboards that allow an analysis of metrics used for model evaluation, analysis of biases typical of recommender systems, and visual analysis of user/item latent space. Verify the dashboards and the web application functionality on at least two different datasets and three different models.

*Electronically approved by Ing. Karel Klouda, Ph.D. on 4 January 2022 in Prague.*

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

# Visual Evaluation of Recommender Systems

***Bc. Jan Šafařík***

Department of Applied Mathematics
Supervisor: Ing. Vojtěch Vančura

May 4, 2022

# Acknowledgements

I would like to thank the entire Recombee research team for their support and valuable advice. The time spent together with them gave me the strength and motivation to write this work. Special thanks are given by name to Ing. Vojtěch Vančura and doc. Ing. Pavel Kordík, Ph.D.

# Declaration

In Prague on May 4, 2022 . . . . . . . . . . . . . . . . . . .

## Citation of this thesis

Šafařík, Jan. *Visual Evaluation of Recommender Systems.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

# Abstrakt

Cílem práce je prozkoumat nové způsoby evaluace doporučovacích systémů, především za pomoci vizualizačních nástrojů. V současném výzkumu je při vyhodnocování úspěšnosti modelu kladen důraz především na přesnost doporučení, méně už však na další důležité metriky, jako je rozmanitost doporučovaných věcí, rozsah pokrytí katalogu nebo možnost objevování nových věcí na úkor bestsellerů. Výsledky evaluací jsou navíc typicky prezentovány pouze ve formě agregovaných hodnot a není dále zkoumána jejich distribuce přes validační množinu uživatelů, což je důležité pro odhalení neférovosti modelu a chyb různého druhu. Práce proto nabízí sadu nových přístupů, které kombinují techniky vizualizace se stávajícími metodami offline evaluace, a které jsou implementovány jako součást frameworku zpřístupněného pod open-source licencí dalším výzkumníkům.

**Klíčová slova** doporučovací systém, vizualiazace, evaluace, framework

# Abstract

The thesis aims to explore new ways of evaluating recommender systems, mainly with the help of visualization tools. In the current research, when evaluating the success of a model, the emphasis is mainly on the accuracy of recommendations but less on other important metrics, such as the diversity of recommended items, the extent of catalog coverage, or the opportunity of discovering novel items at the expense of bestsellers. Moreover, evaluation results are typically presented only in the form of aggregated values, and their distribution through the validation set of users is not further investigated, which is essential for revealing the unfairness of the model and discovering various biases. Therefore, the thesis offers a set of new approaches that combine visualization techniques with existing offline evaluation methods and are implemented as part of a framework made available under an open-source license to other researchers.

**Keywords** recommender system, visualization, evaluation, framework

# Contents

# List of Figures

# List of Tables

# Introduction

With the enormous popularity of recommender systems in recent years, hundreds of new approaches and methods have emerged, primarily aimed at improving the accuracy of predictions over previous solutions. Such effort creates an endless race to get the best average metric value, which says very little about the actual quality of the recommender system. The real quality judges are the individual users working with the system, whose complex and varied needs and interests cannot be overlooked. For this reason, it is necessary to look at other aspects, such as the diversity and novelty of recommendations from the users' point of view or catalog coverage and promotion of niche products from the vendors' point of view.

Several recommender frameworks have been published to simplify access to the beyond-accuracy metrics and metrics for revealing the unfairness of models or different types of biases. Although these frameworks contain many evaluation mechanisms and a rich set of integrated models, they do not bring anything progressive into the array of evaluation procedures.

The Repsys framework proposed in this work is unique because it comes with a new approach to presenting the results through an interactive web application. The measured metrics are displayed to analyze their distribution, which, combined with the projection of users into two-dimensional space, brings a novel technique to various biases detection. The previous works also do not offer the possibility of dataset analysis by taking users and items visualizations and uncovering possible clusters. Furthermore, the lack of a simulation environment makes other frameworks relatively static. On the other hand, in Repsys, one can simulate user behavior by interacting with the catalog of items and then observe how the recommendations change.

In this work, the first chapter will summarize basic recommendation approaches, characterize typical biases, then describe offline evaluation mechanisms together with different metrics, whether focused on the accuracy or other aspects of system behavior. The chapter will conclude with an overview of visualization techniques and current solutions of recommender frameworks.

The second chapter will present the Repsys framework, first highlighting its main benefits and features, then showing its interface with the description of the SDK library. This chapter will also describe how the framework performs preprocessing of the dataset, its visualization, and subsequent evaluation of implemented models. The last section will show previews of the web application and possibilities of its use in examining the gathered results.

The third and fourth chapters are then focused on verifying the basic functionalities of the framework on a set of experiments using two different datasets and three recommender models. At the end of the work, further plans for expanding the framework are presented.

This thesis's goal is not to provide a comprehensive analysis of a wide range of models on various datasets to describe their biases or unfairness but to create an easy-to-use environment that other authors can employ in their research and motivate them to study more aspects of recommender systems than just the isolated accuracy.

Chapter 1

# Related work

## 1.1 Task definition

Recommender System (RS) task is to select from many items available the ones that will potentially be most interesting to users [9, 10, 11]. To perform this, RS typically has several data sources on which to make predictions:

**Item Attributes Data** Items are characterized by attributes that can take on different types, such as numbers, categories, tags, or unstructured text. Specific attributes vary by the RS domain and may, for example, include a product's name, price, description, or set of images [9].

**User Attributes Data** Like items, users also have various attributes, which can be, for example, a user's location, age, or gender [9]. However, this data is often not available due to privacy reasons [12].

**Interaction Data** The most crucial information for RS are historical data of interactions between users and items. These can be, for example, viewing product detail, adding him to a cart, buying it, marking him as liked/disliked, or giving him the number of stars from 1 to 5, where the last two are examples of explicit interactions, while the others are examples of implicit. The data are commonly transformed into an interaction (or rating/ranking) matrix where each row is a user, each item is a column, and corresponding explicit ratings (or implicit feedback) are at the position of the user-item pair [9, 10].

The application of RS can be understood in different ways depending on the specific employment. Common dividing from the perspective of later models evaluation is the rating prediction and ranking task [13]. The rating prediction consists of predicting the rating a user could give an item, where the rating is typically a numeric value from a given range. The ranking task comes from the Information Retrieval (IR) domain and involves ranking the items so that the larger the value, the more likely the item will be attractive to the user. This approach is widely used for evaluation nowadays as it better reflects the way items are presented to the user. This task is typically related to the top-N recommendation, of which all the items rated, only the top ones are chosen, which are typically the first to be seen by the user and have the best chance of catching the user's eye [14, 9].

## 1.2 Recommendation approaches

Different classifications of the RS approaches can be found in the literature, but the prevalent divides them into three main categories: content-based filtering, collaborative filtering, and hybrid techniques [9, 15, 16].

### 1.2.1 Content-based filtering

Content-based filtering comes from the domain of IR systems and therefore uses similar techniques and approaches [17]. The main principle is a recommendation based on a previous analysis of the items the user interacted with and positively rated. This analysis aims to build a user's profile that characterizes his preferences and is used when recommending new items. A wide range of information describing the characteristics of each item can be encoded into a feature vector that is compared to the user profile, and the items that are most similar to the profile are searched for [11, 18, 17].

The advantage of this approach is that it is easy to explain why a particular item was recommended, and it is possible to add new items to the catalog easily without having to wait for a sufficient number of ratings, as is the case with collaborative filtering [18]. On the other hand, it suffers from overspecialization when the system repeatedly recommends items similar to a user profile [11, 18, 17].

### 1.2.2 Collaborative filtering

Collaborative Filtering (CF) is a group of methods that uses an interaction matrix to create a recommendation model [9]. Unlike content-based filtering, it relies not only on the active user but also on all users in the system [11]. The basic idea behind this approach is that if someone shares preferences for a particular group of items with other users, they are likely to share them for other items. [10]. CF algorithms are divided into two groups: memory-based, which uses the user database directly to create predictions, and model-based algorithms, which use the user database to learn a predictive model by which recommendations are then made [19]. The prevailing memory-based algorithms are neighborhood-based methods, further divided into user-based and item-based [16, 11].

The user-based CF, also known as user-based k-nearest neighbors CF or user-user CF, was the first method used for automatic collaborative filtering, and it was introduced in the GroupLens Usenet articles recommender [20]. This method finds a group of users whose preferences are closest to the active user and combines their ratings to recommend new items, as shown in figure 1.1. There are different metrics for measuring similarity between individual users. The most famous are Pearson correlation, which calculates the statistical correlation between commonly rated items, and cosine similarity, where users are represented as vectors, and their distance in space is calculated. The ratings are combined by computing the average, unweighted or weighted, where similarities between the users may be used as weights [10, 11, 16, 9].

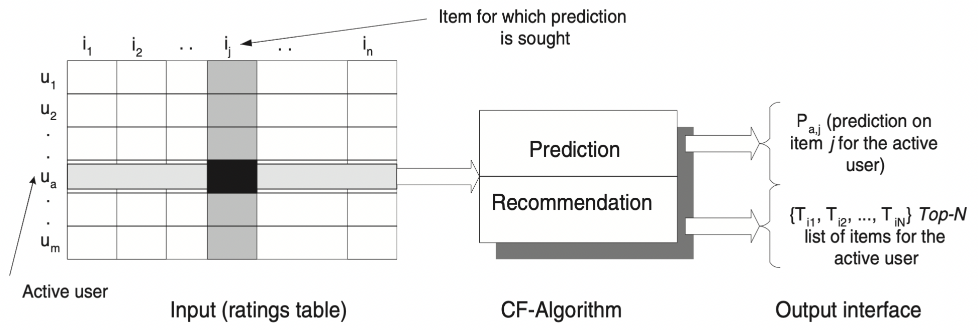

Item for which prediction is sought
$i_1$ $i_2$ . . $i_j$ . . $i_n$
$u_1$ $u_2$ . . $u_a$ . . $u_m$
Active user
Input (ratings table)
Prediction
Recommendation
CF-Algorithm
$P_{a,j}$ (prediction on item $j$ for the active user)
$\{T_{i1}, T_{i2}, ..., T_{iN}\}$ *Top-N* list of items for the active user
Output interface

Figure 1.1: Schema of the user-based CF [1].

The item-based CF, also known as the item-based k-nearest-neighbor CF or item-item CF, was developed to solve the problem of the scalability of the user-based approach. There are many more users than items in a typical RS, so more computational resources are needed to find the closest users over time as the database grows. The item-based CF was first described in an article by Sarwar et al. [1], while Amazon [21] also implemented it at a parallel time. Instead of looking for similar users, the method looks for similar items to those the user has already rated before. Two items are considered similar if users have rated them in the same way, and thus these users also have similar preferences for similar items, as shown in figure 1.2. Measuring the distance of adjacent items and then aggregating the results coincides with the procedures described in the user-based CF. The advantage of this method is that it is easily scalable, as it calculates similarities over a limited number of items that usually do not change as quickly as users in the system. Moreover, if a user's preferences change, it does not affect the representation of items as much as it is only one of many interactions characterizing the item [10, 11].

The memory-based methods have several advantages. They are straightforward to implement and often have fewer hyperparameters. Their predictions can be explained to the user retrospectively, which increases confidence in the recommendation system. Another essential advantage is scalability since

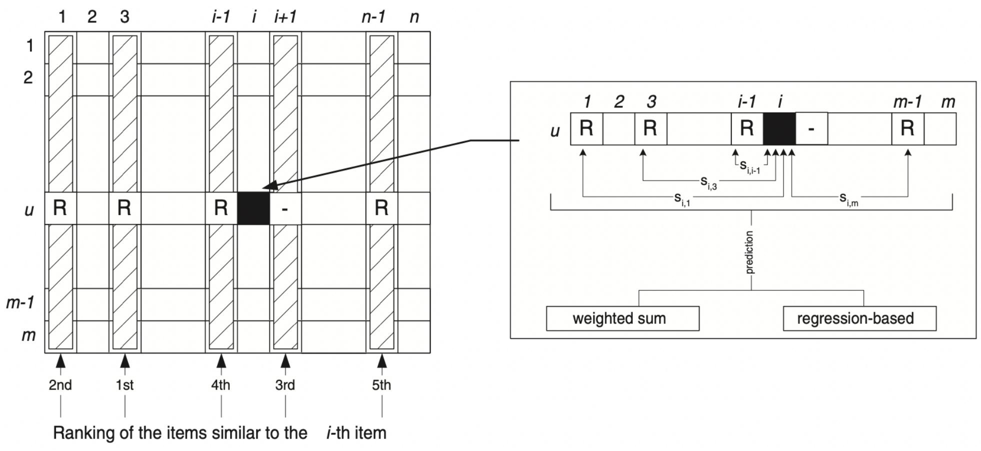


Figure 1.2: Schema of the item-based CF [1].

neighbor-based methods do not need to be wholly retrained when adding new users or items to the system. Although the prediction process has greater computational complexity than the model-based methods, they do not usually need initial time to train themselves, and the prediction can be accelerated by offline precomputing the similarity matrix [11].

The model-based methods work on machine learning principles, where a model is first trained on training data and then makes predictions capable of covering complex non-linear patterns in the unknown data [16]. This group includes several approaches, such as Bayesian networks working with a probability model, clustering methods performing clustering of similar users and classification into these classes, and rule-based methods that look for associative rules between interactions [1]. However, the most prominent methods are latent factor models, which have achieved state-of-the-art results in recent years. They work on the principle of factorizing the rating matrix to user and item factors, representing the characteristics of users and items in the latent space [22], as shown in figure 1.3.

Matrix Factorization (MF) is, generally speaking, the expression of the original matrix by the product of two or more matrices. There are several techniques for MF, such as QR decomposition, LU decomposition, or Singular Value Decomposition (SVD) [23], where the latter has been used already in the IR domain for obtaining latent factors. The SVD method gained the most attention in the RS area in 2006 during the Netflix Prize competition, where Simon Funk proposed an iterative approximative approach for calculating SVD factorization by gradient descent procedure, which allowed its application to large datasets [24]. This approach minimizes approximation error when looking for such low-rank matrices, whose product returns the matrix as close to the original one and allows for predicting unknown user-item pairs values and thus producing recommendations [23].

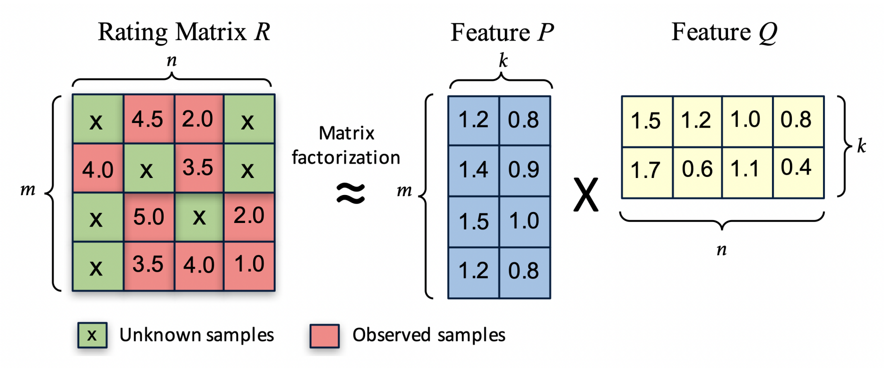


Figure 1.3: Schema of the matrix factorization [2].

It is advantageous for a real-world application of RS if the model can make recommendations for new users without re-learning the predictive model or making additional reparametrization of the model. That is why methods inspired by the neighborhood-based approach have been developed, which can represent users as a combination of item features without solving user-specific parametrization [25, 26].

Since the task of RS can be seen as the ranking problem, which does not predict specific values, Cremonesi et al. [26] proposed the PureSVD algorithm, where unknown user-item pairs are replaced by zero, thus allowing the application of fast SVD algorithms to the decomposition of sparse matrices, as they do not lack any missing values. The ranking prediction for the new user can thus be calculated as a dot product of the user's rating vector and the matrix of similarity of items obtained during the SVD factorization.

With the increasing success of Deep Learning (DL) in past decades, it has also started to be applied in the field of RS. These methods have achieved state-of-the-art results in many different areas, as they can cover complex non-linear relationships in data and thus provide a new perspective on recommendation problem-solving. In the field of RS, various techniques of DL methods are applied, such as autoencoders, convolutional neural networks, attention models, or recurrent neural networks. The latter one has achieved great results in modeling sequential data, so it has started to be applied in combination with others in session-based recommender algorithms [27].

### 1.2.3 Hybrid techiques

Hybrid techniques can incorporate several different approaches for higher performance and lower impact of their weaknesses [15, 28]. The most common combination is CF with some other technique, for example, to solve the problem of cold-start users [28]. There are several strategies [28] for combining the predictions of two or more recommender systems, such as the following.

**Weighted** The results of all available systems are weighted together.

**Switching** The systems are being switched between based on specific criteria.

**Mixed** Results of each system are presented simultaneously.

**Cascade** The following system uses the results of the previous system.

### 1.2.4 Session-based recommendation

The session-based recommendation is a domain that has become more intensively addressed in recent years. Historically, public datasets contained complete user profiles covering their long-term activity within the system. However, this information is not typically available as many users access the website anonymously (without logging into the system), and therefore RS has only a short-term history of interactions on which to respond as the user's session is ongoing. Various approaches to solve this issue were introduced, and the proper evaluation setup was also examined [29, 30].

## 1.3 Biases in recommender systems

The importance of biases examination in RS is steadily growing, as shown by the number of publications written in the last years [31], and it has even expanded into many areas like the psychology field. Melchiorre et al. [32], for example, examined the different success rates of the recommendation algorithm for users with different personality traits, which were obtained based on music preferences.

Similar research by Wang and Chen [33] has also explored user biases combined with personality traits, but this time from the perspective of beyond-accuracy metrics like diversity, novelty, or serendipity. The authors discovered different metrics values for users with different characteristics, which showed the unfairness of selected recommender models.

The coverage of such a complex topic is outside this work's scope; nevertheless, some excellent works like [31] offer a comprehensive overview of definitions, characteristics, and debiasing methods for interested readers.

### 1.3.1 Feedback loop

To understand the effects of each type of bias at different points of the recommendation process, it is necessary to specify an event loop that describes the relations between a user and model. This loop shown in figure 1.4 is called a feedback loop and is constructed as follows: the user creates data through his interactions, which are used to train the model, making predictions for the user and influencing him in his future decisions [31].

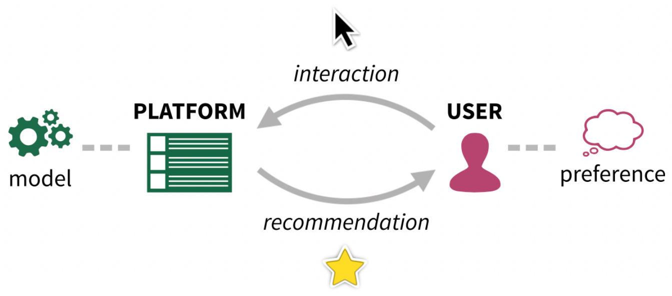


Figure 1.4: Feedback loop in the recommender system [3].

Some papers deal with the potential effects of feedback loops, such as [3], where it was shown that the decision-making process must consider the long-term effect, as one-step analysis cannot fully understand the system's dynamic behavior. Furthermore, another experiment has shown that the feedback loop leads to the homogenization of user behavior, both at the global level of all users and groups of similar users [34]. It was also shown that the feedback loop amplifies existing biases in the recommender systems [35, 31].

### 1.3.2 Biases in data

This group of biases typically comes from the data collection process, when the gathered interactions are only a snapshot of the real-world data influenced by many factors. Such selection arises in the deviation of training data from the test data, leading to the recommender model's suboptimal solution [31].

One of the biases, called selection bias, says that collected ratings do not form a representative sample, as the user is free to determine which items will rate mainly based on individual preferences. The free choice results in the hypothesis that the data is not missing at random [31, 36].

Exposure bias is based on the nature of implicit interactions. Unlike explicit ones, they only provide partial information about the user's behavior, and since the user is only exposed to a fraction of items, it is unclear whether they are missing because the users did not see them or did not like them [31].

The last-mentioned here is position bias, which derives from the items' position in the list of recommendations, as users tend to interact with those in higher places, even if this does not truly reflect their actual preferences [37].

### 1.3.3 Popularity bias

One long-standing problem with collaborative filtering algorithms is that they mainly focus on popular items with many ratings but less on unpopular and less well-known ones. This effect, called popularity bias [38, 39, 40], harms both users, who repeatedly receive only popular products and items where popular ones become more and more popular, and the others become ignored over time [39, 41]. It has also been shown that when recommending less rated items, the accuracy of the model decreases, which is called the long tail recommendation problem [38].

The long-tail expression comes from the book by Anderson [42] describing the phenomenon where niche product sales can eventually grow into a significant portion of all sales. In addition, the sale of long-tail products leads to higher profits because the less popular ones can be sold at a lower price with a higher margin than the popular items [41]. For example, it was discovered that 30-40% of books sold by Amazon are books that are not ordinarily available in mainstream brick-and-mortar stores, and at the time of the study, they had a revenue of $1 billion per year [43].

Such an approach is related to the Pareto Principle (also called the 80/20 rules), which says that a small portion of products (typically 20%) produces a large portion of profits (typically 80%). This principle is typically applied to analyze rating distribution in RS. While the long-tail items have the least number of ratings (in terms of frequency distribution), they generate 20% of total interactions. The rest, 80% of the distribution, is called short-head [41]. Figure 1.5 shows this principle on the MovieLens dataset, where it is apparent that a small portion of top-rated items (the short-head) contains much more ratings than the rest (the long-tail) [4].

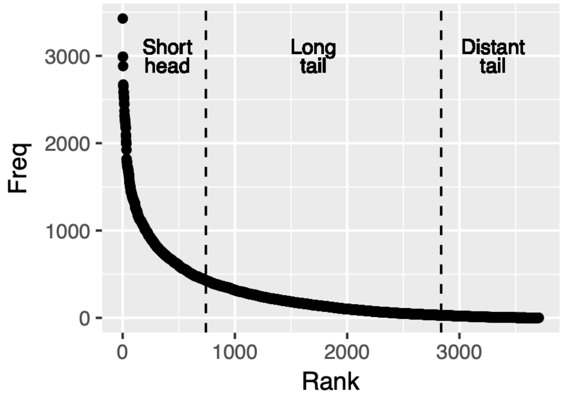


Figure 1.5: Rating frequency distribution in the MovieLens dataset [4].

Abdollahpouri et al. [40] debate that the recommendation from the long-tail is crucial for the success of RS, as most users know popular items while picking from the less popular ones determines whether the system is capable of producing personalized predictions. Their work also shows one of the possible approaches for finding a balance between the coverage of the long-tail items and the model accuracy while solving the multi-objective task. In the article of the same team of authors [44], an interesting relation was found between the number of ratings and the distribution of popular items in the user profile, which says that the larger the user's profile, the more unpopular items there appear over time. They also stated that users interested in less popular items are more active in terms of contributions of ratings than other users, which means their experience with RS should be considered as they can be viewed as stakeholders of the system.

### 1.3.4 Unfairness

The definition of unfairness is that a computer system systematically and unfairly discriminates against some individuals or groups of individuals in favor of others. Unfair discrimination then means restraining the opportunity of individuals or groups in favor of others based on unreasonable or unfounded grounds. The authors stress that if discrimination is carried out not systematically but as an effect of error, or if some form of discrimination occurs for well-founded reasons not on unreasonable bases such as race or gender, it is not necessarily an expression of unfairness [45].

One of the possible causes of unfairness is the data imbalance in representing different groups based on attributes like race, age, gender, education level, and others. Such data used during training causes the model to learn from more represented groups, resulting in discrimination when the predictions do not correspond to the actual preferences of disadvantaged groups [46].

Unfairness can also be observed at the level of personal preferences. In the article by Abdollahpouri et al. [44], a different propagation of the popularity bias was observed for different users, divided based on the degree of interest in popular items. From the example given in their work, if a user positively rates 70 popular and 30 long-tail items, he should get recommendations corresponding to this distribution, which a set of experiments refuted for several models. The authors referenced the definition of calibration fairness [47], which says that the recommendations should reflect various interests of the users based on what they have interacted with in the past.

As fairness becomes a more and more studied topic, different definitions and variants emerge, so in the survey of Chen et al. [31], they were summed up into the following groups.

**Fairness Through Unawareness** The model is fair if it has no access to any sensitive attributes about users used during the training.

**Individual Fairness** A model is fair if it produces similar recommendations for similar users. Such similarity can be measured in different ways.

**Group Fairness** Predictions produced by a model are similar for the group of users sharing similar attributes. This type of fairness includes demographic parity, predictive quality, or equality of opportunity.

**Counterfactual Fairness** The distribution of predicted values stays intact in a world where the user's sensitive attributes differ casually.

## 1.4 Evaluation process

The evaluation process is an essential step in verifying the quality and ability of RS, and many excellent works have been written on this topic, describing various techniques for recommender systems comparison and the pitfalls they can bring [14, 48, 49, 50]. The following section summarizes the primary steps, such as dataset selection and splitting, choosing suitable metrics, and aggregating the measured results.

### 1.4.1 Online vs. offline

The evaluation can take place online or offline. Online evaluation offers the possibility of comparing RS in real-life conditions and can determine whether the system is as good as in the test environment. In A/B testing, a small group of users is usually redirected to a new system, and a change in their behavior is observed. Such change is manifested, for example, by the user starting to follow recommended items more often, resulting in being put in the basket and purchased, while for another system, these suggestions may be avoided. The test group of users should be selected at random to maintain fairness when comparing the two systems. Although online evaluations are robust, they are costly and take a long to collect enough data [49, 14].

For this reason, recommendation algorithms are first preselected using offline evaluation, which consists of calculating a set of metrics utilizing historical data with user interactions captured over a certain period. When selecting a sample of data, it is necessary to avoid various types of data biases that may arise. When establishing an offline evaluation protocol, it is essential to set a process that reflects as much as possible the conditions in production [49].

### 1.4.2 Data selection

The first step of the offline evaluation is data preprocessing, and it may involve filtering and pruning, removing ratings too much in the past, or converting explicit interactions to implicit ones.

Pruning aims to reduce data size or improve prediction accuracy [51]. Beel and Brunel [5] showed that many researchers either employ some form of pruning or operate with datasets already pruned. In the recent variants of the MovieLens dataset, the authors removed all users who had less than 20 ratings because researchers typically need enough ratings per user for accurate evaluation and because of the influence of the previous RS on new users, which led to an untrue representation of their preferences. Figure 1.6 shows that 16% of users rated less than ten movies, and 26% rated between 10 and 19 movies. The pruning is also applied for items, where a minimum number of interactions must be performed with an item to be kept [52, 53, 54].

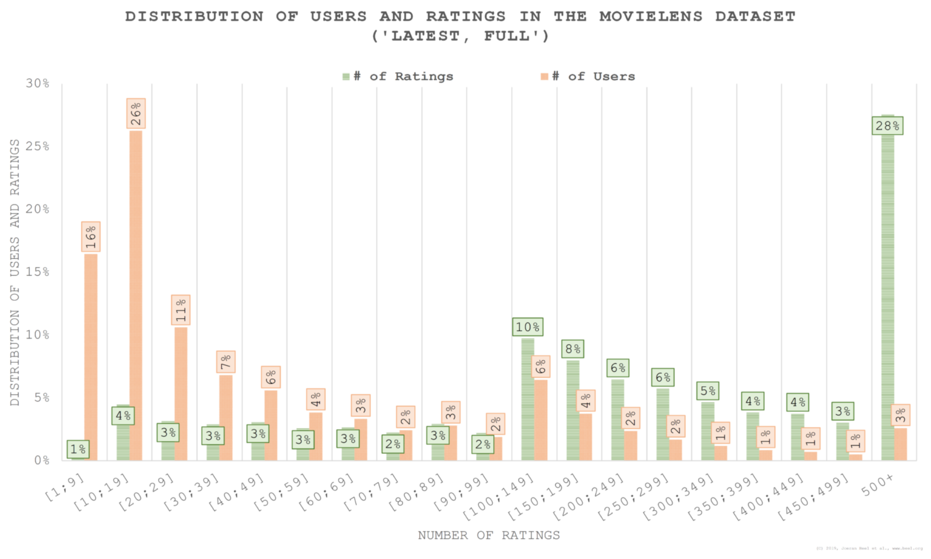


Figure 1.6: Distribution of ratings in the MovieLens dataset [5].

Another type of data pruning is addressed in [51], where different algorithms concerning the timestamp of the interaction were tested. The article argues that many methods for CF do not take time into account and ignore the phenomenon of shifting interest, which has a considerable influence on the user's behavior over time. It has been shown that, despite the removal of a large number of older interactions, the accuracy of predictions has either been improved or at least maintained, with still maximum catalog coverage and a considerable reduction in data size.

Some CF methods work better or only with implicit interactions, so procedures for converting explicit ratings into implicit ones are involved. This approach can more precisely capture user preferences as only high ratings given to an item express genuine interest. For datasets like MovieLens or Netflix Prize, it is typical to keep only ratings of four and higher, as they range between one and five stars [52, 53, 54, 13].

The final thing to consider when working with large datasets is whether the user-item ratings are missing at random. The false illusion can be found in many CF-oriented papers and can result in biased parameter estimation for methods using clustering, matrix factorization, or probability models [36]. Little and Rubin's theory of missing data [55] implies that ratings are missing not at random (MNAR), as users choose which items to rate and which not to rate based on their preferences. It is always good practice to incorporate some of the missing data mechanisms. Marlin and Zemel [36] proposed that evaluation should be done against randomly selected items of held-out data instead of items a user chose to rate because it more reflects a goal of RS, which is recommending new items not seen before.

### 1.4.3 Data splitting

Before model evaluation, the dataset must be split into a training, validation, and test set. Several splitting strategies were developed for this purpose, an overview of which based on their occurrences in the literature is summed in Meng et al.'s [6] article. Both validation and test sets will be viewed as a single one for further simplification, as they are created on the same bases.

**Leave One Last** The last interaction from the user's history is used for the test and the previous ones for training. The advantage of this approach is that as much of the data is kept for training. On the other hand, it does not adequately capture users' behavior at different time points. In addition, due to the inconsistent determination of the period for the last interaction, information only available in the future may appear during the training [6].

**Temporal User/Global Split** Instead of using one interaction, this strategy performs a split into two parts in a specific ratio. There are two variants of how to do this. The first one, called temporal user, uses a portion of the last interactions separately for each user, and the second one, called temporal global, specifies a single global time boundary applied to all users, above which all interactions are used for the test. The advantage of the second variant is that data only available in the future can no longer appear during the training. However, using a global time point may result in no test interactions or very few training interactions available for some users [6]. A comparison of both is shown in figure 1.7.

**Random Split** This strategy divides the interactions randomly into a training and a test set while the time sequence is not considered. The disadvantage of this strategy is the difficulty of reproducibility and the problem of future data leakage during training [6].

**User Split** Instead of splitting the interactions, users are split into training and a test set [6]. While previous strategies only measure the model's generalization to unrated items of the same users (weak generalization), this strategy measures the model's generalization to utterly new user profiles (strong generalization). The evaluation process is as follows: first, the model is trained using all interactions of the training users. Then, the interactions of the test users are divided into an observed and held-out part, where the first is shown to the model to learn user preferences. Finally, the predictions are compared to the held-out part to compute the metrics [56]. The disadvantage is that many CF methods do not support predicting for new users [6]. On the other hand, this strategy is viewed as more realistic because it corresponds to the production usage of RS, and it was used, for example, in [52, 54, 53].

Even though publicly available datasets contain millions of interactions, some datasets may be significantly smaller a do not have enough data for the proper evaluation. For this case, the cross-validation method is used [56]. The most common form is k-fold cross-validation, where the data is evenly divided into $k$ parts (folds), which are repeatedly selected so that one fold is used for testing and the other $k-1$ folds for training. In this way, each data point appears at least once in the evaluation [57].

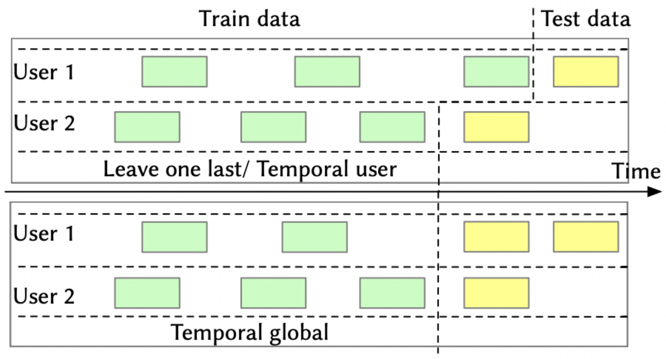


Figure 1.7: Comparasion of different splitting strategies [6].

### 1.4.4 Accuracy metrics

When selecting metrics for the model evaluation, it is distinguished between ranking and rating prediction tasks [13]. In the first case, the accuracy of the predicted value is measured against the rating user gave to an item. Various metrics come from machine learning or statistics literature, but the most common is the Root Mean Square Error (RMSE), which is specific as it penalizes more significant deviations than the other metrics. The less commonly used is the Mean Square Error (MSE) or the Normalized Mean Average Error (NMAE), which normalizes the predicted values in order to be able to compare results across different datasets [49]. Usage of the Mean Absolute Error (MAE) in some older papers is also possible to find [1]. The RMSE metric is defined as 1.1, where $\hat{R}$ is a list of predictions for a user-item pair $\hat{r}_{ui}$ and $r_{ui}$ is the correct rating value.

$$RMSE = \sqrt{\frac{1}{|\hat{R}|} \sum_{r_{ui} \in \hat{R}} (r_{ui} - \hat{r}_{ui})} \tag{1.1}$$

The second set of metrics suitable for evaluating a rating task is based on the IR theory. A user typically enters a query to the system, which returns a set of documents sorted according to their similarity to the that query [58]. Many metrics have been designed for measuring the quality of results, but only the most used ones in the RS domain will be presented here.

The first metric is Precision@N, which expresses the ratio of relevant items between top-N recommendations versus the number of recommended items $N$. This metric makes little sense in absolute terms, as only a small fraction of relevant items are observed and therefore usually has a low value. The reason is related to the MNAR phenomenon when it is not possible to tell only from the data whether non-rated items are irrelevant to the user or he has just chosen not to rate them [13]. In definition 1.2, $rel_i$ represents a function that returns 1 if there is a match at a position $i$, else zero.

$$Precision@N = \frac{\sum_{i=1}^{N} rel_i}{N} \tag{1.2}$$

The second metric is Recall@N, which expresses the ratio of relevant items between top-N recommendations and the number of relevant items. It has the advantage over the previous metric that if we assume that relevant items are missing at random (while other items may be missing not at random), this metric can be calculated without the MNAR bias effect [13]. In definition 1.3, $I_u$ is a list of relevant items rated by a user.

$$Recall@N = \frac{\sum_{i=1}^{N} rel_i}{min\,(N, |I_u|)} \tag{1.3}$$

The last one is Normalized Discounted Cumulative Gain (NDCG@N) [59], which compares the order of top-N items among the ideal order of the same items. Unlike the precision and recall, NDCG is top-weighted, so incorrect predictions at higher positions are more penalized as they have more impact on the user. The relative penalties come from the definition 1.4 of Discounted Cumulative Gain (DCG). Also, the ranking from a user perspective is needed to calculate the maximal possible DCG and use it for normalization [14].

$$DCG@N = \sum_{i=1}^{N} \frac{2^{rel_i} - 1}{\log_2{(i+1)}} \tag{1.4}$$

It is usual for most metrics to be calculated only for the top-N recommendations, denoted by the symbol @N. This cut-off approach best represents the natural usage of RS, where the user typically sees only a few items from the many available [14].

### 1.4.5 Beyond-accuracy metrics

The quality of RS can be viewed in various ways, but based on definitions from the service industry domain, the real judge of quality are the users interacting with the system, which can be both customers or vendors, and the expectations of both groups should be met [60]. Although the accuracy of recommendations is a significant indicator of the quality, it is far from the only one and should be used in combination with other equally important. Instead of looking at recommendations as on the individual items, it is necessary to see the whole list as the user sees it. Users do not need precise recommendations but meaningful and pleasant ones that meet their expectations. When selecting between two algorithms, it is necessary to compare both from other aspects than just the accuracy because even if two algorithms are equally accurate, they can differ significantly in other factors that the user discovers very quickly and can have a positive but also a very negative impact on him [61].

Over the past decade, many articles have been devoted to the beyond-accuracy metrics. However, as these are still relatively new concepts, their definitions vary on how the authors perceive the nature of each. Covering these perspectives is beyond the scope of this work. However, several high-quality papers have been produced for this purpose [48, 62, 63], which provide a comprehensive overview of the definitions alongside the approaches for recommender algorithms adjustments.

The first two metrics are novelty and diversity. In comparison, novelty is the difference between the present and the prior experience, while diversity stands for the difference between the current set of items. For example, if a user is recommended a song that he has never heard, it can be said that this is the expression of a novelty; on the contrary, the recommendation is diverse if it contains songs of different genres instead of different songs of the same genre. It should be borne in mind that the two concepts described are very close and influence each other. Although there are other metrics from the beyond-accuracy category, these two are considered the most important. As it is very complicated to model user preferences that are complex and change over time, using diversity as a complementary metric to accuracy makes it possible to increase the chance that if RS does not hit the user's preferences exactly, at least something will appear in the list of recommended items that will please or surprise him [63].

The most common definition of diversity is an average intra-list diversity or an average distance between items within a list of recommendations [64]. This definition can be expressed as 1.5, where $R$ is a list of top-N predictions, and $dist(i, j)$ is a function returning the distance of items $i$ and $j$. The distance metric varies in the literature, and it can be the complement of Jaccard similarity, cosine similarity, or Hamming distance if the items are represented in the form of rating vectors [48].

$$Diveristy@N = \frac{\sum_{i \in R} \sum_{j \in R \setminus \{j\}} dist\left(i, j\right)}{|R| \left(|R| - 1\right)} \tag{1.5}$$

Novelty is often defined as the opposite of popularity, implying that an item is novel if very few people know it. Popularity, in this case, is described as the probability that a random user knows the item, which is additionally scaled using an inverse logarithm to achieve a decreasing function giving more meaning to the rare items. In definition 1.6, $R$ is a list of top-N recommendations, and $p_i$ is the popularity of an item expressed as the ratio of users who have interacted with the item at least once [63].

$$Novelty@N = \frac{\sum_{i \in R} - \log_2 p_i}{|R|} \tag{1.6}$$

Diversity and novelty can sometimes be seen as objectives opposite to accuracy, where it is necessary to find their optimal trade-off while solving the multi-objective problem. If the system starts recommending novel items, higher diversity also appears both from the perspective of users and the system as a whole [65]. Although experiments showed that more diverse recommendations might reduce accuracy, users still preferred getting various items [61].

A metric often confused with novelty is serendipity. While novel items may not be necessarily surprising to the user, serendipitous recommendations are the ones a user would not expect, and they can be surprising as he would not usually discover them by himself [60, 50]. The metric is expressed as 1.7, where $R_{unexp}$ is a subset of top-N recommendations $R$ containing unexpected items, and $R_{useful}$ is a subset of items useful to the user. The unexpectedness is measured using primitive prediction models, which are assumed to produce obvious items without any surprise. Such unexpected items are obtained by subtracting $R$ from the items recommended by the primitive model [60].

$$Serendipity@N = \frac{|R_{unexp} \cap R_{useful}|}{|R|} \tag{1.7}$$

Another essential metric is coverage, which most often occurs in the form of item coverage as the ability of RS to cover the catalog of items [60, 50]. Moreover, it is distinguished between prediction coverage, which determines the coverage without any cut-offs, and the second, catalog coverage, limited only to top-N predictions, thus more approximating the typical usage of RS. As coverage decreases for smaller values of $N$, it is often practical to find a threshold for which the coverage rate stabilizes [60]. In definition 1.8, $R_u$ is a list of top-N recommendations for user $u$, and $I$ is a catalog of available items.

$$Coverage@N = \frac{|\bigcup_{u \in U_t} R_u|}{|I|} \tag{1.8}$$

Within the CF algorithms, RS is more accurate the more ratings of the users it has access to. As popular items have much more interactions than others, a system with the aim of the highest accuracy will recommend the short-head items, which goes against trying to cover as much of the catalog as possible. Another finding is that increased coverage does not necessarily lead to higher serendipity; on the contrary, higher serendipity leads to higher coverage over time. Thus, if serendipity needs to be increased, it is also necessary to extend the catalog coverage to rare and less popular items [60].

### 1.4.6 Long-tail metrics

As discussed before, popularity bias is an often problem. Therefore, several metrics have been described in the literature to measure how well the RS recommends from the class of long-tail items. The first is the Average Percentage of Long-tail Items (APL), which measures the ratio of long-tail items in each user's list of recommendations [4]. In definition 1.9, $R$ is a list of the top-N recommended items for a user, and $\Gamma$ is a set of the long-tail items.

$$APL@N = \frac{|\{i, i \in (R \cap \Gamma)\}|}{|R|} \tag{1.9}$$

The second metric is the Long Tail Catalog Coverage (LCC), which measures the same ratio but within all recommendations over the entire catalog of items. This metric is beneficial because it may happen that RS will recommend the same subset of long-tail items for each user so that the APL metric will be very high, but the rare items will be very little covered [4]. The definition 1.10 is similar to the definition of classic coverage, but it contains an intersection with the set of long-tail items $\Gamma$.

$$LCC@N = \frac{|(\bigcup_{u \in U_t} R_u) \cap \Gamma|}{|\Gamma|} \tag{1.10}$$

### 1.4.7 Metrics aggregation

Most metrics are calculated for each user, resulting in an array of values in which aggregation has to be applied to express the final value. Such aggregation may be a simple average, median, or geometric mean [14]. The weighted average was also employed, where the weight was determined as the number of items a user rated, so the evaluation took more accent on active users [13].

Ekstrand et al. [66] emphasize the benefit of distributions over the point estimation of metric values, as only in this way critical aspects of RS efficiency can be captured between individual user groups. They propose using a marginal distribution for each stakeholder group separately and across them, monitoring distribution changes at different periods, and comparing shifts in distribution to the current system or against the ideal target.

### 1.4.8 Evaluation protocols

As Said et al. [67] demonstrated, many factors can influence the final value of metrics, even if they are calculated on the same datasets. These factors include various pruning and splitting strategies, specific implementations of recommender algorithms, hyperparameters settings, the strategy for selecting candidate items, or the exact procedure for measuring model performance. As there is no unified system for RS evaluation, comparing two models under different conditions is meaningless. It was also debated that a single evaluation system such as the one used in the Netflix Prize competition led to a significant shift in the area of RS because it was possible to compare algorithms fairly. Therefore, it is essential to precisely report all steps taken during the experiments to reproduce the results by others later.

The importance of a detailed evaluation protocol is also underlined in the work by Meng et al. [6], which sets out several steps that need to be fulfilled when publishing the results. It is necessary to describe a specific strategy for splitting the dataset with details about data pruning; splitting parts of the data should be publicly released, and standardized tools for model evaluation should be used. The authors also recommend employing the temporal global splitting strategy as being the most realistic.

## 1.5 Visualization techniques

Visualization techniques help understand the results obtained during the experiments, as they bring a new perspective on a problem and allow looking for hidden patterns or anomalies that are not obvious when examining raw data.

One of the methods is a projection of latent space using dimension reduction (DR) tools, which embeds a high-dimensional representation of an item into a space with a dimension of two or three [7]. Although it is possible to set any output dimension in DR methods, not all of them are intended for visualization purposes [68]. It must be kept in mind that usage of DR for visualization can often be misleading if the hyperparameters are wrongly selected. Methods can create clusters not present in the original data or display similar pairs far apart and vice versa [68].

In the literature, two categories of methods for visualization using DR are distinguished: local methods, which try to preserve the local structure of high-dimensional data, and global methods, which try to preserve the global structure of the data. Examples of global methods are principal component analysis (PCA) [69] and multidimensional scaling (MDS) [70], but these cannot capture the complex non-linear structure in the data [68] and will not be further discussed. On the other hand, more novel local methods exist, such as t-distributed stochastic neighbor embedding (t-SNE) [71], uniform manifold approximation and projection (UMAP) [72], and the minimum-distortion embedding (MDE) framework [7].

The t-SNE method constructs probability distributions between two points in a high-dimensional space and tries to find such an embedding representation in a low-dimensional space to minimize the Kullback-Leibler divergence between the previous and current distribution. The main hyperparameter of this method is perplexity, which sets the effective number of neighbors, and the larger the value, the more uniform the distribution of probabilities becomes. By contrast, a low value forces the algorithm to focus the probabilities only on the nearest points. From the predecessor, the SNE method, it differs mainly by using the Student-t distribution instead of the Gaussian distribution to measure the similarities of two points in a low-dimensional space [71].

The UMAP method was inspired by t-SNE, and its algorithm consists of two phases. First, a weighted graph of the closest neighbors is assembled, where the probability distribution represents individual weights. The second optimization phase performs moving points of this graph in a low-dimensional space so that it gives together points that are close to each other in the high-dimensional space and, conversely, it pushes points away that are far apart in the high-dimensional space. Unlike the t-SNE method, UMAP better preserves global structures and has significantly better time performance. In addition to the dimension, this algorithm accepts two other influential hyperparameters. The first is the number of neighbors, and the higher this value is, the more large structures are captured, but at the expense of averaging the smaller detailed structures during the local approximation. Thus, the lower value splits the manifold into many small connected components. The second hyperparameter is min-dist, which determines the distance to the nearest neighbor to maintain local connectivity. The smaller this value is, the more dense each region will be; the larger it is, the more points will be spread apart. The spreading solves the overlapping of points but reduces the faithfulness of the manifold structure representation [72]. Different visualizations of UMAP and t-SNE can be seen in figure 1.8.

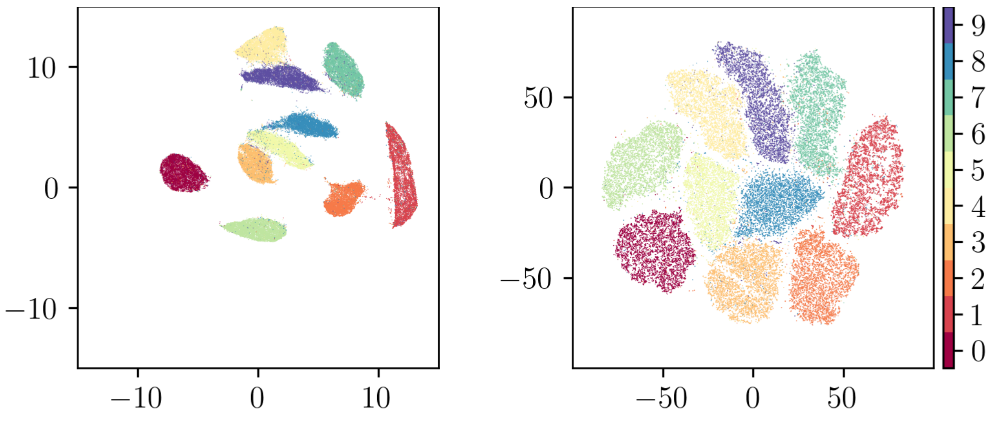


Figure 1.8: UMAP (left) and t-SNE (right) on the MNIST dataset [7].

The MDE framework generalizes the concept of measuring similarities between two points using weights or distances. A distortion function of the Euclidian distance between the high-dimensional vectors is introduced for a pair of points, and the goal is to find such embeddings that minimize the total distortion. Searching for these embeddings can be influenced by various constraints, such as centering them at mean zero, fixing the positions of some existing embeddings when adding new ones, or a standardization constraint that requires embeddings to be centered while having an identity covariance.

The quality of embeddings depends on the specific implementation of the distortion function, which gets a weight of two points, and it should return a low value for similar (attractive forces) and a high value for dissimilar pairs (repulsive forces). A comparison of results for different distortion functions is shown in figure 1.9. As with UMAP, the optimization phase is preceded by building a neighborhood graph, where the edges are ranked based on the similarity of two items, and the number of nearest neighbors to be preserved can be set as a hyperparameter. An advantage of this method is the high scalability to large datasets while keeping a superior computational time, among others, thanks to GPU acceleration. In addition, both t-SNE and UMAP can be implemented using the MDE framework by setting the appropriate distortion functions, which makes this method the most versatile [7]. Unlike t-SNE, UMAP and MDE may be used not only for visualization but also generally for DR in machine learning [72, 7].

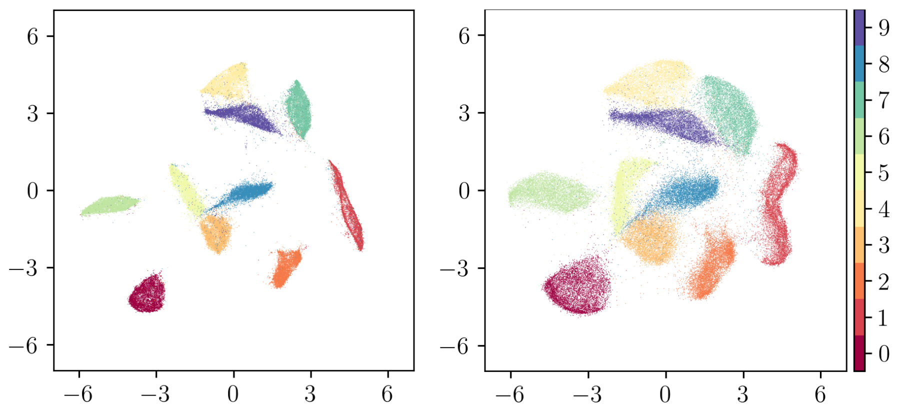


Figure 1.9: Distortion functions 1 (left) and 2 (right) of MDE [7].

The projection into the two-dimensional space can often be found in the recommender systems domain. Some works, for example, focus on developing models that involve user interactions and visual representations of items such as clothing [8, 73] or artwork [74] to construct the predictions. They encoded the products images into features vectors using various machine learning techniques, and the representations thus learned were then displayed using the t-SNE method, as illustrated in figure 1.10, which shows that similar types of clothing are close to each other while different ones are far away.

A similar approach was proposed by Shen et al. [75] in their system for visual exploration of the latent music space, where music files were used instead of images. The files were first converted to a spectrogram and then processed using DL models to create latent representations reflecting a characteristic of each musical composition. Finally, the projection with t-SNE combined with other visualization tools allowed them to discover insights about the data. Gil et al. [76] introduced the VisualRS framework for rendering a tree structure of relationships between items or users. These relationships were distin-

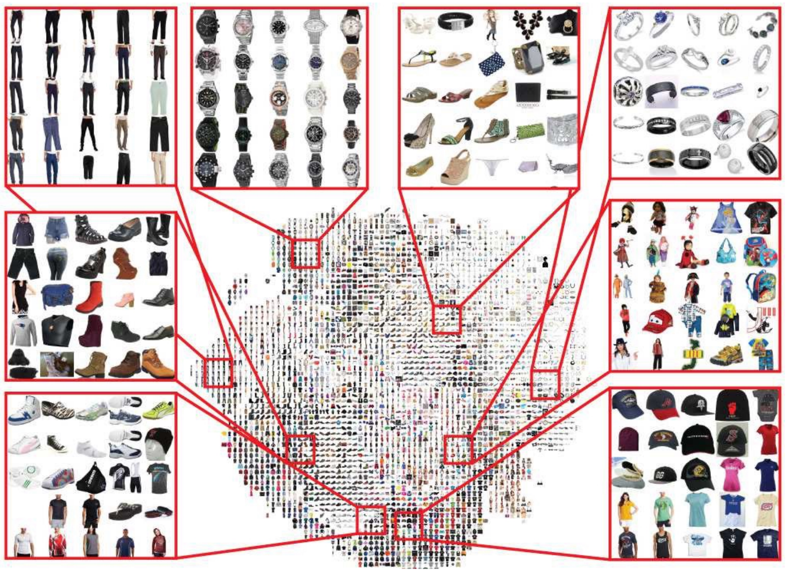


Figure 1.10: Visualization of the clothing style space with t-SNE [8].

guished graphically using different colors and node sizes to highlight essential similarities between two elements. It is also possible to choose from different tree charts depending on which is most suitable for the current problem.

Visualizations are not only advantageous for researchers. There are also tools designed for the end-users of RS and for whom some form of visualization can help understand the reasons behind the recommendations they received, which increases confidence in the system. Also, it leads to growing user interest in RS, higher acceptance of further recommendations, and education on how the system operates [77]. Some works like [78] offer, in addition to the explanation, the possibility to influence the weights of data sources used for the prediction and see how these weights impact the results. A similar interactive tool was proposed by O'Donovan et al. [79], in which the recommendations were explained through a graph representation of the closest users with a possibility to modify the neighborhood size and observe the changes in the list of predicted items.

## 1.6 Recommender frameworks

Over the last decade, many recommendation frameworks have been developed, most of which were motivated by the need for reproducibility and fair comparability of results obtained during experiments. The earliest frameworks consisted mainly of fine-tuned implementations of the most well-known recommender models and essential tools for offline evaluation. Some of these are LensKit [80], RankSys [81], or LibRec [82], and thanks to them, easy-to-use software for the development and testing of recommender systems was made available to both existing researchers and new entrants. Over time, other frameworks such as DeepRec [83] extended implemented models to include state-of-the-art solutions, including deep learning models.

The increased interest in reproducibility of experimental results led to frameworks oriented toward comprehensive evaluation mechanisms, such as various dataset filtering and splitting techniques, tools for hyperparameters tuning, beyond-accuracy metrics, and metrics for biases and unfairness detection. Sun et al. [84] analyzed 85 papers and summarized the fundamental factors influencing the evaluation process of RS. Then they proposed a standardized evaluation procedure based on this analysis, which they wrapped in the DaisyRec library. On the other hand, Anelli et al. [85] analyzed the most prominent recommender frameworks, and due to some of these shortcomings, they designed and created the most extensive framework called Elliot. It contains over a dozen splitting methods, around 50 strategies for hyperparameter optimization, and over 30 different evaluation metrics. The third one mentioned is the Librec-auto tool of Sonboli et al. [86], which acts as a wrapper around the original LibRec framework and aims to automate the experimental process, emphasizing the analysis of diversity and fairness in combination with the integration of re-ranking algorithms.

In addition to the issue of consistent evaluation protocols, there is a demand for recommender systems explainability, which was recently covered in the new framework called recoXplainer [87]. It offers an extendable library for developing explainable recommender systems, including multiple model-based and post-hoc explanatory techniques.

When reviewing current solutions, it is notable that some new perspective on model evaluation is missing. The metrics are often presented in plain tables having values aggregated over the users. In addition, most frameworks lack implementation of the highly discussed beyond-accuracy and long-tail-oriented metrics. The previous works also do not offer the possibility of dataset analysis by projecting users into the two-dimensional space using their interactions and discovering possible clusters based on their behavior. Furthermore, the lack of a simulation environment where one could test the RS in real-time makes other frameworks relatively static without any interactive element.

Chapter 2

# Repsys framework

Although the most recent frameworks include a variety of evaluation mechanisms and a rich set of integrated models, they do not add anything unique to the array of evaluation procedures. The Repsys framework is distinct in that it employs a novel approach to presenting results in a highly interactive format. The key features and objectives are as follows.

**Easy Usage** The interface for integrating models and datasets is intuitive. The Python SDK library has been created for this purpose, offering two basic classes where it just needs to implement a few basic methods, and the framework will take care of everything else. The system also looks at the models as black-box algorithms and thus does not require any specific modifications in model implementation beyond the primary interface. The core component is a performance-optimized web application that serves as a gateway when working with the framework, and its design is strongly focused on clarity and user-friendliness.

**Simple Control** Command-line utility has been created to operate with the framework efficiently. It can be used to run all the integrated tools, such as dataset splitting, creating visualizations, training models, and running the evaluation process. In addition, all these steps can be controlled using a configuration file to adapt the framework to a specific project fully.

**Visual Exploration** The measured metrics are examined for their distribution instead of aggregating the values across all users and displaying a single number. This approach allows locating user groups treated differently by the RS than others. In addition, these distributions are not only presented in histogram form but are also mapped into the 2D user space, bringing an additional level of bias analysis. In addition, all visualizations are interactive and, thanks to the rendering library used, offer a range of tools to customize them.

**Beyond Accuracy** Evaluating models solely on recommendation accuracy is no longer sufficient to create a system capable of production use. Therefore, the framework provides a set of beyond-accuracy metrics such as diversity, novelty, or coverage, along with metrics aimed at detecting the popularity of bias, making it possible to analyze catalog coverage and the extent of recommending long-tail items.

**Authentic Simulation** The researcher's perspective on designing a new recommender model may be diametrically opposed to how the end-user perceives it. One way to verify the functionality of a model other than by using evaluation metrics is to try working directly with the system as if it were an actual production deployment. For this purpose, it is possible to build a unique dashboard containing panels with real-time updated predictions combined with different settings or models and track changes during interaction with currently recommended items.

The aim is not to create another benchmarking system with many recommender models but a simple and easy-to-use environment that offers a new perspective on model evaluation. The framework is currently oriented only on the top-N recommendation tasks, and although the sequence-aware RS are not supported yet, they will be in the near future.

This chapter will first demonstrate integrating models and the datasets into the framework using the SDK library, the command-line utility, and the configuration file. Then there will be described the mechanisms of preprocessing the dataset, such as filtering and splitting, followed by an overview of implemented tools for projecting users and items into two-dimensional space. Finally, the main features of the integrated web application will be presented.

## 2.1 Basic usage

Working with the framework is straightforward. Create files with the models and dataset source code using the SDK library, add a configuration file with specific project settings, and run the commands from the command line utility.

The SDK library contains interfaces for importing data from any custom dataset and for the integration of any recommender model. Starting with the dataset, Repsys comes with a set of internal data types for labeling the input data, which helps the framework understand the meaning of the individual attributes. These data types are UserID (unique identifier of a user), ItemID (unique identifier of an item), Interaction (numerical value of interaction like user's rating), Title (name of an item or product), String (textual attribute), Number (numerical attribute), Category (categorical attribute with a single value), or Tag (categorical attribute with multiple values). In order to add a new dataset, an instance of the Dataset class must be created, and the following methods implemented.

**item_cols()** returns a dictionary with the item attribute names as keys and the framework data types as values.

**interaction_cols()** returns a dictionary with the interaction data columns as keys and the framework data types as values.

**load_items()** returns a data frame with a catalog of items. If there is a need for data preparation or feature extraction, it should be done here.

**load_interactions()** returns a data frame with all user-item interaction pairs. The preliminary preprocessing of interactions like converting explicit feedback to implicit should be deployed here.

```python
class MovieLens(Dataset):
    def item_cols(self):
        return {"movieId": ItemID(), "title": Title(),
                "genres": Tag(sep=","), "year": Number()}

    def interaction_cols(self):
        return {"movieId": ItemID(), "userId": UserID()}

    def load_items(self):
        df = pd.read_csv("./ml-20m/movies.csv")
        df["year"] = df["title"].str.extract(r"\((\d+)\)")
        return df

    def load_interactions(self):
        df = pd.read_csv("./ml-20m/ratings.csv")
        return df[df["rating"] > 3.5]
```

Listing 1: Integration of the ML dataset into the framework.

Listing 1 shows an example of the MovieLens [88] dataset implementation. Each attribute is assigned one of the data types, such as ItemID (movie identifier), Title (movie title), Tag (list of genres separated by a comma), and Number (shooting year). At the same time, the interactions are labeled with UserID and ItemID (pairs of users and movies). Because the rating value is omitted, the framework will automatically create a binary interaction matrix. The preprocessing of extracting the year from the movie title and filtering only interactions with a rating higher than four is also made.

When adding a new model, an instance of the Model class must be created, and the following methods must be implemented. The first three are mandatory, and the remaining two are optional.

**fit()** runs the model training or loads its latest checkpoint if it already has been trained. This method is called during the training process, before the model evaluation, or before the web application startup.

**predict()** gets an interaction matrix and returns a prediction for each user-item pair. This method is used for the recommendation previews inside the web application or during the model evaluation process.

**web_params()** returns a dictionary with parameter names as keys and the framework's UI components as values. These parameters then appear as input fields in the web application, where it is possible to set any value, which is later postponed to the prediction method.

**compute_embeddings()** gets an interaction matrix and returns a tuple of user and item embeddings. This method is used for the data visualization if the DR algorithm is set to a custom type.

```python
class PureSVD(Model):
    def __init__(self, n_factors=50):
        self.n_factors = n_factors
        self.item_sim = None

    def fit(self, training=False):
        X = self.dataset.get_train_data()
        U, sigma, VT = randomized_svd(X, self.n_factors)
        self.item_sim = VT.T.dot(VT)

    def predict(self, X, **kwargs):
        X_predict = X.dot(self.item_sim)
        X_predict[X.nonzero()] = 0
        return X_predict
```

Listing 2: Integration of the PureSVD model into the framework.

Listing 2 shows an example of the PureSVD [26] model implementation. The training method includes matrix factorization, while the prediction method uses learned latent representations to evaluate all items based on input user interactions. The predictions can be modified before returning, such as removing ratings for items with which the user has already interacted. The framework will later take care of cutting top-N predictions according to the application settings or evaluation mechanism.

### 2.1.1 Command-line utility

The framework command-line utility makes it easy to operate with the implemented tools. A basic overview of the commands is in the following list.

**repsys dataset split** splits the interactions into the training, validation, and test set. It also stores a mapping of user/item indices to their IDs.

**repsys dataset eval** runs the dataset visualization, creating an item and user 2D embeddings based on the selected algorithm. The algorithm can be one of the following: MDE [7], t-SNE [71], UMAP [72], or custom.

**repsys model train** calls the training method of implemented models. The model has access to the training data through dataset reference.

**repsys model eval** runs an evaluation process of the models using the validation set as input to the prediction method.

**repsys server** runs the web application, where the results of the visualizations, model evaluations, and recommendation previews can be observed.

### 2.1.2 Configuration file

The configuration file contains options for the dataset splitting and pruning, settings of the evaluation mechanism and visualization algorithms, or server setup. Listing 3 shows a sample of such a file with the training split portion and the minimum number of user interaction parameters, the cut-off for the diversity and coverage metrics, and the number of neighbors for the MDE method at the end of the file.

```
[dataset]
train_split_prop = 0.9
min_user_interacts = 10

[evaluation]
diversity_k = 10
coverage_k = 15,30

[visualization]
pymde_neighbors = 10
```

Listing 3: Example of the configuration file.

## 2.2 Framework pipeline

After implementing the dataset with the SDK library, the framework takes all the necessary steps to load the data correctly, create visualizations of the user/item space and evaluate the models across various metrics. All the outputs produced within each step of the framework pipeline are combined to create different views on RS analysis, as shown in Figure 2.1.

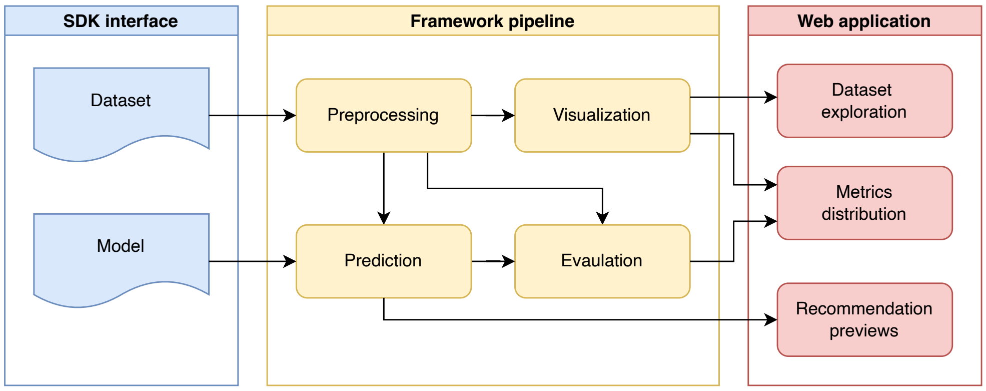


Figure 2.1: Schema of the framework pipeline.

### 2.2.1 Dataset preprocessing

There are two types of data loaded through the SDK: interactions data and items catalog. The interactions must take the form of user-item ID pairs, and they may include any additional information, such as the value of user rating or the amount of product purchased. During the validation, it is checked that all interacted items are also included in the catalog to preserve consistency.

Then, the data are pruned by two conditions: a minimum number of interactions made by a user and a minimum number of interactions with an item. Both thresholds can be specified in the configuration file, and they are both set to zero by default. Usually, the interactions are filtered out by other conditions like the minimal value of a rating, but this procedure is kept to the developer's responsibility before the data are handed over to the framework for the most versatile usage.

After that, the interactions are divided into multiple sets using the user split strategy, which is currently the only strategy available. It was selected as the best simulation of the RS production usage because it allows measuring the model's generalization to the completely new user profiles. The splitting process is shown in Figure 2.2, and it takes the following steps.

Firstly, the users are randomly divided into training, validation, and test set, while the portion of training users can be defined in the configuration file, and by default, it is set to 85%. The rest of the users are divided equally

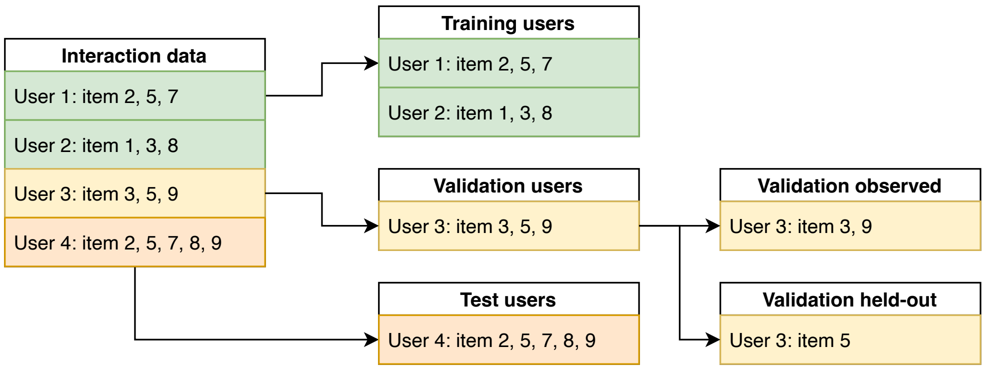


Figure 2.2: Schema of the splitting strategy.

into the validation and test sets. Then, some validation and test interactions are filtered out if they are related to an item not seen by any training user. After that, the pruning process is repeated because the data may change due to the previous steps, and some users with less than a minimum number of interactions could appear. Also, to keep the consistency of the catalog, only items interacted by the training users are kept.

Next, for each user in the validation and test set, the interaction history is split with the random split strategy by defined portion into the observed and held-out subsets, making 80% and 20% by default. This strategy can not take time to consider, but more realistic strategies like the global temporal split will be added in the future. All three sets and their subsets (in case of validation and test data) are converted into sparse matrices, where each row represents a user, and each column represents an item. When no additional information about the user-item interaction is provided, the corresponding positions are filled with number one; otherwise, the value, like the user's rating, is used. At the end of the splitting process, there are five matrices. The first is for the training, and two pairs of matrices are for the evaluation, where each pair contains one with observed and one with held-out interactions.

For the items catalog data, the ID and name of an item must be at least provided. The data are then processed depending on each attribute's data type, like precomputing distributions of numerical values or storing unique values of categorical and label attributes. The catalog is paired with the interaction data to provide various explanation details during the dataset analysis or model evaluation analysis. The catalog data are also used when previewing recommendations to simulate the production experience of the RS.

In order to keep the mapping between the user's or item's index in the interaction matrix and their original ID, an optimized bidirectional dictionary is created, which is stored alongside the split results. Thanks to this, it is easy to retrieve the ID of an item at some index in the prediction vector and use this ID to fetch additional information from the catalog and vice versa.

### 2.2.2 Models evaluation

The evaluation process aims to verify the accuracy of predictions and determine how diversified or new recommendations are and whether they cover the whole catalog of items equally. The framework is designed not only to measure the model's overall performance in aggregated form but also to make it possible to identify which groups of users are preferred by the model or exhibit different behaviors compared to others. Therefore, along with averaging measured values across all users, metrics are stored as an array of values computed individually for each user. This procedure later allows the results to be viewed as a random variable distribution, which sheds more light on how the model behaves towards different subjects. A significant advantage of this approach is that the individual metrics can be paired with the user's position in the two-dimensional space, bringing other details about various biases, such as the model's fairness within a group of users sharing similar interests.

As shown in Figure 2.3, the evaluation is directly related to the process of the dataset splitting, as described in the previous section. The model is first trained on the interactions of the training users. Subsequently, the observed interactions of the validation users are submitted to the model's input, which returns the rankings predictions for all items in the catalog. In some cases, it is usual to remove from the predictions those items that the user has already interacted with, as the model should recommend new ones that the user does not yet know. The predictions are then compared with interactions that the model has not seen (held-out data), based on which the individual metrics are calculated. Using a user split strategy and a strong generalization, the model never sees the validation users during the training process. As the implementation of the recommender model changes over time, it is helpful to know how changes affect evaluation results. For this reason, the framework stores all the results and allows a comparison of aggregated metrics values for the current and the previous evaluation performed.

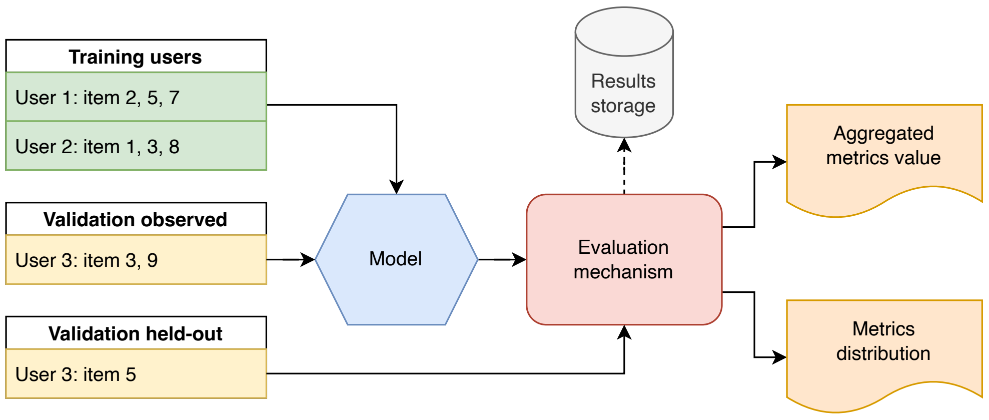


Figure 2.3: Schema of the evaluation process.

The overview of implemented metrics is summarized in Table 2.1. By default, some metrics are computed using different cut-offs selected based on the regular usage in the research papers. No cut-offs were applied to the rating prediction metrics as it makes no sense by their nature. Along with aggregated values, the distribution of metrics across validation users is available for all metrics except the coverage and long-tail catalog coverage, which are computed as a ratio of the items catalog.

| **Category** | **Metric** | **Cut-offs** | **Distributed** |
|---|---|---|---|
| Rating accuracy | RMSE | None | Yes |
| | MSE | None | Yes |
| | MAE | None | Yes |
| Ranking accuracy | Recall | 20, 50 | Yes |
| | Precision | 20, 50 | Yes |
| | NDCG | 100 | Yes |
| Beyond-accuracy | Diversity | 20 | Yes |
| | Novelty | 20 | Yes |
| | Coverage | 20 | No |
| Long-tail metrics | APL | 20 | Yes |
| | LCC | 20 | No |

Table 2.1: Overview of the implemented metrics.

The metrics are divided into four groups: accuracy of the rating prediction task, accuracy of the ranking task, beyond-accuracy metrics, and metrics oriented on the long-tail items analysis. The first group measures an error between the predicted rating and the user's actual rating, and they are implemented only for completeness, as they have been a standard method of evaluation in the past. These metrics are Mean Absolute Error (MAE), Mean Square Error (MSE), and Root Mean Square Error (RMSE). The second group of metrics is much more suitable for measuring the model's accuracy as they are more consistent with how users work with RS. Into this group belongs recall, precision, and Normalized Discounted Cumulative Gain (NDCG).

Three beyond-accuracy metrics belong to the third group: diversity, novelty, and item coverage. For the diversity metric, which measures the average distance between the items within a list of recommendations, the cosine similarity is used for the distance computation, and the ratings given to an item are used to construct the feature vectors.

The last group is added for the popularity bias detection, as the items from the short-head are recommended much more than the long-tail items. The first metric is the Average Percentage of Long-tail Items (APL), and the second is the Long Tail Catalog Coverage (LCC). The short-head is computed as 20% of the most popular items in terms of cumulative rating frequency distribution, while the long-tail is the rest 80% of items.

One of the framework's goals is a good performance on multiple levels, including the evaluation process. A usual issue spotted in several solutions is repetitive operations, like sorting the predictions by ranks constantly from the beginning to compute the metric at different cut-offs. For this reason, only the highest cut-off value is used for which the ranks are sorted to avoid unnecessary calculations. In the same way, pair-wise distances of items or distribution of long-tail items are calculated at the beginning of the process. Although it does not make the framework extremely modular, merging the computational processes for similar metrics brings a significant acceleration.

### 2.2.3 Dataset visualization

The framework provides tools for visualizing data in two-dimensional space to understand the latent structure of users and items. The process is performed by reducing the dimensionality of the interaction matrix vectors, and the embeddings are created separately for the user space and item space to explore these two groups individually, as shown in Figure 2.4. This approach is built on the assumption that rated items characterize users, and the items are characterized by users who interact with them. It is also possible to provide any other representation, such as user/item attributes or some lower-dimensional embedding vector obtained as an intermediate of the recommender model.

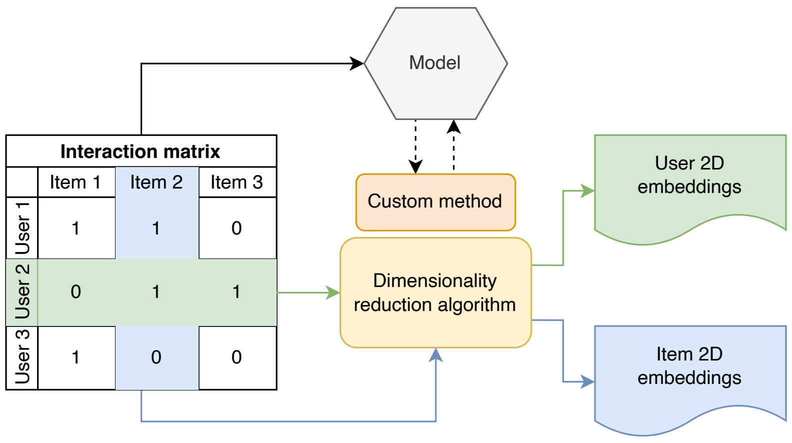


Figure 2.4: Schema of the dataset visualization.

Visualization of users is done separately for the training and test data to maintain consistency with the evaluation process. The projection of training users into the latent space helps understand the character of the dataset before any model is deployed. At the same time, the projection of test users plays a strong role when studying model evaluation results because it offers a new perspective on the distribution of the metrics. The user's position in the 2D space is paired with the metric value, allowing the detection of various biases, such as popularity bias or unfairness of the model.

The first integrated method for the visualization is t-SNE [71], whose implementation in the scikit-learn [89] library is selected. The documentation advice is followed, which says that if the input data dimension is too large, it is necessary to use another tool to reduce the dimensionality to an acceptable value. Thus, if the dimension is higher than 50, the PCA [69] method first performs the reduction. Compared to the default setting, the initialization is set to random, and the cosine metric is set for measuring distances. The perplexity is kept unchanged, but it is possible to adjust it for the specific dataset using the configuration file. The second method is UMAP [72], for which the implementation from the umap-learn [90] library is used. Again, the distance metric is changed to the cosine similarity from the default setting. The latest method integrated is the MDE [7] framework. The implementation from the PyMDE [91] library is employed, where initialization is set to random to speed up the calculation, while the number of iterations, memory size, and residual norm boundaries are increased to get more stable results. For both UMAP and MDE methods, the number of neighbors is kept unchanged, and it is configurable to be adjusted for the specific problem. Among the three methods implemented in the framework, MDE is chosen as the default one because it is highly configurable while creating competitive embeddings capturing both local and global data structures. Moreover, it is scalable to large datasets and supports GPU acceleration for fast computation.

Suppose the integrated algorithms are insufficient, or there is a need for creating embeddings not based on the interaction data. It is possible to implement a custom method, which can be used directly as a visualization technique returning two-dimensional embeddings, or as an intermediate DR algorithm. If the output dimension from the custom method is higher than two, the MDE framework is additionally applied to calculate the final reduction.

## 2.3 Results exploration

A user-friendly interactive web application has been created to explore the evaluations and visualizations made by the framework. It consists of three main components: the analysis of user and item latent space created based on the training data, analysis of the results obtained during the evaluation process, and simulation environment with previews of recommendations made by a model for existing or synthetic users.

### 2.3.1 Dataset analysis

Understanding the input data is a fundamental prerequisite for developing a successful solution for any machine learning problem. The framework allows exploring the dataset using the visualizations created using one of the algorithms for dimensionality reduction.

The first is a visualization of items in the form of a scatter plot, where each point represents one item and whose title can be displayed after the mouse hover. Individual items can be filtered based on the attributes, automatically loaded through the SDK library. For categorical and tag attributes, the filter can be picked from the set of their unique values. For the numerical attributes, all values are first divided into several bins, of which ranges are then used for the filtering. Additionally, the lasso tool enables selecting a group of points to analyze different clusters of items. The distributions of individual attribute values are calculated and then displayed in the side panel for such a cluster to provide distinctive details. An example of the various options when working with the items is shown in Figure 2.5.

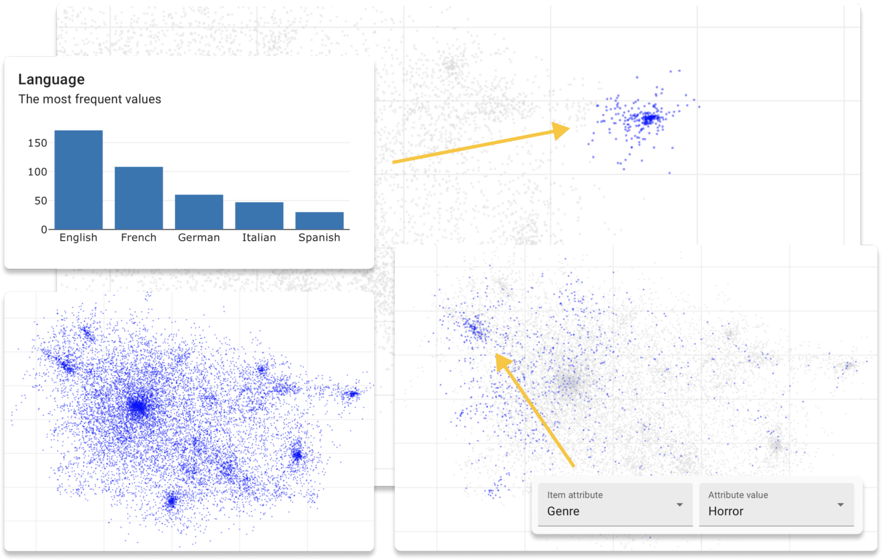
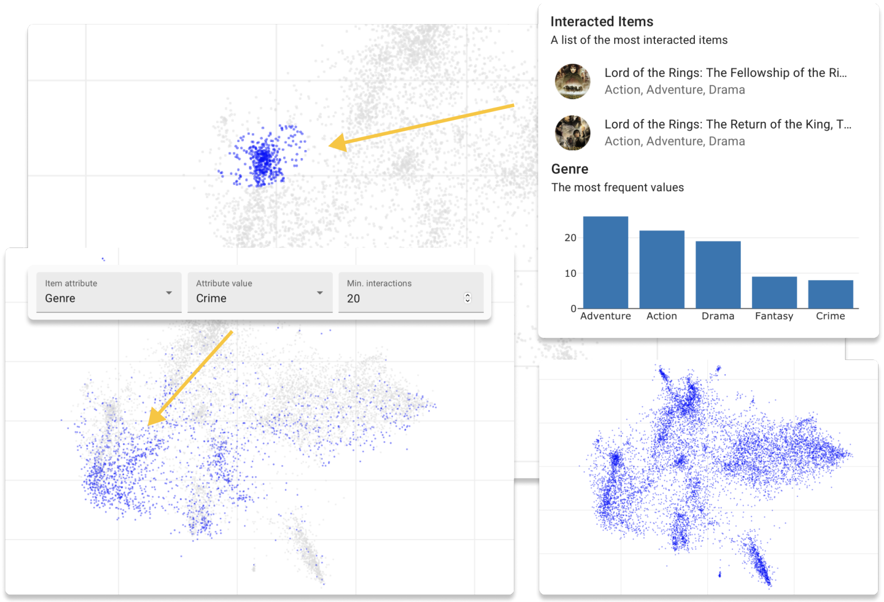


Figure 2.5: Example of the item embeddings visualization.

Similarly, the analysis of the users is performed, whose visualization is shown in the second scatter plot. Points representing individual users can also be filtered, but this time based on the attributes of the items with which these users interacted. Unlike filtering items, it is necessary to specify the minimum number of interactions the users made with a given item. It is also possible to select a group of points to analyze a cluster of users. The characteristic details are shown as a list of the most interacted items by these users, along with the attribute values distributions. Because the short-head items have much more ratings than the others, they would appear in almost every list. The TF-IDF weighting is employed to avoid this problem, where users and items replace the documents and words. An example of the described tools and visualization of the user embeddings is shown in Figure 2.6.



Interacted Items

A list of the most interacted items

Lord of the Rings: The Fellowship of the Ri…
Action, Adventure, Drama

Lord of the Rings: The Return of the King, T…
Action, Adventure, Drama

Genre

The most frequent values

Item attribute: Genre

Attribute value: Crime

Min. interactions: 20

Figure 2.6: Example of the user embeddings visualization.

### 2.3.2 Evaluation analysis

The following fundamental component of the web application is the dashboard with model evaluation results. The first panel includes aggregated metric values divided into several tabs based on the implemented models. If there is a previous evaluation for a given model, each indicator also displays how much the model has improved or worsened from the last result. There is also a bar chart to quickly compare the results across the models. An example of the metrics summary is shown in Figure 2.7.

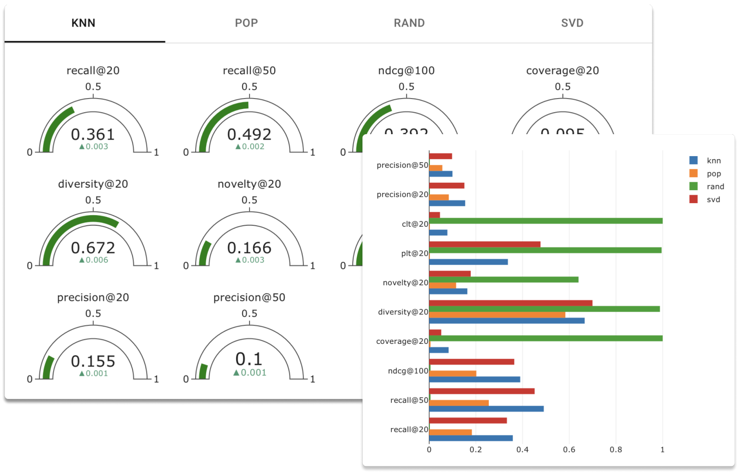
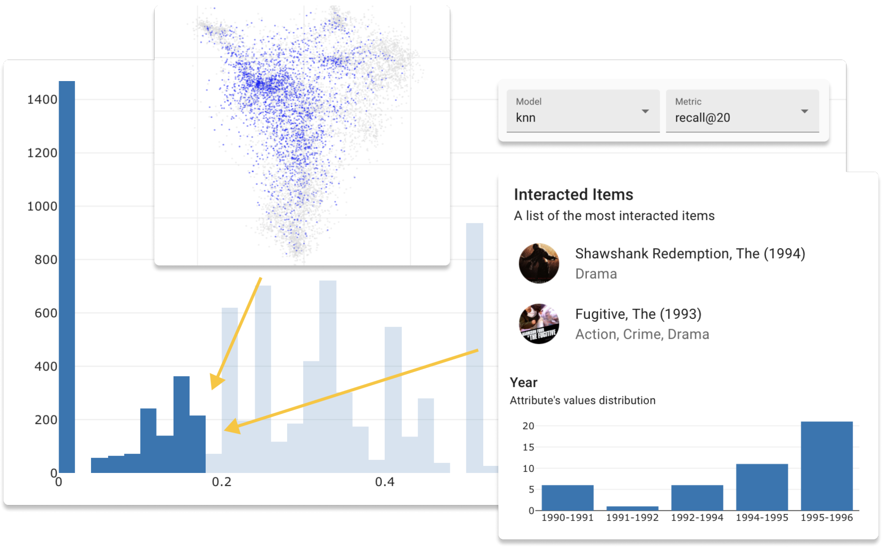


Figure 2.7: Example of the model evaluation summary.

The second panel is a histogram with the distribution of metric values calculated for all validation users. It is possible to choose any combination of the model and metric whose distribution should be displayed. Also, there is an option to select any part of the distribution representing a group of users and view more detailed information about them. Besides the most frequently rated items and the distributions of item attributes, the users are projected into the two-dimensional space, and their position is visualized in the scatter plot. An example of this selection is in Figure 2.8.



Interacted Items
A list of the most interacted items

Shawshank Redemption, The (1994)
Drama

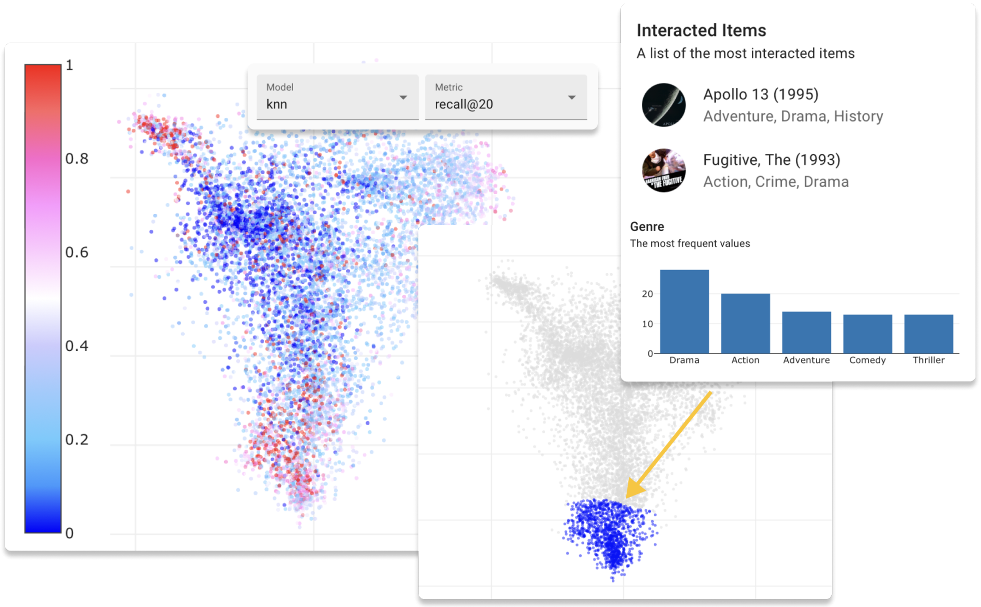
Fugitive, The (1993)
Action, Crime, Drama

Figure 2.8: Example of the metric distribution analysis.

The final panel also shows the distribution of metric values but projected into the space of user embeddings. Each point represents one validation user and is colored according to the metric value, as shown in Figure 2.9. Furthermore, different color ranges and point sizes for better visibility can be set. To better understand the differences between data clusters, it is possible to select the group of points using the lasso tool and display additional details as in the last panel. Both last two visualizations allow locating a group of users discriminated by the model or exhibiting different characteristics.

### 2.3.3 Simulation environment

Once the models are evaluated, it is helpful to test their functionality on some users, for which purpose, there is a simulation environment. The simulator has two modes: build mode and preview mode. In the build mode, it is possible to create a custom dashboard as a list of panels, where each of them represents one recommender. These panels have a title, an associated model, and a set of parameters specific to the model implementation. As mentioned in the SDK library interface, one of the methods may return a list of web parameters,


Model: knn

Metric: recall@20

Interacted Items

A list of the most interacted items

Apollo 13 (1995) — Adventure, Drama, History

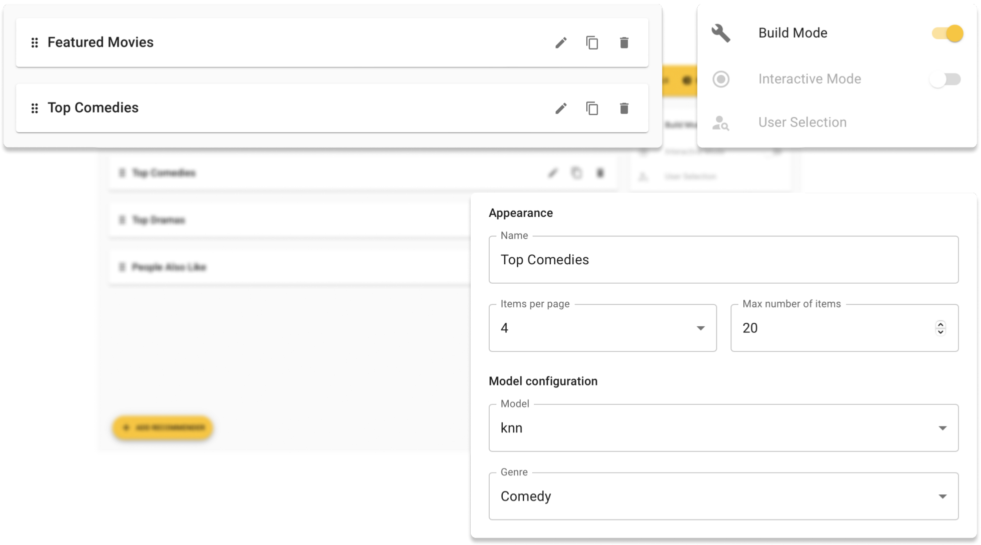
Fugitive, The (1993) — Action, Crime, Drama

Genre

The most frequent values

Figure 2.9: Example of the metric values projection.

which then appear here, and they can be set to any value, which is later passed to the prediction method. For example, adding a parameter with a set of movie genres creates a selection field, and based on the chosen genre, the predicted items are filtered out. The number of visible items with the number of total fetched items can be set to enable another level of customization. An example of the simulator in the build mode is shown in Figure 2.10.


Featured Movies

Top Comedies

Build Mode

Interactive Mode

User Selection

Appearance

Name: Top Comedies

Items per page: 4

Max number of items: 20

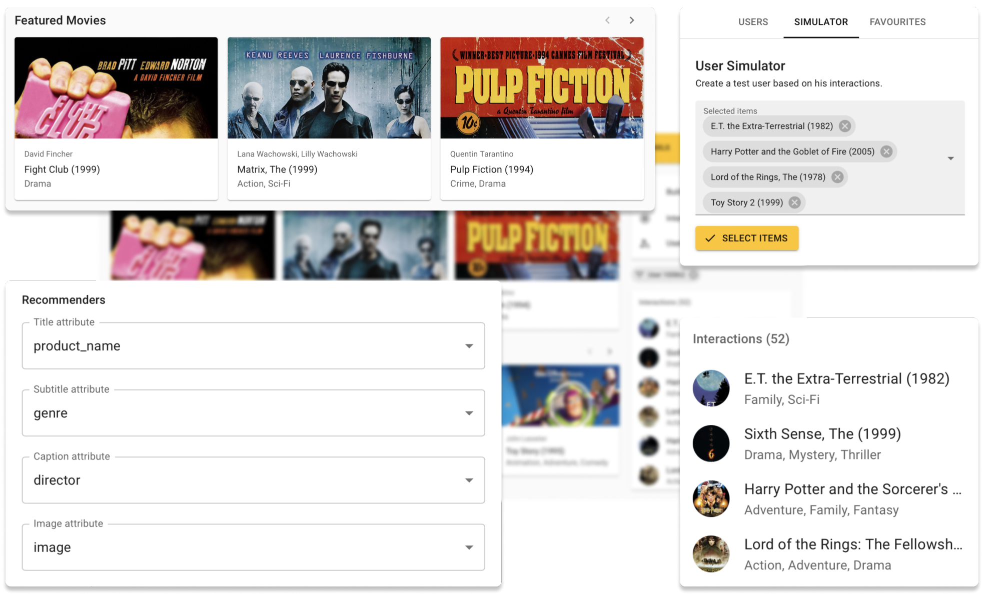
Model configuration

Model: knn

Genre: Comedy

Figure 2.10: Example of the simulator in the build mode.

In the preview mode, the created panels are transformed into lists of recommendations. The response to various input interactions can be tracked by selecting an existing user from a validation set or creating a synthetic user by assigning a list of implicit feedback on items from a catalog. Items can be searched by their title, and the searching mechanism is optimized for datasets even with thousands of products. Once a user is selected, its previous interactions are displayed on the right side, and recommendations returned by models individually for each recommender panel appear on the left side. Each panel consists of card components, the specific form of which can be customized by mapping the attributes of items to specific fields. To create the most realistic simulation of RS, one can switch to interactive mode, during which he controls the user's subsequent behavior by clicking on the recommended items and watching how the prediction changes. Figure 2.11 shows an example of the simulator in the preview mode.



David Fincher
Fight Club (1999)
Drama

Lana Wachowski, Lilly Wachowski
Matrix, The (1999)
Action, Sci-Fi

Quentin Tarantino
Pulp Fiction (1994)
Crime, Drama

User Simulator
Create a test user based on his interactions.

Selected items
E.T. the Extra-Terrestrial (1982)
Harry Potter and the Goblet of Fire (2005)
Lord of the Rings, The (1978)
Toy Story 2 (1999)

Title attribute: product_name
Subtitle attribute: genre
Caption attribute: director
Image attribute: image

Interactions (52)

E.T. the Extra-Terrestrial (1982)
Family, Sci-Fi

Sixth Sense, The (1999)
Drama, Mystery, Thriller

Harry Potter and the Sorcerer's ...
Adventure, Family, Fantasy

Lord of the Rings: The Fellowsh...
Action, Adventure, Drama

Figure 2.11: Example of the simulator in the preview mode.

Chapter 3

# Experiments setup

A set of experiments are conducted to test all the basic framework's functionalities, such as splitting the dataset, training several selected models, performing their evaluation, and creating data visualizations. The results are being compared for individual models, and differences for individual groups of metrics are described. The distribution of metrics values and the relation to users' location in two-dimensional space are also explored. In addition, the simulation dashboard is created with recommendation previews.

## 3.1 Datasets

For testing purposes, two datasets from the domain of the recommender systems are chosen. The experiments mainly focus on the first one, while the second one supplements some of the examples and shows the versatile usage of the framework and ability to adapt to other datasets.

The first one, the MovieLens (ML) [88] dataset, is probably the best known and most widely used in the recommendation system community, as it serves as a stable benchmark in the development of new models and techniques. The 20M variant contains 20,000,263 explicit interactions made by 138,493 users who gave ratings to 27,278 different movies, where the rating takes the form of stars from 0.5 to 5.0, with a step of 0.5 stars. The preprocessing follows the procedure described in [54, 53], where explicit interactions are converted to implicit by keeping only those with a rating higher than 3.5 and leaving out those users who have made less than five interactions and items with less than five ratings. The dataset contains a catalog of movies with three attributes: ID, title, and a list of genres. Also, a year of shooting is extracted from the title, and poster images are scrapped to extend the attributes.

The second dataset is the Book-Crossing (BX) [92] dataset, containing 1.1 million ratings given by 100,000 users to 340,000 books. This data comes from the online book rating community between the years 2001 and 2004. The interactions are given on a scale from 0 to 10, where values 1-10 represent explicit ratings and the zero value implicit feedback. The subset of implicit interactions is selected here following the procedure proposed in [93], and the pruning is performed in the same way as with the ML dataset. The catalog of books has four attributes: book title, year of publication, author's name, publisher, and the book's cover image. Some entries in rating history data include books with no catalog record, so these are removed.

Both datasets are split into a training, validation, and test set with the following setting: 85% of the users are used for the training, and the rest are divided equally into validation and test set. Then, 80% of the test user interactions are given the model to produce the predictions, and metrics are calculated against the remaining 20%.

Visualizations of items and users are created using the MDE algorithm by reducing the dimensionality of the interaction matrix. The parameter with the number of neighbors is set to 10 to obtain smaller interconnected clusters.

## 3.2 Models

Four models are in total implemented for the experiments, where one is a representative of the memory-based CF (User-KNN [11]), one of the model-based CF (PureSVD [26]), and two elementary models from the category of non-personalized algorithms (RAND [4] and TopPop [26]).

The User-KNN model is implemented using the neighbor search algorithm. The cosine similarity is selected to measure the distances to nearest neighbors, which are expressed by a feature vector of implicit feedback with a size of a total number of items. Feature vectors of the neighbors are weighted by their distance and summed up to get ratings for all items.

The PureSVD model is trained by performing SVD factorization of the interaction matrix, creating a latent representation of users and items according to the size of the factor determined during the initialization. The latent representation of items is then turned into a similarity matrix of each item-item pair, multiplied by the interactions of an unknown user to get the predictions.

The hyperparameters of the User-KNN and PureSVD model are chosen based on the grid search algorithm taking the values with the highest accuracy. For User-KNN, the number of neighbors $k$, and for PureSVD, the size of the latent factor $f$ was selected as follows.

**User-KNN** $k = 50$ (ML), $k = 100$ (BX)

**PureSVD** $f = 30$ (ML), $f = 50$ (BX)

The RAND model recommends items by randomly selecting the ranks from the uniform distribution. The TopPop model sums up the number of implicit feedback for each item in the training data and always recommends only the most popular items. Only the predictions for items a user has not interacted with are kept for all the above models.

Chapter 4

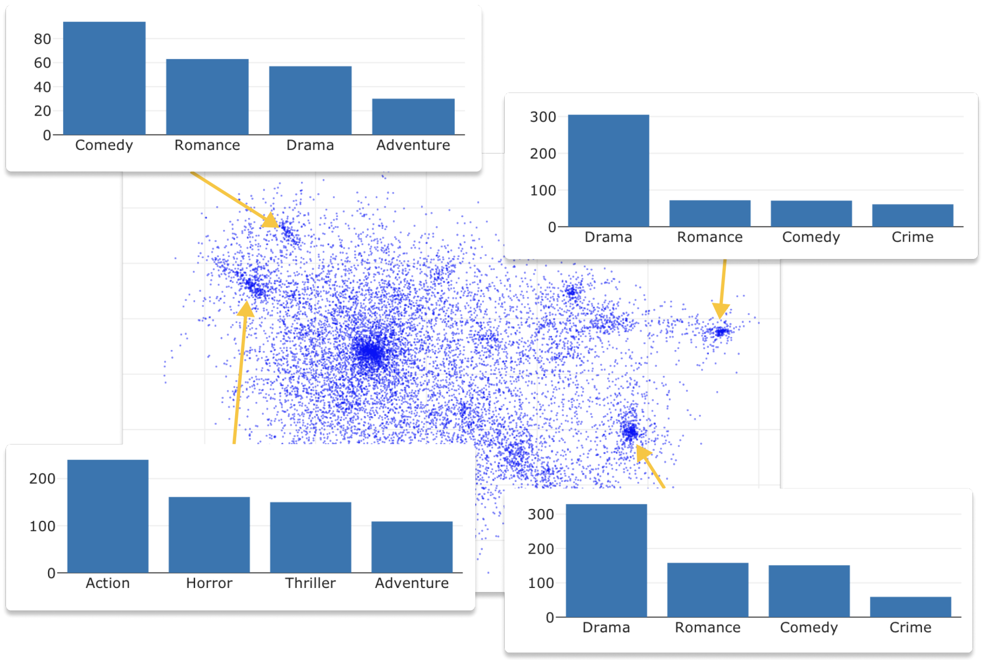
# Experiments results

## 4.1 Dataset exploration

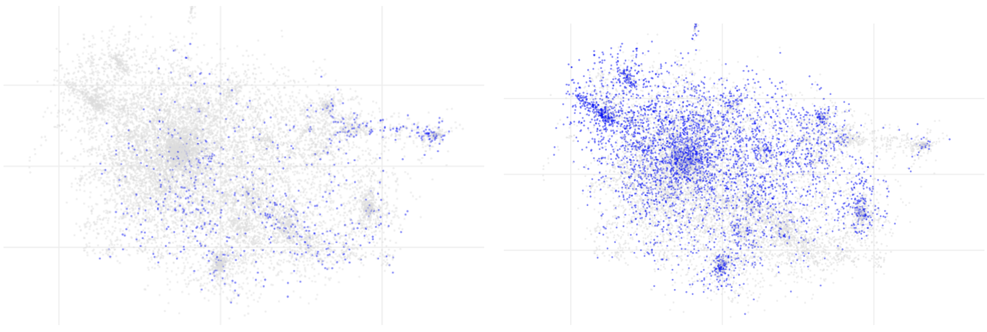
The first part of the experiments focuses on analyzing both datasets using visualization of embeddings obtained from the interaction matrix. Figure 4.1 shows the movie space within the ML dataset, where the distribution of genres is displayed for groups of items selected as part of separate clusters. The figure reveals that the location of movies in the latent space computed from the user interactions correlates with the genre attribute. Comedies and actions occur in the left part of the space, and in contrast, the right part of the space contains more dramas and romantic movies. This assumption can be verified by filtering only action movies highlighted in the upper left part of the space. The shooting year also corresponds with user interactions as movies from 1959 to 1969 are located in the right area of the space. The movies from 2000 to 2010 are spread across the space, but the higher concentration is in the areas of action and comedy movies, which were typically shot in later years.

Similarly, the analysis of the user space is performed. Figure 4.2 shows the distribution of genres for the 100 most interacted movies by this group of users for each cluster. The position of users corresponds to their preferences, as on the left and bottom side, users with a taste for adventure and action movies can be seen in the middle for dramas and at the top for comedies. When selecting only users with at least 20 interactions with adventure movies or movies from 2000 to 2010, these groups have a similar position in the space, as the adventures were shot in later years.

When analyzing the book space of the BX dataset, it is apparent that individual clusters were created based on the similarity of the interactions and authors of the books. As shown in Figure 4.3, an increased number of books by the same author can be observed in the selected clusters, which seems to be related to users reading more of their favorite author's literary works. As for the shape of the space, compared to the ML dataset, individual clusters of items are much better separated, and there are considerably more. After filtering out users with at least 20 interactions with books from 1989–1993 and 2000–2003, it can be observed that older works are located mainly in the upper and middle parts of the space, while newer works are in the peripheral parts mainly on the left and right. This result can be attributed to when the authors mainly published their books.

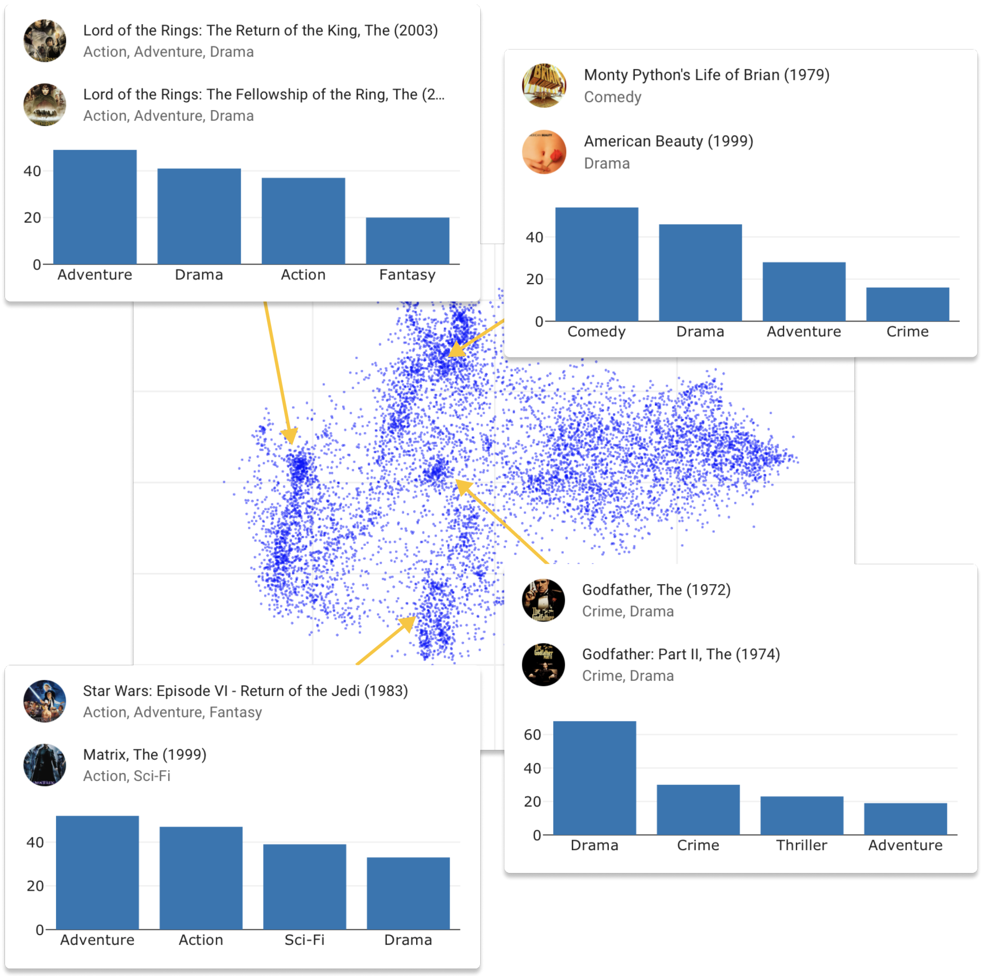


(a) Distribution of genres for selected clusters of movies.

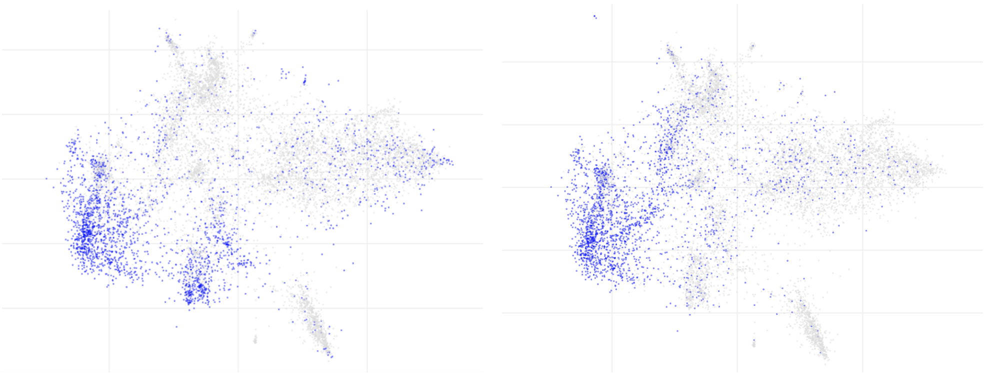


(b) Selection of movies from 1959–1969 (left) and movies from 2000–2010 (right).

Figure 4.1: Analysis of item embeddings (ML).

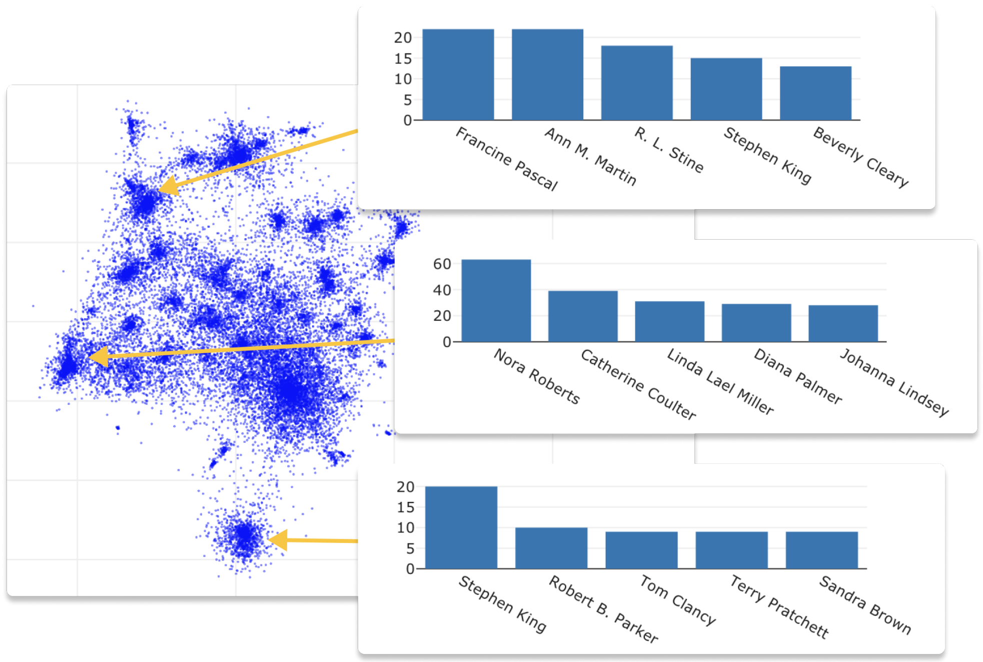


(a) Genres distribution of the most interacted movies for selected clusters of users.

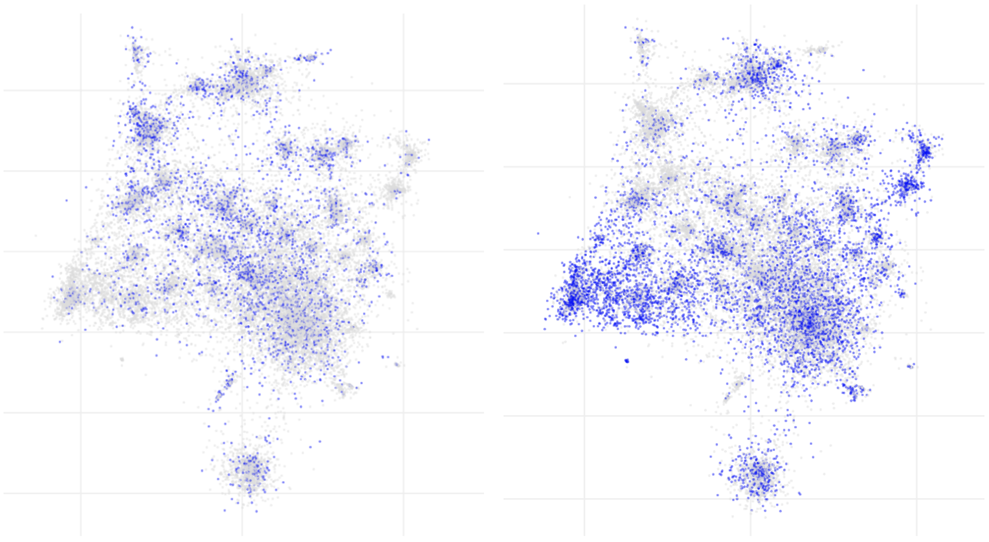


(b) Users rated at least 20 adventure movies (left) and movies from 2000–2010 (right).

Figure 4.2: Analysis of user embeddings (ML).



(a) Distribution of authors for selected clusters of books.



(b) Users rated at least 20 books from 1989–1993 (left) and 2000–2003 (right).

Figure 4.3: Analysis of item embeddings (BX).

## 4.2 Models evaluation

The results of evaluations of the implemented models are shown for both datasets in Figure 4.4. These are the aggregated values of metrics averaged across the whole set of validation users. The ML dataset shows very similar metric values for User-KNN and PureSVD models, which differ only in the proportion of long-tail items (APL@20 metric), where PureSVD has a higher value. The TopPop model also achieves decent prediction accuracy values, but it fails to cover the catalog of items (Coverage@20 metric) and the coverage of long-tail items (LCC@20 metric). As expected, the RAND has the worst accuracy results. However, it achieves 100% in diversity, novelty, or coverage, which indicates the need for a balance between accuracy and beyond-accuracy metrics. While all three other models show high popularity bias visible mainly on LCC@20, Coverage@20, and Novelty@20 metrics, the RAND model's predictions are spread over the catalog as they are not dependent on the popularity of items. The results also show how the ratio of long-tail items measured by the APL@20 metric alone is not enough to reveal the popularity of bias. As the LCC@20 metric shows, both User-KNN and PureSVD models repeatedly recommend the same set of less popular items.

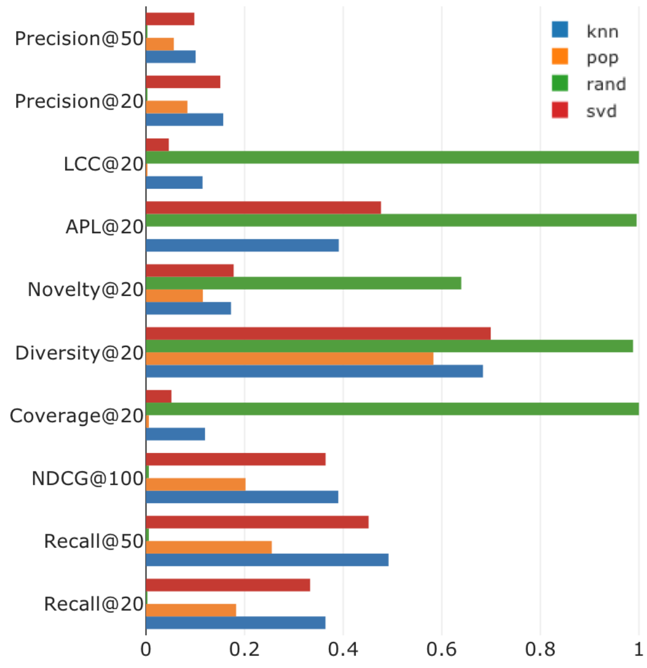
With the evaluation results for the BX dataset, it is interesting that the differences between the User-KNN, PureSVD, and TopPop models are significantly compared through all metrics. Regarding accuracy, the TopPop model is only slightly worse, which may be because the data characteristics are mainly helping to recommendations based on popular items. The significant popularity bias can be found in all models, except RAND, which again dominates in the diversity of predictions. Compared to the ML dataset, the RAND model has a worse coverage because there are many times fewer validation users through which it would be possible to cover the catalog.

Although some models may exhibit similar behavior based on aggregated metrics, it is impossible to determine whether they behave similarly to the same user groups. For this reason, it is also essential to examine the distribution of metrics, as shown in Figure 4.5.. From this, it can be seen that the User-KNN and PureSVD models are very similar, as their distributions have the same shape. When looking at the distributions of the Recall metric, the values 0, 0.5, and 1 are prominent, indicating a high imbalance in prediction accuracy. The Diversity@20 and Novelty@20 metrics distributions are then very pointed and concentrated around the mean, which is different from the APL@20 metric, where the proportion of long-tail items is widely dispersed. The TopPop model then shows only absolute values for the beyond-accuracy metrics, as it recommends the same items to all validation users.

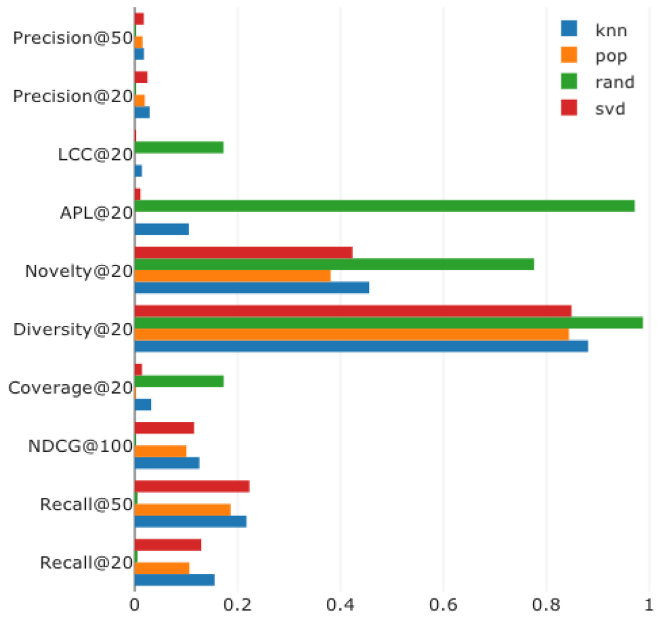
A visual comparison of metrics value distributions helps determine the correlation between them. Figure 4.6 shows projections of three metrics for the User-KNN model and the ML dataset. The first one references the accuracy of the predictions (Recall@50) and two other characteristics of the recommended movies (Diversity@20 and APL@20). At first glance, it is evident that the positions with the prevalence of well-predicted users correspond to the users who were recommended different items. However, this difference is not based on their attributes but their distinctive groups of interactions. So it is not so much about delivering different genres as it is about recommending movies through different user preferences. The APL@20 metric even more closely corresponds to the areas with high accuracy. It can be assumed that a higher proportion of long-tail items leads to a higher prediction success rate, as users want to discover less popular but exciting new movies. The specific locations of users with high beyond-accuracy metrics are also pointed out in Figure 4.6, where only users with a metric above a specific threshold are highlighted.

A closer analysis of the distribution of recall metric values for the UserKNN model in the space of validation users reveals a correlation between prediction accuracy and user preferences. Figure 4.8 shows 3 clusters, two at the top left and the middle bottom with a high metric value (around 0.8–1) and one at the top right with a significantly lower metric value (around 0.3–0.5). Although there is little difference between clusters in the distribution of genres for the most frequently interacted films, there is a visible difference in the distribution of the movie shooting year. A cluster with hard-to-predict users is characterized by a high proportion of movies from 1995 to 2005, while well-predicted users are fond of movies made before 1995.

Although it is not a specific feature of the framework, it can also be used to analyze hyperparameter tuning. As shown in Figure 4.9, the results of the evaluations for the PureSVD model change as the size of the latent space factor increases. When comparing two evaluations, one made with a factor of 30 and subsequently with a factor of 150, a significant decrease in accuracy (Recall@50 metric) can be observed. On the other hand, a significant increase in the ratio of long-tail items for each recommendation list (APL@20 metric). This shift is also visible in the distributions of these two metrics, where the captured evolution is with the increasing factor size. While the ratio of correctly predicted users decreases for the Recall@50 metric, there is a clear shift towards higher values for the APL@20 metric, which significantly changes the skew of the distribution. Based on this short analysis, it would be possible to find an appropriate balance between accuracy and the ratio of niche products so that the prediction ability would not be too damaged, but the user could discover less popular items.

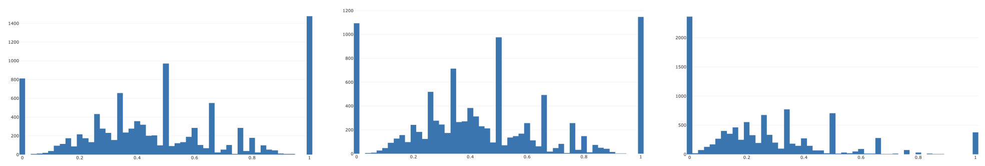


(a) Results for the ML dataset.

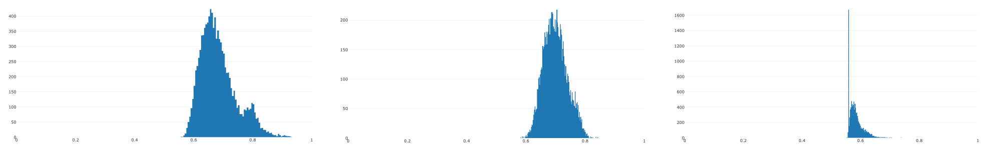


(b) Results for the BX dataset.

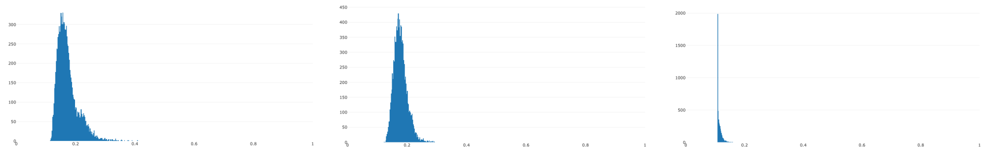
Figure 4.4: Metrics values averaged across the validation users.



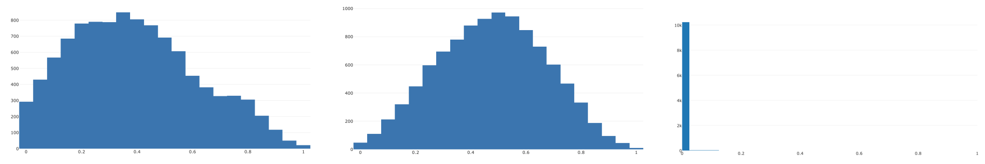
(a) Distributions of Recall@50



(b) Distributions of Diversity@20

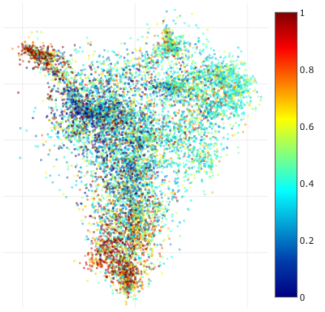


(c) Distributions of Novelty@20

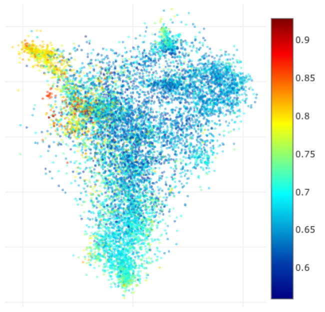


(d) Distributions of APL@20

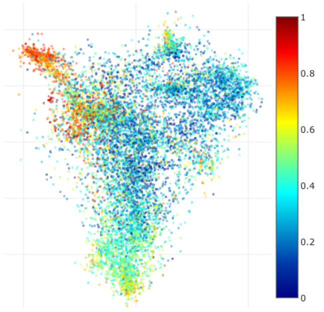
Figure 4.5: Distributions of metrics values for User-KNN (left), PureSVD (middle), and TopPop (right) models (ML).



(a) Recall@50

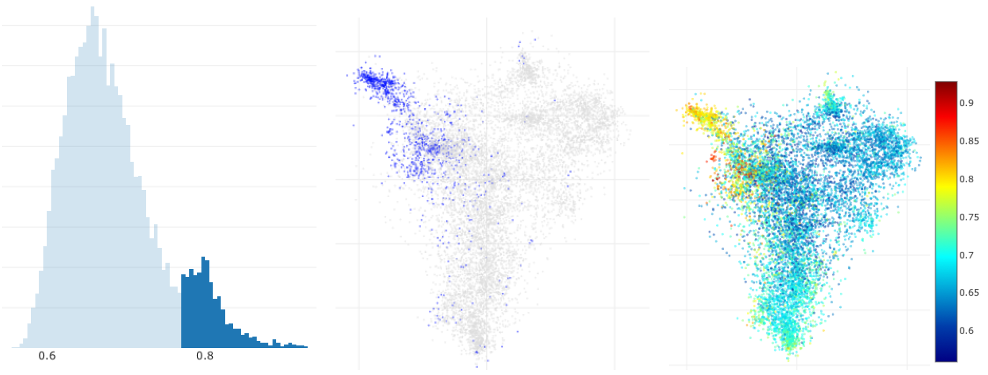


(b) Diversity@20

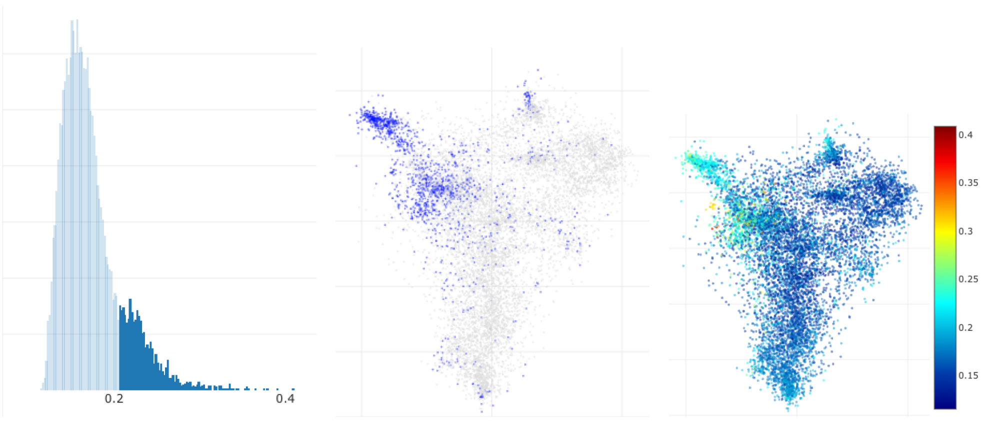


(c) APL@20

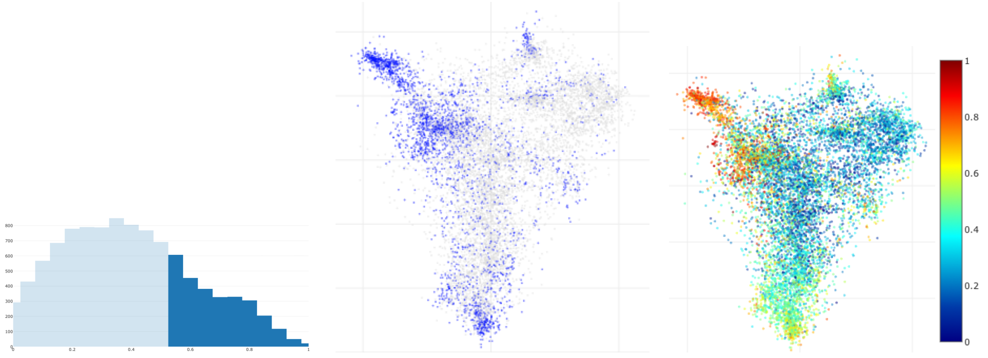
Figure 4.6: Projection of metrics values for User-KNN (ML).



(a) Distribution of Diversity@20 (values > 0.75)



(b) Distribution of Novelty@20 (values > 0.2)



(c) Distribution of APL@20 (values > 0.55)

Figure 4.7: Relation between the distribution of metrics values and position of users in 2D space for User-KNN (ML).

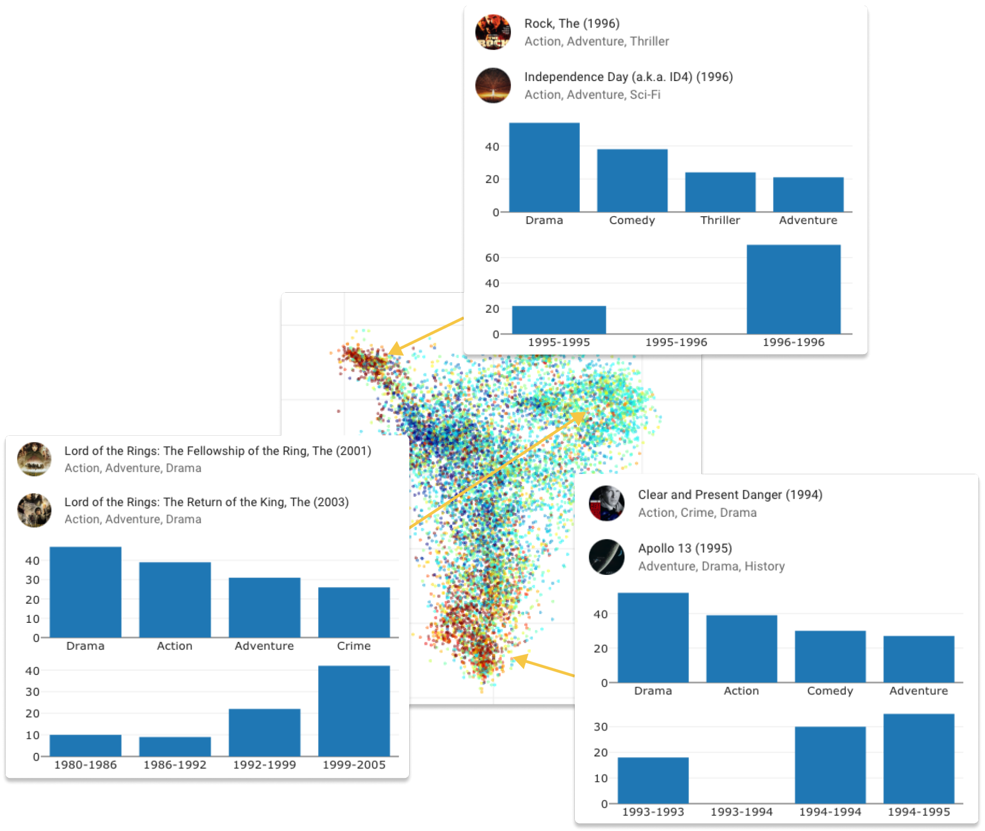
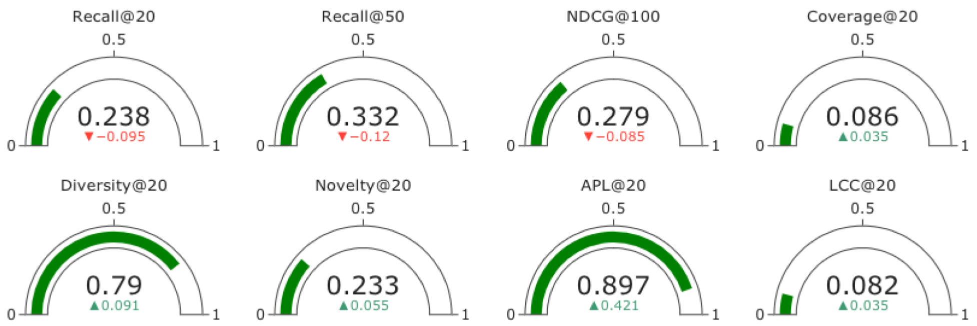


Figure 4.8: Clusters of users within the Recall@50 metric for User-KNN (ML).

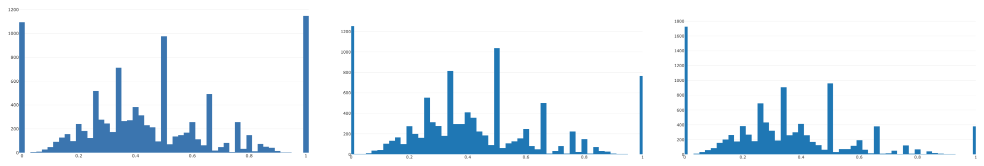
## 4.3 Prediction previews

The simulation environment is used to create recommendation preview dashboards for both datasets, each with three panels for one model, as figure 4.10 shows. Based on manually entered input interactions (right panel), the User-KNN (top row), PureSVD (middle row), and TopPop (bottom row) models return a list of the top four predictions.

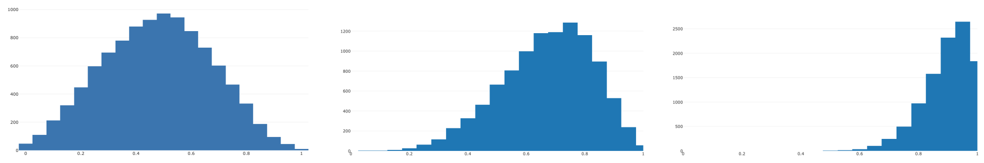
Seven action and adventure movies are selected for the ML dataset on the input. From the prediction previews, it can be observed that the User-KNN model does not provide very diverse recommendations, as 3 out of 4 places are taken by different Lord of the Rings episodes. On the other hand, PureSVD does not recommend additional episodes, and even it returns movies of similar genres, these are different movies giving the user more options. The example also shows that the TopPop model does not respond to input interactions, and it just recommends only the most popular movies. With the RAND model, which is not listed here, we would find random recommendations through the whole catalog, which might not fit into the user's preferences, but on the other hand, they could make him discover something unexpected and appealing.



(a) Shift of average metrics values between factor sizes $f = 30$ and $f = 150$.



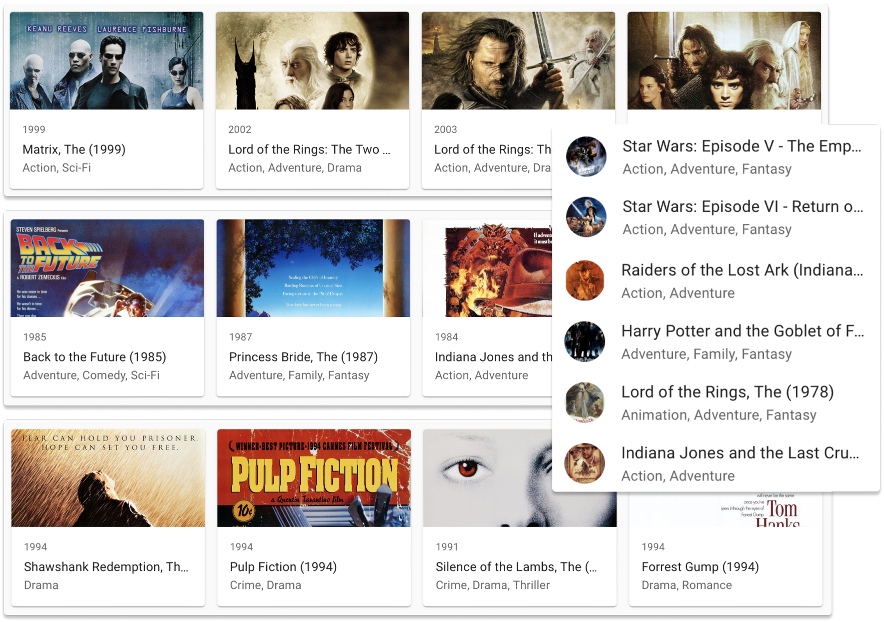
(b) Distributions of the Recall@50 metric.



(c) Distributions of the APL@20 metric.

Figure 4.9: Shift of metrics for different latent factor sizes $f = 30$ (left), $f = 70$ (middle), $f = 150$ (right) of the PureSVD model (ML).

The same problem with User-KNN as in the previous case can also be observed for the BX dataset. Although there is only one volume between the interactions, 3 out of 4 places are taken by different Harry Potter books, which does not give the user much choice at first glance. What is notable about these three models is a similarity with the TopPop model, as they agree on at least one prediction. This finding corresponds to the evaluation results, where all three models achieved comparable diversity, novelty, and accuracy.



1999
Matrix, The (1999)
Action, Sci-Fi

2002
Lord of the Rings: The Two ...
Action, Adventure, Drama

2003
Lord of the Rings: Th
Action, Adventure, Dra

1985
Back to the Future (1985)
Adventure, Comedy, Sci-Fi

1987
Princess Bride, The (1987)
Adventure, Family, Fantasy

1984
Indiana Jones and th
Action, Adventure

1994
Shawshank Redemption, Th...
Drama

1994
Pulp Fiction (1994)
Crime, Drama

1991
Silence of the Lambs, The (...
Crime, Drama, Thriller

1994
Forrest Gump (1994)
Drama, Romance

Star Wars: Episode V - The Emp...
Action, Adventure, Fantasy

Star Wars: Episode VI - Return o...
Action, Adventure, Fantasy

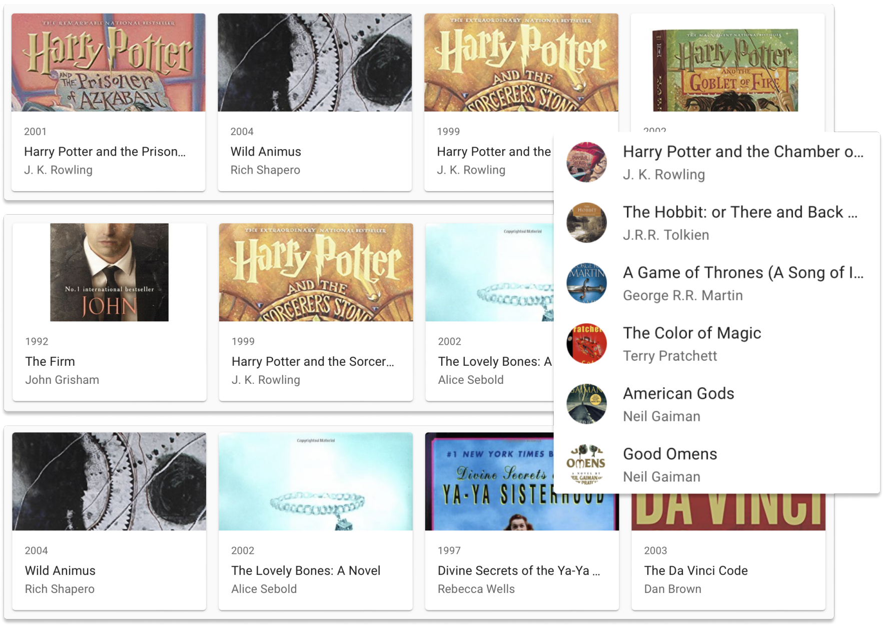
Raiders of the Lost Ark (Indiana...
Action, Adventure

Harry Potter and the Goblet of F...
Adventure, Family, Fantasy

Lord of the Rings, The (1978)
Animation, Adventure, Fantasy

Indiana Jones and the Last Cru...
Action, Adventure

(a) Movies recommendations for the ML dataset.



2001
Harry Potter and the Prison...
J. K. Rowling

2004
Wild Animus
Rich Shapero

1999
Harry Potter and the
J. K. Rowling

1992
The Firm
John Grisham

1999
Harry Potter and the Sorcer...
J. K. Rowling

2002
The Lovely Bones: A
Alice Sebold

2004
Wild Animus
Rich Shapero

2002
The Lovely Bones: A Novel
Alice Sebold

1997
Divine Secrets of the Ya-Ya ...
Rebecca Wells

2003
The Da Vinci Code
Dan Brown

Harry Potter and the Chamber o...
J. K. Rowling

The Hobbit: or There and Back ...
J.R.R. Tolkien

A Game of Thrones (A Song of I...
George R.R. Martin

The Color of Magic
Terry Pratchett

American Gods
Neil Gaiman

Good Omens
Neil Gaiman

(b) Books recommendations for the BX dataset.

Figure 4.10: Recommendation previews of User-KNN (top), PureSVD (middle), and TopPop (bottom) models based on the input interactions (right).

# Conclusion

In the thesis, recommendation approaches were first summarized, including types of biases, evaluation mechanisms, visualization techniques, and existing solutions of recommender frameworks. In the next part, the Repsys framework was introduced, which brings a new perspective on the analysis of recommender systems. Its methods for dataset processing, dataset visualization, and evaluation of the quality of the recommender model were described. Also, the interactive web application for the analysis of the results was presented. In the last part, a set of experiments on two datasets was performed, which verified the functionality and usability of the framework.

The work fulfills the assignment, as the designed and implemented framework allows the development and testing of recommender systems. The solution includes an SDK library to implement custom models and datasets. The set of dashboards for both recommendation previews and exploring the visualization of latent space of users and items is incorporated within the web application. Also, the dashboard for analyzing measured metrics is not missing, which, combined with correctly selected visualization tools, allows the detection of several types of biases.

The framework's source code was published under the GNU license and thus offers the involvement of the broader community in expanding functionalities. Plans for further expansion of the framework mainly include support of sequential models, the possibility of integrating user attributes, and the option of comparing the shapes of distributions between different evaluations. In addition, a paper based on this thesis will be submitted to the demo track of the RecSys 2022 conference.

# Bibliography

[1] Sarwar, B.; Karypis, G.; et al. Item-based collaborative filtering recommendation algorithms. *Proceedings of the tenth international conference on World Wide Web - WWW '01*, 2001: pp. 285–295, doi:10.1145/371920.372071. Available from: http://portal.acm.org/citation.cfm?doid=371920.372071

[2] Xie, X.; Tan, W.; et al. CuMF_SGD. *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, 2017-06-26: pp. 79–92, doi:10.1145/3078597.3078602. Available from: https://dl.acm.org/doi/10.1145/3078597.3078602

[3] Chaney, A. J. B.; Stewart, B. M.; et al. How algorithmic confounding in recommendation systems increases homogeneity and decreases utility. *Proceedings of the 12th ACM Conference on Recommender Systems*, 2018-09-27: pp. 224–232, doi:10.1145/3240323.3240370. Available from: https://dl.acm.org/doi/10.1145/3240323.3240370

[4] Abdollahpouri, H.; Burke, R.; et al. Popularity-Aware Item Weighting for Long-Tail Recommendation. *ArXiv preprint arXiv:1802.05382*, 2018.

[5] Beel, J.; Brunel, V. Data Pruning in Recommender Systems Research. *13th ACM Conference on Recommender Systems (RecSys)*, 2019.

[6] Meng, Z.; McCreadie, R.; et al. Exploring Data Splitting Strategies for the Evaluation of Recommendation Models. *Fourteenth ACM Conference on Recommender Systems*, 2020-09-22: pp. 681–686, doi:10.1145/3383313.3418479. Available from: https://dl.acm.org/doi/10.1145/3383313.3418479

[7] Agrawal, A.; Ali, A.; et al. Minimum-Distortion Embedding. *ArXiv preprint arXiv:2103.02559*, 2021.

[8] Veit, A.; Kovacs, B.; et al. Learning visual clothing style with heterogeneous dyadic co-occurrences. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4642–4650.

[9] Kordík, P. Data Mining Algorithms, 2020, soubor přístupný po přihlášení do sítě ČVUT – kopie souboru uložena na přiložené SD kartě. Available from: https://courses.fit.cvut.cz/NI-ADM/lectures/files/MI-ADM-10-en-handout.pdf

[10] Ekstrand, M. D.; Riedl, J. T.; et al. *Collaborative Filtering Recommender Systems*. Now Publishers, 2011, ISBN 9781601984425. Available from: https://books.google.cz/books?id=vVwu5B_1q-IC

[11] Desrosiers, C.; Karypis, G. A Comprehensive Survey of Neighborhood-based Recommendation Methods. *Recommender Systems Handbook*, 2011: pp. 107–144, doi:10.1007/978-0-387-85820-3_4. Available from: http://link.springer.com/10.1007/978-0-387-85820-3_4

[12] Jeckmans, A. J. P.; Beye, M.; et al. Privacy in Recommender Systems. *Social Media Retrieval*, 2013: pp. 263–281, doi:10.1007/978-1-4471-4555-4_12. Available from: http://link.springer.com/10.1007/978-1-4471-4555-4_12

[13] Steck, H. Evaluation of recommendations. *Proceedings of the 7th ACM conference on Recommender systems*, 2013-10-12: pp. 213–220, doi:10.1145/2507157.2507160. Available from: https://dl.acm.org/doi/10.1145/2507157.2507160

[14] Cañamares, R.; Castells, P.; et al. Offline evaluation options for recommender systems. *Information Retrieval Journal*, volume 23, no. 4, 2020: pp. 387–410, ISSN 1386-4564, doi:10.1007/s10791-020-09371-3. Available from: http://link.springer.com/10.1007/s10791-020-09371-3

[15] de Gemmis, M.; Lops, P.; et al. Recommender Systems, Basics of. *Encyclopedia of Social Network Analysis and Mining*, 2018: pp. 2125–2137, doi:10.1007/978-1-4939-7131-2_110158. Available from: http://link.springer.com/10.1007/978-1-4939-7131-2_110158

[16] Su, X.; Khoshgoftaar, T. M. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*, volume 2009, 2009-10-27: pp. 1–19, ISSN 1687-7470, doi:10.1155/2009/421425. Available from: https://www.hindawi.com/journals/aai/2009/421425/

[17] Balabanović, M.; Shoham, Y. Fab. *Communications of the ACM*, volume 40, no. 3, 1997: pp. 66–72, ISSN 0001-0782, doi:10.1145/245108.245124. Available from: https://dl.acm.org/doi/10.1145/245108.245124

[18] Lops, P.; de Gemmis, M.; et al. Content-based Recommender Systems. *Recommender Systems Handbook*, 2011: pp. 73–105, doi:10.1007/978-0-387-85820-3_3. Available from: http://link.springer.com/10.1007/978-0-387-85820-3_3

[19] Breese, J. S.; Heckerman, D.; et al. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, ISBN 155860555X, p. 43–52.

[20] Konstan, J. A.; Miller, B. N.; et al. GroupLens. *Communications of the ACM*, volume 40, no. 3, 1997: pp. 77–87, ISSN 0001-0782, doi:10.1145/245108.245126. Available from: https://dl.acm.org/doi/10.1145/245108.245126

[21] Linden, G.; Smith, B.; et al. Amazon.com recommendations. *IEEE Internet Computing*, volume 7, no. 1, 2003: pp. 76–80, ISSN 1089-7801, doi:10.1109/MIC.2003.1167344. Available from: http://ieeexplore.ieee.org/document/1167344/

[22] Ning, X.; Karypis, G. SLIM. *2011 IEEE 11th International Conference on Data Mining*, 2011: pp. 497–506, doi:10.1109/ICDM.2011.134. Available from: http://ieeexplore.ieee.org/document/6137254/

[23] Kordík, P. Data Mining Algorithms, 2020, soubor přístupný po přihlášení do sítě ČVUT – kopie souboru uložena na přiložené SD kartě. Available from: https://courses.fit.cvut.cz/NI-ADM/lectures/files/MI-ADM-11-en-handout.pdf

[24] Amatriain, X. Big & personal. *Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining Algorithms, Systems, Programming Models and Applications - BigMine '13*, 2013: pp. 1–6, doi:10.1145/2501221.2501222. Available from: http://dl.acm.org/citation.cfm?doid=2501221.2501222

[25] Koren, Y. Factorization meets the neighborhood. *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*, 2008: pp. 426–434, doi:10.1145/1401890.1401944. Available from: http://dl.acm.org/citation.cfm?doid=1401890.1401944

[26] Cremonesi, P.; Koren, Y.; et al. Performance of recommender algorithms on top-n recommendation tasks. *Proceedings of the fourth ACM conference on Recommender systems - RecSys '10*, 2010: pp. 39–46, doi:10.1145/1864708.1864721. Available from: http://portal.acm.org/citation.cfm?doid=1864708.1864721

[27] Zhang, S.; Yao, L.; et al. Deep Learning Based Recommender System. *ACM Computing Surveys*, volume 52, no. 1, 2020-01-31: pp. 1–38, ISSN 0360-0300, doi:10.1145/3285029. Available from: https://dl.acm.org/doi/10.1145/3285029

[28] Burke, R. Hybrid Recommender Systems. *User Modeling and User-Adapted Interaction*, volume 12, no. 4, 2002: pp. 331–370, ISSN 09241868, doi:10.1023/A:1021240730564. Available from: http://link.springer.com/10.1023/A:1021240730564

[29] Ludewig, M.; Jannach, D. Evaluation of session-based recommendation algorithms. *User Modeling and User-Adapted Interaction*, volume 28, no. 4-5, 2018: pp. 331–390, ISSN 0924-1868, doi:10.1007/s11257-018-9209-6. Available from: http://link.springer.com/10.1007/s11257-018-9209-6

[30] Quadrana, M.; Cremonesi, P.; et al. Sequence-Aware Recommender Systems. *ACM Computing Surveys*, volume 51, no. 4, 2019-07-31: pp. 1–36, ISSN 0360-0300, doi:10.1145/3190616. Available from: https://dl.acm.org/doi/10.1145/3190616

[31] Chen, J.; Dong, H.; et al. Bias and debias in recommender system. *ArXiv preprint arXiv:2010.03240*, 2020.

[32] Melchiorre, A. B.; Zangerle, E.; et al. Personality Bias of Music Recommendation Algorithms. *Fourteenth ACM Conference on Recommender Systems*, 2020-09-22: pp. 533–538, doi:10.1145/3383313.3412223. Available from: https://dl.acm.org/doi/10.1145/3383313.3412223

[33] Wang, N.; Chen, L. User Bias in Beyond-Accuracy Measurement of Recommendation Algorithms. *Fifteenth ACM Conference on Recommender Systems*, 2021-09-13: pp. 133–142, doi:10.1145/3460231.3474244. Available from: https://dl.acm.org/doi/10.1145/3460231.3474244

[34] D'Amour, A.; Srinivasan, H.; et al. Fairness is not static. *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 2020-01-27: pp. 525–534, doi:10.1145/3351095.3372878. Available from: https://dl.acm.org/doi/10.1145/3351095.3372878

[35] Mansoury, M.; Abdollahpouri, H.; et al. Feedback Loop and Bias Amplification in Recommender Systems. In *Proceedings of the 29th ACM international conference on information & knowledge management*, New York, NY, USA: ACM, 2020-10-19, ISBN 9781450368599, pp. 2145–2148, doi:10.1145/3340531.3412152. Available from: https://dl.acm.org/doi/10.1145/3340531.3412152

[36] Marlin, B. M.; Zemel, R. S. Collaborative prediction and ranking with non-random missing data. *Proceedings of the third ACM conference on Recommender systems - RecSys '09*, 2009: pp. 5–12, doi: 10.1145/1639714.1639717. Available from: http://portal.acm.org/citation.cfm?doid=1639714.1639717

[37] Collins, A.; Tkaczyk, D.; et al. A Study of Position Bias in Digital Library Recommender Systems. *ArXiv preprint arXiv:1802.06565*, 2018.

[38] Park, Y. J.; Tuzhilin, A. The long tail of recommender systems and how to leverage it. *Proceedings of the 2008 ACM conference on Recommender systems - RecSys '08*, 2008: pp. 11–18, doi: 10.1145/1454008.1454012. Available from: http://portal.acm.org/citation.cfm?doid=1454008.1454012

[39] Zhu, Z.; He, Y.; et al. Popularity Bias in Dynamic Recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, New York, NY, USA: ACM, 2021-08-14, ISBN 9781450383325, pp. 2439–2449, doi:10.1145/3447548.3467376. Available from: https://dl.acm.org/doi/10.1145/3447548.3467376

[40] Abdollahpouri, H.; Burke, R.; et al. Controlling Popularity Bias in Learning-to-Rank Recommendation. *Proceedings of the Eleventh ACM Conference on Recommender Systems*, 2017-08-27: pp. 42–46, doi: 10.1145/3109859.3109912. Available from: https://dl.acm.org/doi/10.1145/3109859.3109912

[41] Yin, H.; Cui, B.; et al. Challenging the long tail recommendation. *Proceedings of the VLDB Endowment*, volume 5, no. 9, 2012: pp. 896–907, ISSN 2150-8097, doi:10.14778/2311906.2311916. Available from: https://dl.acm.org/doi/10.14778/2311906.2311916

[42] Anderson, C. *The long tail*. Hachette UK, 2006.

[43] Brynjolfsson, E.; Hu, Y. J.; et al. From niches to riches. *Sloan management review*, 2006: pp. 67–71.

[44] Abdollahpouri, H.; Mansoury, M.; et al. The Unfairness of Popularity Bias in Recommendation. *RecSys Workshop on Recommendation in Multistakeholder Environments (RMSE)*, 2019. Available from: https://arxiv.org/pdf/1907.13286

[45] Friedman, B.; Nissenbaum, H. Bias in computer systems. *ACM Transactions on Information Systems*, volume 14, no. 3, 1996: pp. 330–347, ISSN 1046-8188, doi:10.1145/230538.230561. Available from: https://dl.acm.org/doi/10.1145/230538.230561

[46] Lin, K.; Sonboli, N.; et al. Crank up the volume. *ArXiv preprint arXiv:1909.06362*, 2019.

[47] Steck, H. Calibrated recommendations. *Proceedings of the 12th ACM Conference on Recommender Systems*, 2018-09-27: pp. 154–162, doi:10.1145/3240323.3240372. Available from: https://dl.acm.org/doi/10.1145/3240323.3240372

[48] Kaminskas, M.; Bridge, D. Diversity, Serendipity, Novelty, and Coverage. *ACM Transactions on Interactive Intelligent Systems*, volume 7, no. 1, 2017: pp. 1–42, ISSN 2160-6455, doi:10.1145/2926720. Available from: https://dl.acm.org/doi/10.1145/2926720

[49] Gunawardana, A.; Shani, G. A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. *The Journal of Machine Learning Research*, volume 10, no. 1, 2009: p. 2935–2962, ISSN 1532-4435. Available from: https://dl.acm.org/doi/10.1145/963770.963772

[50] Herlocker, J. L.; Konstan, J. A.; et al. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, volume 22, no. 1, 2004: pp. 5–53, ISSN 1046-8188, doi:10.1145/963770.963772. Available from: https://dl.acm.org/doi/10.1145/963770.963772

[51] Margaris, D.; Vassilakis, C. Improving collaborative filtering's rating prediction quality in dense datasets, by pruning old ratings. *2017 IEEE Symposium on Computers and Communications (ISCC)*, 2017: pp. 1168–1174, doi:10.1109/ISCC.2017.8024683. Available from: http://ieeexplore.ieee.org/document/8024683/

[52] Steck, H. Embarrassingly Shallow Autoencoders for Sparse Data. *The World Wide Web Conference on - WWW '19*, 2019: pp. 3251–3257, doi:10.1145/3308558.3313710. Available from: http://dl.acm.org/citation.cfm?doid=3308558.3313710

[53] Vančura, V.; Kordík, P. Deep Variational Autoencoder with Shallow Parallel Path for Top-N Recommendation (VASP). *Artificial Neural Networks and Machine Learning – ICANN 2021*, 2021: pp. 138–149, doi:10.1007/978-3-030-86383-8_11. Available from: https://link.springer.com/10.1007/978-3-030-86383-8_11

[54] Liang, D.; Krishnan, R. G.; et al. Variational Autoencoders for Collaborative Filtering. *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, 2018: pp. 689–698, doi:10.1145/3178876.3186150. Available from: http://dl.acm.org/citation.cfm?doid=3178876.3186150

[55] Little, R. J. A.; Rubin, D. B. *Statistical Analysis with Missing Data, Second Edition*. John Wiley & Sons, 2002, ISBN 9780471183860.

[56] Marlin, B. *Collaborative Filtering*. Dissertation thesis, University of Toronto, 2004.

[57] Refaeilzadeh, P.; Tang, L.; et al. Cross-Validation. *Encyclopedia of Database Systems*, 2009: pp. 532–538, doi:10.1007/978-0-387-39940-9_565. Available from: http://link.springer.com/10.1007/978-0-387-39940-9_565

[58] Kordík, P. Data Mining Algorithms, 2020, soubor přístupný po přihlášení do sítě ČVUT – kopie souboru uložena na přiložené SD kartě. Available from: https://courses.fit.cvut.cz/NI-ADM/lectures/files/MI-ADM-09-en-handout.pdf

[59] Järvelin, K.; Kekäläinen, J. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, volume 20, no. 4, 2002: pp. 422–446, ISSN 1046-8188, doi:10.1145/582415.582418. Available from: https://dl.acm.org/doi/10.1145/582415.582418

[60] Ge, M.; Delgado-Battenfeld, C.; et al. Beyond accuracy. *Proceedings of the fourth ACM conference on Recommender systems - RecSys '10*, 2010: pp. 257–260, doi:10.1145/1864708.1864761. Available from: http://portal.acm.org/citation.cfm?doid=1864708.1864761

[61] McNee, S. M.; Riedl, J.; et al. Being accurate is not enough. *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, 2006-04-21: pp. 1097–1101, doi:10.1145/1125451.1125659. Available from: https://dl.acm.org/doi/10.1145/1125451.1125659

[62] Hurley, N.; Zhang, M. Novelty and Diversity in Top-N Recommendation – Analysis and Evaluation. *ACM Transactions on Internet Technology*, volume 10, no. 4, 2011: pp. 1–30, ISSN 1533-5399, doi:10.1145/1944339.1944341. Available from: https://dl.acm.org/doi/10.1145/1944339.1944341

[63] Castells, P.; Hurley, N. J.; et al. Novelty and Diversity in Recommender Systems. *Recommender Systems Handbook*, 2015: pp. 881–918, doi:10.1007/978-1-4899-7637-6_26. Available from: http://link.springer.com/10.1007/978-1-4899-7637-6_26

[64] Smyth, B.; McClave, P. Similarity vs. Diversity. *Case-Based Reasoning Research and Development*, 2001-7-12: pp. 347–361, doi:10.1007/3-540-44593-5_25. Available from: http://link.springer.com/10.1007/3-540-44593-5_25

[65] Vargas, S.; Castells, P. Rank and relevance in novelty and diversity metrics for recommender systems. *Proceedings of the fifth ACM conference on Recommender systems - RecSys '11*, 2011: pp. 109–116, doi:10.1145/2043932.2043955. Available from: http://dl.acm.org/citation.cfm?doid=2043932.2043955

[66] Ekstrand, M. D.; Carterette, B.; et al. Evaluating Recommenders with Distributions. *RecSys Workshop on Perspectives on Evaluation*, 2021.

[67] Said, A.; Bellogín, A. Comparative recommender system evaluation. *Proceedings of the 8th ACM Conference on Recommender systems - RecSys '14*, 2014: pp. 129–136, doi:10.1145/2645710.2645746. Available from: http://dl.acm.org/citation.cfm?doid=2645710.2645746

[68] Wang, Y.; Huang, H.; et al. Understanding How Dimension Reduction Tools Work. *ArXiv preprint arXiv:2012.04456*, 2020.

[69] Pearson, K. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 1901: pp. 559–572.

[70] Torgerson, W. S. Multidimensional scaling: I. Theory and method. *Psychometrika*, 1952: pp. 401–419.

[71] van der Maaten, L.; Hinton, G. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, volume 9, no. 86, 2008: pp. 2579–2605.

[72] McInnes, L.; Healy, J.; et al. UMAP. *ArXiv preprint arXiv:1802.03426*, 2020.

[73] He, R.; McAuley, J. Ups and Downs. *Proceedings of the 25th International Conference on World Wide Web*, 2016-04-11: pp. 507–517, doi:10.1145/2872427.2883037. Available from: https://dl.acm.org/doi/10.1145/2872427.2883037

[74] He, R.; Fang, C.; et al. Vista. *Proceedings of the 10th ACM Conference on Recommender Systems*, 2016-09-07: pp. 309–316, doi:10.1145/2959100.2959152. Available from: https://dl.acm.org/doi/10.1145/2959100.2959152

[75] Shen, J.; Wang, R.; et al. Visual exploration of latent space for traditional Chinese music. *Visual Informatics*, volume 4, no. 2, 2020: pp. 99–108, ISSN 2468502X, doi:10.1016/j.visinf.2020.04.003. Available from: https://linkinghub.elsevier.com/retrieve/pii/S2468502X20300152

[76] Gil, S.; Bobadilla, J.; et al. VisualRS. *Knowledge-Based Systems*, volume 155, 2018: pp. 66–70, ISSN 09507051, doi:10.1016/j.knosys.2018.04.028. Available from: https://linkinghub.elsevier.com/retrieve/pii/S095070511830193X

[77] Tintarev, N.; Masthoff, J. A Survey of Explanations in Recommender Systems. *2007 IEEE 23rd International Conference on Data Engineering Workshop*, 2007: pp. 801–810, doi:10.1109/ICDEW.2007.4401070. Available from: http://ieeexplore.ieee.org/document/4401070/

[78] Bostandjiev, S.; O'Donovan, J.; et al. TasteWeights. *Proceedings of the sixth ACM conference on Recommender systems - RecSys '12*, 2012: pp. 35–42, doi:10.1145/2365952.2365964. Available from: http://dl.acm.org/citation.cfm?doid=2365952.2365964

[79] O'Donovan, J.; Smyth, B.; et al. PeerChooser. *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*, 2008: pp. 1085–1088, doi:10.1145/1357054.1357222. Available from: http://portal.acm.org/citation.cfm?doid=1357054.1357222

[80] Ekstrand, M. D.; Ludwig, M.; et al. Rethinking the recommender research ecosystem. *Proceedings of the fifth ACM conference on Recommender systems - RecSys '11*, 2011: pp. 133–140, doi:10.1145/2043932.2043958. Available from: http://dl.acm.org/citation.cfm?doid=2043932.2043958

[81] Saúl, V. *Novelty and diversity evaluation and enhancement in recommender systems*. Phd thesis, Universidad Autónoma de Madrid, Spain, Madrid, 2015.

[82] Guo, G.; Zhang, J.; et al. LibRec. In *UMAP Workshops*, volume 4, Citeseer, 2015.

[83] Gupta, U.; Hsia, S.; et al. DeepRecSys. *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020: pp. 982–995, doi:10.1109/ISCA45697.2020.00084. Available from: https://ieeexplore.ieee.org/document/9138960/

[84] Sun, Z.; Yu, D.; et al. Are We Evaluating Rigorously? Benchmarking Recommendation for Reproducible Evaluation and Fair Comparison. *Fourteenth ACM Conference on Recommender Systems*, 2020-09-22: pp. 23–32, doi:10.1145/3383313.3412489. Available from: https://dl.acm.org/doi/10.1145/3383313.3412489

[85] Anelli, V. W.; Bellogin, A.; et al. Elliot. *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021-07-11: pp. 2405–2414, doi:10.1145/

3404835.3463245. Available from: https://dl.acm.org/doi/10.1145/3404835.3463245

[86] Sonboli, N.; Mansoury, M.; et al. Librec-auto. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, New York, NY, USA: ACM, 2021-10-26, ISBN 9781450384469, pp. 4584–4593, doi:10.1145/3459637.3482006. Available from: https://dl.acm.org/doi/10.1145/3459637.3482006

[87] Coba, L.; Confalonieri, R.; et al. RecoXplainer. *IEEE Computational Intelligence Magazine*, volume 17, no. 1, 2022: pp. 46–58, ISSN 1556-603X, doi:10.1109/MCI.2021.3129958. Available from: https://ieeexplore.ieee.org/document/9679765/

[88] Harper, F. M.; Konstan, J. A. The MovieLens Datasets. *ACM Transactions on Interactive Intelligent Systems*, volume 5, no. 4, 2016-01-07: pp. 1–19, ISSN 2160-6455, doi:10.1145/2827872. Available from: https://dl.acm.org/doi/10.1145/2827872

[89] Scikit-learn: Machine Learning in Python. c2022. Available from: https://scikit-learn.org

[90] McInnes, L. Umap-learn: Uniform Manifold Approximation and Projection for Dimension Reduction. c2018. Available from: https://umap-learn.readthedocs.io

[91] PyMDE: Minimum-Distortion Embedding. c2021. Available from: https://pymde.org

[92] Ziegler, C.-N.; McNee, S. M.; et al. Improving recommendation lists through topic diversification. *Proceedings of the 14th international conference on World Wide Web - WWW '05*, 2005, doi:10.1145/1060745.1060754. Available from: http://portal.acm.org/citation.cfm?doid=1060745.1060754

[93] Cheng, Y.; Yin, L.; et al. LorSLIM. *2014 IEEE International Conference on Data Mining*, 2014: pp. 90–99, doi:10.1109/ICDM.2014.112. Available from: http://ieeexplore.ieee.org/document/7023326/

[94] Pandas: powerful Python data analysis toolkit. c2022. Available from: https://pandas.pydata.org

[95] SciPy: Fundamental algorithms for scientific computing in Python. c2022. Available from: https://scipy.org

[96] Sanic: Build fast. Run fast. c2021. Available from: https://sanic.readthedocs.io

[97] Click: Composable command line interface toolkit. c2014. Available from: https://click.palletsprojects.com

[98] React: A JavaScript library for building user interfaces. c2022. Available from: https://reactjs.org

[99] MUI: The React UI library you always wanted. c2022. Available from: https://mui.com

[100] React Redux: Official React bindings for Redux. c2015. Available from: https://react-redux.js.org

[101] Plotly JavaScript Open Source Graphing Library. c2021. Available from: https://plotly.com/javascript

Appendix A

# Acronyms

| | |
|---|---|
| **RS** | Recommender System |
| **CF** | Collaborative Filtering |
| **DR** | Dimension Reduction |
| **IR** | Information Retrieval |
| **ML** | MovieLens |
| **BX** | Book-Crossing |
| **MF** | Matrix Factorization |
| **SDK** | Software Development Kit |

Appendix **B**

# Contents of enclosed CD

```
readme.txt ....................... the file with CD contents description
└─ DP_Safarik_Jan_2022.pdf ................ the thesis text in PDF format
└─ repsys .................................. the directory of source codes
└─ thesis ................. the directory of LaTeX source codes of the thesis
└─ experiments ........... the directory of source codes of the experiments
└─ bibliography ............. the directory of copies of referenced lectures
   └─ MI-ADM-09-en-handout.pdf
   └─ MI-ADM-10-en-handout.pdf
   └─ MI-ADM-11-en-handout.pdf
```

Appendix C

# Implementation details

## C.1 Architecture overview

The framework's core is responsible for loading models and dataset instances through the SDK alongside the configuration file overriding the default settings for the current project. The core functionality includes dataset pruning and splitting, models training and evaluation, and data visualization while connecting all the main components. Most of the features are available from the command-line utility or the REST API interface, which provides access for the web application to fetch the necessary data, as shown in Figure C.1.

Both framework and the SDK are written in Python because of the possible usage inside the Jupyter notebooks and the wide range of data-oriented and scientific libraries, while the core functionality is built around three of them: Pandas [94], SciPy [95], and scikit-learn [89]. The Sanic [96] library is used for the REST API interface implementation because of its small size and high performance. The command-line utility is created using the Click [97] library, which simplifies handling various types of commands. Many other tools and libraries are used for the development, but the above are the main ones.

The web application is implemented as a stand-alone JavaScript project using the React [98] library, and the Material UI [99] library as it offers a broad set of pre-built components. It communicates with the framework using the REST API interface, which it connects to after loading the page. This way, it gets information about implemented models, evaluation results, a list of users and their interactions, a catalog of items, and much more. The React Redux [100] library and its RTK Query tool have been used to quickly implement the API interface and manage the application's state. Configuration of the recommendations preview dashboard and mapping of item attributes to the view component are stored in the browser memory, where they remain after closing the browser until the next application launch. All visualizations like scatter plots, bar charts, or histograms are implemented with the Plotly.js [101] library, able to render a large amount of data thanks to WebGL support.

## C.2 Server endpoints

In order to maximize the modularity of the framework and separate the presentation layer from the business logic layer, the REST API was created for communication between the core functionality and web application. After the server service is launched, several API endpoints are available, the basic overview of which is described in the following list.

**GET /dataset** returns information about the dataset with a list of item attributes. For categories or tags, it returns all unique values found in the catalog, and for the numeric type, it returns a list of bins computed from the distribution of the values.

**GET /models** returns a list of implemented models.

**GET /models/metrics** returns a summary of each model's current and previous evaluation results.

**POST /models/[model]/predict** receives the model's name and user's ID and returns a list of the top items recommended by the model. Instead of the user's ID, a list of interacted items can be provided on the input.

**GET /models/[model]/metrics/[users, items]** returns complete evaluation results of each metric for the given model across the whole distribution of users or items.

**GET /users** returns a list of user IDs from the specified dataset split.

**GET /users/[id]** returns the interaction history for a given user.

**GET /items** returns all items from the catalog filtered by an input query, while the search is done against the item's attribute marked as a title.

**POST /[users, items]/search** receives an attribute name with a value, and it returns a list of items that match the query or a list of users that interacted with items matching the query. A minimum number of interactions with such items must be provided in the case of users.

**POST /[users, items]/describe** receives a list of users, and it returns the top items they interacted with along the distribution of attribute values for these items. If a list of items is provided, it directly returns the distribution of attribute values.

**GET /[users, items]/embeddings** returns an array of computed 2D embeddings for each user or item.

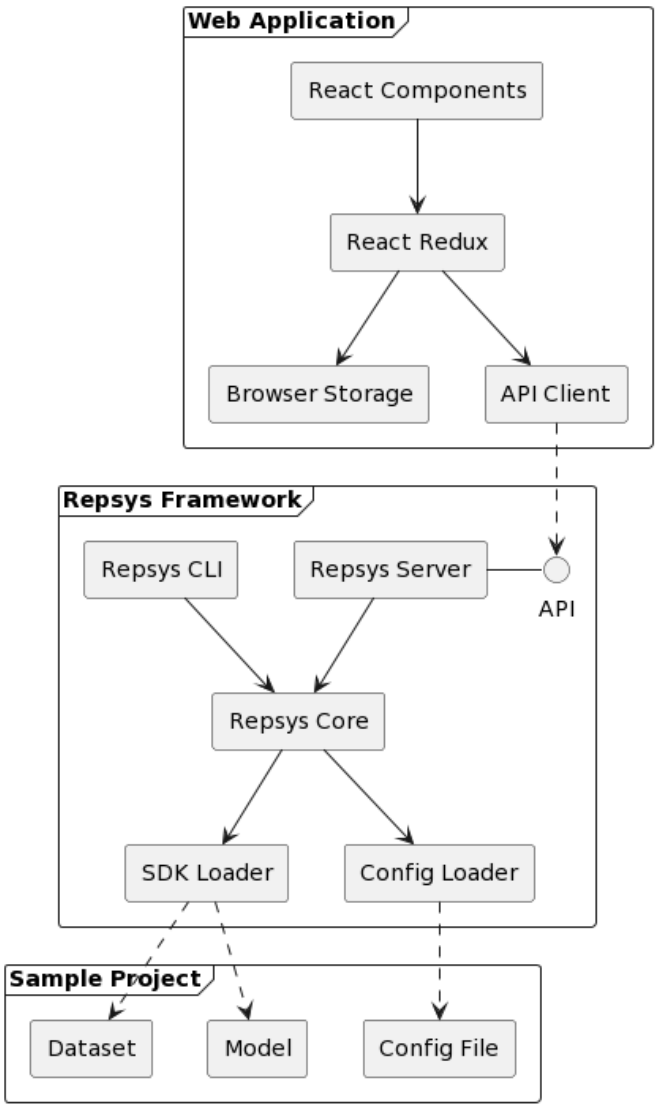


Figure C.1: Overview of the Repsys framework architecture.