VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
SOFTWARE ENGINEERING STUDY PROGRAM

Deep Neural Network-based Personalized Book Recommender System Feasibility Study for LIBIS

Giliaisiais neuroniniais tinklais grįstos suasmenintos knygų rekomendacijų sistemos įgyvendinamumo tyrimas LIBIS sistemoje

Master Thesis

Author:	Kazimieras Senvaitis	(signature)
Supervisor:	Assist., Dr. Vytautas Valaitis	(signature)
Reviewer:	Assist., Dr. Linas Petkevičius	(signature)

Vilnius – 2022

Santrauka

Rekomendacinės sistemos (RSs) tampa neatskiriama kasdienio gyvenimo dalimi. Didžiuliuose informacijos kloduose jos padeda rasti mėgstamus daiktus pirkimui, draugus socialiniuose tinkluose ar mėgstamiausius filmus žiūrėjimui. Tačiau rekomendacinės sistemos viešųjų bibliotekų dalykinėje srityje tyrinėtos menkai, nors bendrai RS mokslo šaka šiuo metu klesti dėl pažangos dirbtiniame intelekte ir versle kuriamos aukštos vertės. Šis magistro baigiamasis darbas papildo šią tyrimų šaką išsamiu tyrimu apie giliaisiais neuroniniais tinklais grįstos suasmenintos knygų rekomendacijos sistemos įgyvendinamumą LIBIS sistemoje, užtikrinant privatumo išsaugojimą. Šiame darbe pristatomi LIBIS sistemoje esantys duomenys ir jų ribotumas, paruoštas 20 milijonų knygų užsakymų duomenų rinkinys. Keletas industrijoje lyderiaujančių bendradarbiavimo filtravimo (angl. Collaborative filtering) modelių įvertinami naudojant minėtąjį duomenų rinkinį. Toliau, pristatomas hibridinis modelis sukombinuojant geriausiai pasirodžiusį bendradarbiavimo filtravimo modelį su demografinio filtravimo ir turiniu grįsto filtravimo modeliais, rezultate sukuriantis patikimą rekomendacinę sistemą, sugebančią teikti rekomendacijas esant tiek naujo vartotojo šalto starto tiek naujos knygos šalto starto problemas, tuo tarpu skaitymo istoriją sukaupusiems vartotojams teikia įvairialypes, iki tol nematytas rekomendacijas su laimingo atsitiktinumo galimybėmis. Taip pat pateikiamas išsamus integracijos su LIBIS sistema variantas.

Raktiniai žodžiai: rekomendacinės sistemos, viešosios bibliotekos, LIBIS, METIS, gilieji neuroniniai tinklai, bendradarbiavimo filtravimas, demografinis filtravimas, turiniu grįstas filtravimas, hibridinis modelis

Summary

Recommender systems (RSs) are becoming an inseparable part of our everyday lives. In vast volumes of data, they help us find our favorite items to purchase, our friends on social networks, and our favorite movies to watch. However, in the domain of public libraries, RSs are yet poorly covered even though RS science field is currently flourishing due to the advancements in machine learning and high business value. This master thesis bridges this gap by offering in-depth feasibility study of implementing deep neural network-based personalized book recommender system for Lithuanian public library software LIBIS with emphasis on privacy preservation. In this paper, LIBIS data and limitations regarding book orders are presented and a dataset of 20 million book orders has been developed. Multiple industry-leading collaborative filtering (CF) algorithms have been evaluated on the aforementioned dataset. Furthermore, a hybrid model has been developed by combining best-performing variant of CF with Demographics filtering model and Content-based filtering model into a robust recommender system, which is capable of resolving new user cold-start and new item cold-start problems, while for users with long reading history provides diverse, novel, serendipitous recommendations. Lastly, integration of the RS into LIBIS system is presented.

Keywords: recommender system, public library, LIBIS, METIS, deep neural networks, collaborative filtering, demographics filtering, content-based filtering, hybrid model

CONTENTS

Introduction

Research Problem Area

Problematic in domain

According to International Publishers Association, a decrease of traditional book reading has been a global tendency for the last 20 years. Particularly worrying is the fact that the decrease is most prevalent among young people. However, the report also shows that the number of people who do not read books has decreased. In addition, several countries exhibit an increase in leisure reading [Int20].

Lithuanian Integral Information System of Libraries - LIBIS is a software that runs across all public libraries in Lithuania. LIBIS is centered around Universal Machine Readable Catalogue (UNIMARC) standard, as in most European countries [Lie]. LIBIS serves the following main purposes:

- Storing and organizing various publications in form of bibliographic records that store key information about the publication;
- Storing and organizing various authorities records. These records connect various variants of entities, e.g. people, families, geographical and other titles. In turn, authorities records allow joining multiple bibliographic records, thus allowing advanced reference searches and identification;
- Storing and organizing exemplar information (exemplar is an actual physical publication) with relation to its bibliographic record;
- Information exchange with public libraries of other countries;
- Providing library services to readers;
- Obtaining various library performance reports.

From mid-2019, Lithuanian National Library (further LNB) with funding from European Union has invested nearly 2 million Euros into the modernization of LIBIS [Cen19]. The new system is the sixth major rework, thus called LIBISv6. Historically, the last ultimate rework is LIBISv4, which development started in the 1990s, took about 5 years to develop, and is planned to be actively maintained until 2021. Whereas LIBISv5 is an introduction of ibiblioteka.lt module. Unlike LIBISv5, LIBISv6 is a ground-up rework, which is expected not to mimic LIBISv4 but to greatly excel it. This modernization project could be used to fight back declining reading in Lithuania.

However, attracting people to book media is not an easy task. As more and more media is battling for user's time, it is unlikely that this pressure will change soon [Int20]. These media platforms are using various techniques for viewer capture. Although nearly all the largest media platforms use one trick: personalized content recommendation systems. Recommendations add another dimension to the user experience. Recommender systems are the major part of the largest internet content platforms, such as YouTube and Netflix websites [KBV09], with 2.3 billion active monthly users on YouTube [Mar21] and 203.66 million active subscriptions on Netflix [Bri21]. For instance, 80 percent of watched movies on Netflix came from recommendations and 60 percent of Youtube

video clicks were recommended from the home page [ZYS+19]. They provide recommendations that are personalized, vastly scalable, dynamic, and implement techniques that greatly capture any audience's attention [CAS16]. Yet LIBISv6 does not offer any of that. Although, the previous version - LIBISv4 - had a similar books proposal which was neither personalized nor flexible.

With millions of bibliographic records LIBIS also experiencing information overload, where no librarian can effectively and accurately recommend items to a reader. Recommender systems not only attract and retain users but is also an effective strategy to overcome ever-growing volumes of information [ZYS+19].

Available solutions

In recent years, deep learning has generated vast interest in many research fields. Deep learning has driven a remarkable revolution in industrial recommender applications showing a significant improvement over traditional non-ML models. The field of deep learning in recommender systems is flourishing [DSD17; ZYS+19].

There are mainly two approaches in the recommendation systems (further: RS): personalized and non-personalized. Each approach has different use cases and machine learning techniques [SGB16]. This paper focuses only on personalized RS, where the main algorithm categories are: 1) Knowledge-based filtering 2) Demographics-based filtering 3) Content-based filtering 4) Collaborative filtering 5) Hybrid.

Knowledge-Based recommender systems are based on explicitly specified user requirements, thus are particularly useful when sufficient ratings may not be available for the recommendation process. Such domains include real estate, automobiles, or luxury items. Constraints are collected by using questionnaires or submitting requirements on a form [Agg16].

Content-based filtering techniques are built on customer behavior and item descriptions. Usually, customer preferences were collected in accordance to items the user enjoyed in the past or is currently looking at in present. In this technique other users do not have any effect on recommendations [Agg16; DSD17; MB19].

Oh et al. [OLL+14] developed a content-based filtering model based on a deep neural network to provide personalized news recommendations. They state that as news becomes old news very soon, collaborative filtering is not applicable in their domain because it requires a significant amount of time for collecting information on which news is popular among similar users.

Collaborative filtering was popularized back in 2009 by Koren et al. [KBV09]. The idea behind it was that the user will probably like items that users with similar viewing habits like. They developed an ML-based matrix factorization technique winning Netflix Prize competition for movie recommendations, which exceeded previously state-of-the-art nearest-neighbor techniques. Deep learning is able to infer latent user and item features in collaborative filtering which allows getting rid of manual feature engineering of both user and item features.

Generally, in the collaborative filtering approach, deep neural networks are used to extract latent features which are used to relate to similar users. Without ML, features would be developed by feature engineers, which is both cost-time ineffective and inflexible.

Collaborative filtering was also used for Youtube content recommendations [CAS16]. This

recommender system is considered one of the largest scale and most sophisticated recommendation systems ever built. They use a classic two-stage approach implemented with deep neural networks: a deep candidate generation and a deep ranking model. Their variation of collaborative filtering was able to outperform previously used matrix factorization approaches.

Hybrid recommender systems join different recommender systems to mitigate the demerits of each. For example, collaborative filtering by its definition fails on newly added items, because they do not have ratings. However, combining it with content-based filtering allows recommending such new items based on features item or user exhibits [DSD17].

Gjoreski [Gjo13] presented a hybrid recommender for point of interest recommendation for trip planning. First, the content-based filtering algorithm filters out the points of interest according to user preferences. Then, attractiveness is determined using a combination of knowledge-based and collaborative filtering-based approaches.

Soediono [Soe15] presented a student-centered hybrid recommender system to provide relevant learning objects. While combining content-based, collaborative, and knowledge-based techniques, this recommender system aims at generating knowledge, skills, attitudes and competencies according to the student profile.

Problematic in solutions

Different techniques introduce various trade-offs [DSD17]. For example, knowledge-based RS is unable to recommend content that users may like but they do not query for, thus lacks properties of audience retention. Content-based filtering may accurately infer unique user taste but fails at recommending content to new users, also overspecializes. Collaborative filtering solves the overspecialization problem but introduces the first rater problem and popularity bias, which results in click-bait results. Also, it suffers from a cold start problem. At the cost of increased complexity, the hybrid approach provides a variety of hybridization methods solving the aforementioned issues [SGB16].

Performance measures of recommendation algorithms are also content dependant. For instance, content-based systems are designed mostly to recommend text-based items, so keywords are used as content here, thus is not as applicable on video-based content recommendations.

To date, no articles were found on UNIMARC-based public library content recommendations. This domain inherits problems:

1. Cold-start - globally book consumption trends are low and decreasing, many readers barely read a single book each year [Int20];
2. Lack of training data - due to low book consumption in general [Int20];
3. Absence of feedback information - there are no tools for users to give feedback whether they liked a book or not;
4. Inferred feedback accuracy - feedback could be inferred from how long the reader had a book, however, this may prove to be inaccurate;
5. Recommended items subset selection - only a small subset of all bibliographic records represent books. Whereas the majority of bibliographic records describe articles in magazines or newspapers;

6. RS model training on personal data - without attention to personal data processing algorithmic decisions may be discriminatory and reproducing human biases, thus violating GDPR. In addition, improper data preparation introduces a risk of data subject re-identification [Bie19].

Research Object

The research object is recommendation system implementation in LIBIS, including:

1. UNIMARC data structure in relation to recommendation system;
2. Collected user data and its limitations in LIBIS;
3. Machine learning-based recommendation system algorithms;
4. Recommender system integration into LIBIS system;
5. GDPR conformation analysis;

The research object is **not**:

1. Development of new machine learning techniques or algorithms;
2. Non-personalized recommendation systems;

Research Goal and Tasks

The **research goal**: identify suitable techniques and suggest implementation of recommendation system for LIBIS.

In order to reach the goal, the following **tasks** are raised:

1. Analyze data and limitations in LIBIS;
2. Analyze UNIMARC data structure in relation to recommendation system;
3. Analyze recommender system requirements;
4. Collect data for recommender system training following GDPR guidelines;
5. Select methodology of recommendation system implementation;
6. Implement prototype recommendation system;
7. Suggest integration solution to integrate recommender system into LIBIS system;
8. Evaluate recommender system;

To reach the goal Data Science process [CMA16] with addition of Software Engineering activities was followed, see Figure 1.

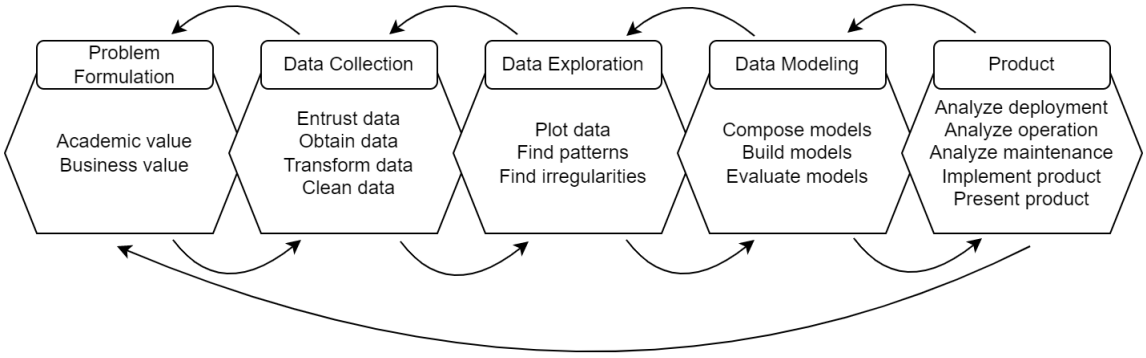


Figure 1: Process of writing this thesis

1. Literature overview

This chapter overviews literature on recommender systems (further, RS). Sources were collected using Google Scholar, IEEE.org, arxiv.org, ResearchGate, ACM Digital Library and heavily influenced by reference book on RS [Agg16] by Charu C. Aggarwal, published by Springer. Overall, there is plenty of scientific literature on recommender systems.

Overviewed literature includes: RS algorithms from the earliest and simplest models to recent advanced neural networks models; privacy in RS; evaluation of RS; attacks on RS.

The goal of RS is to make good recommendations on items that user has not yet seen. As one cannot see the future, they can only hope to build a system that would accurately predict unknown ratings. Accuracy should be evaluated by various measures. The deployed RS should not have privacy leaks and be robust to attacks.

1.1. Types of ratings

Ratings are the cornerstone of most recommender algorithms, yet some algorithms tend to work better with more detailed ratings. One exception is knowledge-based recommenders, which do not use ratings at all, instead completely rely on item attributes.

Ratings can be defined in a variety of ways, however the selected way significantly interferes with the implementation of the recommender system. Types of ratings are [Agg16]:

1. Continuous ratings: Ratings are specified using a continuous scale, which corresponds to the level of like and dislike. For example, a rating could represent any decimal value between -5 and 5. However, this approach is relatively rare, because it creates a burden on the user of having to come up with a real value from a potentially infinite range.
2. Interval-based ratings: the ratings are drawn from the N-point scale, e.g. 5-point or 7-point are commonly used. Such ratings could be numeric integers from 1 to 5, -2 to 2, or 1 to 7. The numerical values explicitly define the distances between the ratings, which are typically equidistant. The case, where an even number of points is used, is called a forced choice method because the neutral option is not present, although the neutral response may be represented by not selecting any value.
3. Ordinal ratings: Much like interval-based ratings, except that instead of integer values it uses categorical values. Examples of such categorical values may be such as “Strongly Disagree”, “Disagree”, “Neutral”. “Agree”, “Strongly Agree”. The main difference from interval-based ratings is that the difference between any pair of adjacent ratings is not assumed to be equal. However, in practice, categorical values are mapped to equally spaced integer values. In such cases, ordinal ratings are almost the same as interval-based ratings.
4. Binary ratings: Only two options are given, which correspond to a positive or negative responses. Binary ratings could be seen as a special cases of interval-based or ordinal ratings. In case when the user is neutral, he is expected to not specify rating at all. Such systems are used widely, with Youtube being the most known one.

5. Unary ratings: Such a rating system allows specifying only a positive preference for an item. This rating system is widely used in form of the “like” button on social media platforms, such as Facebook, Instagram, and Twitter. Often unary ratings are derived from implicit user feedback. For example, an act of the user opening the preview of an item or buying it could be considered a positive response. Such derived ratings can be used in conjunction with explicit feedback in order to create a robust RS.

1.1.1. Ratings matrix

In case of collaborative filter, the ratings are represented as an m×n matrix R, where one represents users and the other represents items, see example in Figure 2. This matrix is usually sparse because a single user can only watch a small fraction of all items. The purpose of the ratings matrix is to extract similarities among users. Meanwhile, content-based algorithm recommendations are limited to specific user scope, thus such matrix is not required [Agg16] .

	users					
items			2			
		5			5	
	3		1		4	
	2			1		2
					4	
		2				
	1				5	

Figure 2: Example ratings matrix R, in this case, 6×7. Numbers in the matrix represents given rating, whereas empty cells represent that the user has not given a rating: either did not watch or has not given a rating. In the general case, empty cells should be predicted.

1.2. Basic Models of Recommender Systems

The principal differences and ideas of the basic models of recommender systems are visualized in Figure 3.

1.2.1. Collaborative Filtering

Collaborative Filtering has two main techniques: Neighborhood-based and Model-based as depicted in CF techniques classification tree in Figure 4, refer to this diagram while reading further subsections.

1.2.1.1. Neighborhood-based Collaborative Filtering

Neighborhood-based collaborative filtering algorithms were among the earliest algorithms developed for collaborative filtering.

The neighborhood-based methods may be viewed as a generalization of k-nearest neighbor (KNN) classifiers, which are commonly used in machine learning. Neighborhood-based methods are generalizations of instance-based learning methods (sometimes called lazy learning methods, or memory-based learning) which are the systems that learn the training examples by heart and

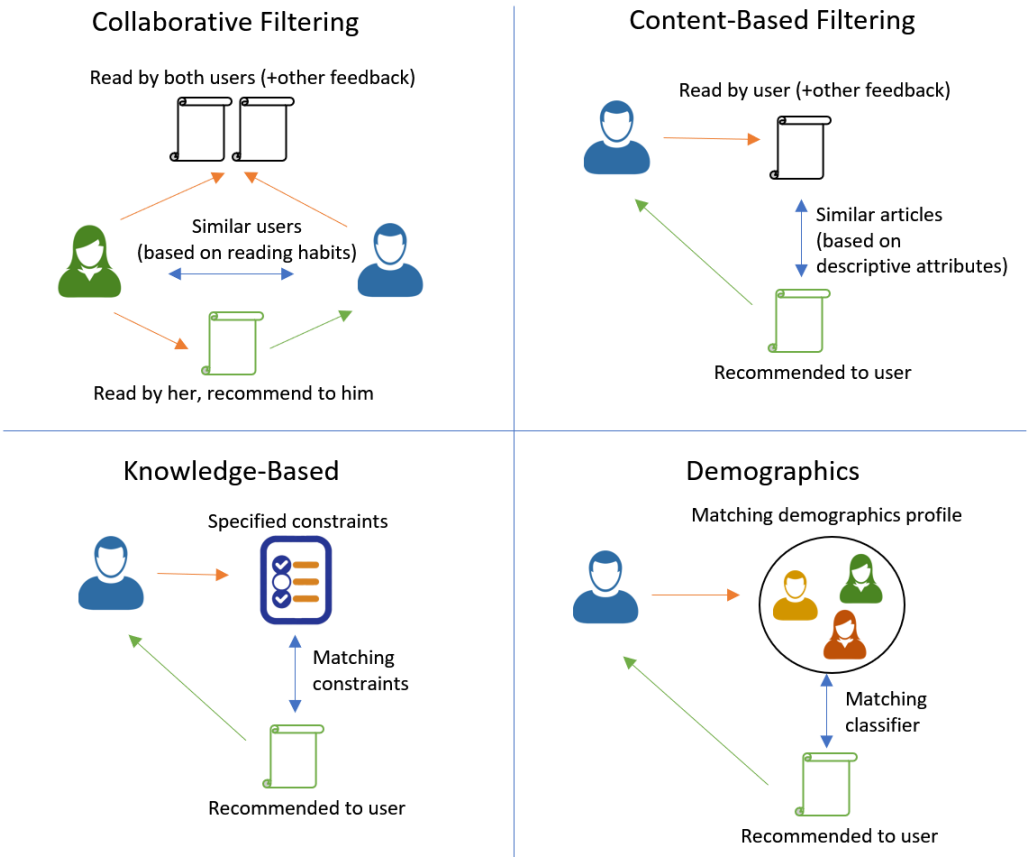


Figure 3: Abstract visualization of main ideas behind the basic models from recommender systems. This diagram influenced by visualization in [Ngo18]

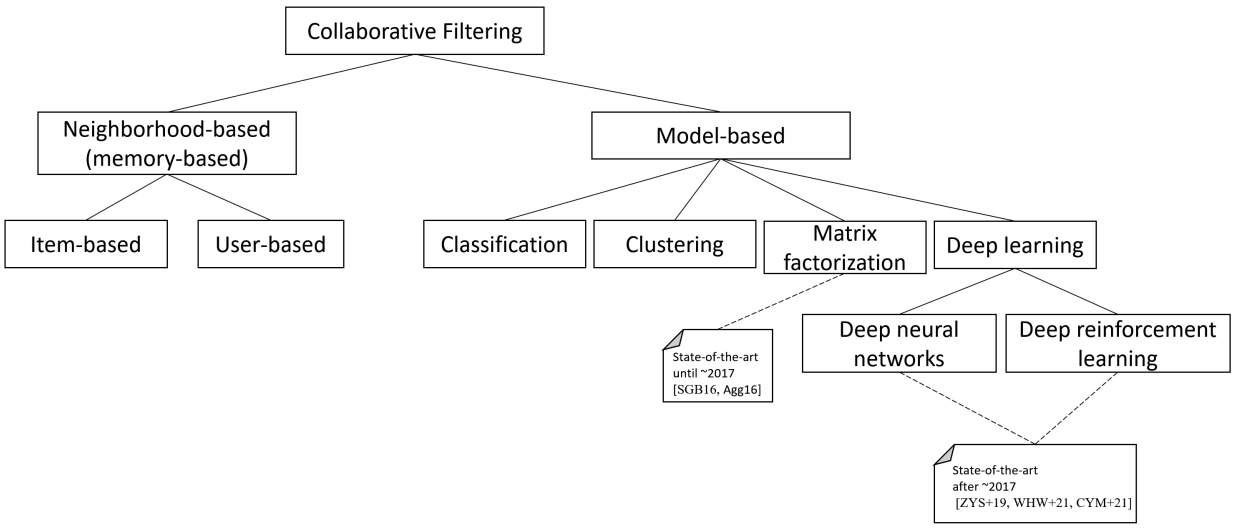


Figure 4: Classified techniques of Collaborative Filtering

then generalize to new instances based on some similarity measure. Hence, this method stores all instances in memory to be able to calculate similarity measure to the newly added entry, it stores it in generalized pockets of entries [Agg16; SS16] .

These algorithms are based on the fact that similar users display similar patterns of rating behavior and similar items receive similar ratings. There are two primary types of neighborhood-based algorithms [Agg16; SS16] :

1. **User-based collaborative filtering**: In this case, the ratings provided by similar users (similar to user A according to similarity measure calculated in the offline phase) are used to make recommendations for user A. Then, the predicted ratings for user A are calculated for each item: as the weighted average of "peer group" ratings.
2. **Item-based collaborative filtering**: In this case, in order to make recommendations for target item B, the first step is to determine a set S of most similar items (similar to item B according to similarity measure calculated in offline phase). Then, in order to predict the rating of a particular user A for item B, the ratings in set S, which are given by user A, are collected. Lastly, the weighted average of these collected ratings is calculated and the average is used as the predicted rating of user A for item B.

Table 1 lists strengths and weaknesses of Model-based CF.

Table 1: Strengths and weaknesses of Neighborhood-based Collaborative Filtering [Agg16]

Neighborhood-based Collaborative Filtering	
Strengths	**Weaknesses**
1. Simple and intuitive approach 2. Easy to implement and debug 3. Often easy to justify why a specific item is recommended 4. The justification of item-based methods is particularly well interpretable. 5. The recommendations are relatively stable with the addition of new items and users.	1. Offline phase (recalculation of similarities) of the user-based variant may be too slow or too space-intensive in large-scale settings because it requires at least $O(m^2)$ time and space (m denotes users). 2. Limited item coverage because of ratings sparsity. For instance, if the nearest neighbors of Alice have not rated the movie "Harry Potter", then it is not possible to provide a rating prediction to Alice for that movie. However, usually, only top-k items are relevant, thus if none of Alice's neighbors rated "Harry Potter" then it might be evidence that such a movie is irrelevant for Alice. 3. Sparsity also creates difficulties for robust user similarity computation when the number of mutually rated items between two users is small.

1.2.1.2. Model-based Collaborative Filtering

Although neighborhood-based methods were among the earliest collaborative filtering methods and were also among the most popular due to their simplicity, they are not necessarily the most accurate models available today. Some of the most accurate methods are based on model-based collaborative filtering techniques in general, and on latent factor models in particular: they are considered state-of-the-art. Table 2 lists strengths and weaknesses of Model-based CF.

The main model-based collaborative filtering algorithms are:

- Decision and Regression Trees. Decision trees are among the most popular machine learning algorithms, mainly because of their intelligibility and simplicity. The decision tree is a hierarchical partitioning of the data space with the use of a set of hierarchical decisions, known as the split criteria in the independent variables. For example, in a binary decision tree, one of the two branches will predominantly contain one class, whereas the other branch will predominantly contain the other class. In a primitive approach, a particular item (e.g. a particular book) is used as a split criteria. It could be split by, for example, whether the user watches the movie, or by the rating the user has given. Although, multiple measures are put in place to adapt general decision and regression trees to work in collaborative filtering RS [Agg16].
- Rule-based. Association rules learning was first proposed for discovering regularities between products in large-scale transaction data recorded by point-of-sale (POS) systems in supermarkets. For example, if a customer buys paint and tape together, it is likely that he will also buy a brush. Association rules are particularly useful for performing recommendations in the context of unary ratings matrices. The first step is to determine all association rules that have been fired by customer A. All of the fired rules are then sorted in order of reducing confidence. The first k items discovered in the resulting of these sorted rules are recommended as the top-k items to the customer. A variety of modifications of rule-based CF exist in recommender system literature [Agg16].
- Naive Bayes. This is a simple but surprisingly powerful predictive modeling algorithm. Naive Bayes classifier calculates the probabilities for every factor and every class (categorical value) from instance data. Then it selects the outcome with the highest probability. Typically, the items are treated as features (factors) and users as instances in order to infer the missing entries with a classification model [VOC+19]. In addition, a number of measures are added to adapt this Naive Bayes algorithm to RS domain, where rating sparsity is particularly problematic [Agg16].
- Matrix factorization (MF). This is the most popular implementation of the model-based CF approach, mainly due to accurate recommendations, ease of understanding and implementation. Hence the method name, the ratings of users to items are modeled with a set of latent factors that represent features of the users and items [VOC+19]. MF is based on the dimension reduction basis, where a reduced set of hidden factors contains the ratings information. Predictions are obtained by taking the dot product of the users and the items hidden factors [Agg16; BAH20; HLZ+17]. A wide variety of literature and variations exists on this technique.
- Deep learning techniques. Neural Collaborative Filtering technique is among the most popular ones, which replaces the inner product from Matrix Factorization with a neural architecture that can learn an arbitrary function from data. NCF is generic and can express and generalize matrix factorization under its framework [HLZ+17]. Multi-

layered Perceptron (MLP) [TCC21] is another neural network technique, which is a feed-forward neural network with multiple hidden layers between the input layer and the output layer. It can be interpreted as a stacked layer of non-linear transformations to learn hierarchical feature representations. Furthermore, there are algorithms based on reinforcement learning, however, the field is still in its infancy and needs plenty of advancements [ACF21].

Table 2: Strengths and weaknesses of Model-based Collaborative Filtering [Agg16; SS16]

Model-based Collaborative Filtering	
Strengths	**Weaknesses**
1. Space-efficiency: Usually, the size of the learned model is much smaller than the original ratings matrix. Therefore, the space requirements are often quite low. Meanwhile, a user-based neighborhood method might have $O(m^2)$ space complexity, where m is the number of users, and $O(n^2)$ item-based variant. 2. Training speed and prediction speed: One problem with neighborhood-based methods is that the preprocessing (offline) stage is quadratic in either the number of users or the number of items. Model-based systems are usually much faster in the preprocessing phase of constructing the trained model. In most cases, the compact and summarized model can be used to efficiently make predictions. 3. Avoiding overfitting: Overfitting is a serious problem in many machine learning algorithms, in which the prediction is overly influenced by random artifacts in the data. This problem is also encountered in classification and regression models. The summarization approach of model-based methods can often help in avoiding overfitting. Furthermore, regularization methods can be used to make these models robust. 4. No knowledge engineering required (e.g. item attributes or user attributes)	1. Requires complex machine learning knowledge 2. Hard or impossible to debug 3. In most cases, reasons behind recommendations are hardly explainable 4. Requires model retraining with the addition of new ratings 5. Various problems due to ratings matrix sparsity

1.2.2. Content-Based Filtering Recommender Systems

In content-based filtering (further, CBF) RS, the descriptive attributes of items (hence, content-based) are used to provide recommendations. The ratings and buying behavior of a user are combined with the content attributes of the items [Agg16].

For example, Alice rated the movie “Harry Potter” highly, suppose genre attributes of this movie contain keywords: fantasy and fiction. Thus we can recommend movies that have similar genre keywords. Depending on the similarity strictness, the recommendation could be “Lord of the Rings” with genre keywords fantasy fiction or “Terminator” with science fiction keywords.

When developing CBF RS, the item attributes are labeled with ratings (or other feedback information) to create a user-specific classification or regression modeling problem. Then the model is used to predict whether the user at hand will like an item for which his rating or buying behavior is unknown [Agg16]. Table 3 lists strengths and weaknesses of CBF RS.

The main content-based filtering algorithms are:

- Bag of words (BoW). BoW is simple frequency counter, which gives more importance to a word that occurs more often in a given document. Similarity between two documents using either cosine or Jaccard similarity literally checks which or how many words are exactly the same across two documents. However, this technique was generally replaced by word embeddings.
- Term Frequency-Inverse Document Frequency (TF-IDF). TF-IDF helps in evaluating importance of a word in a document. TFIDF is also sort of like a frequency counter, which gives more importance to a word that occurs less often in the whole document. The TF part describes how frequent the term/word appears in the document and also represents the document in vector form. Firstly dictionary of words (also known as bag of words) is created of words present in the whole document space, hence stop words are ignored as they are contained in most of the documents and do not help in selecting important words. For example, if bag of words consists of 1-Machine 2-Learning 3-Recommender 4-Systems 5-Library 6-Public, and we have documents with text R1-Machine Learning Recommender, R2-Machine Learning in Public Library, R3-Public Library Recommender Systems. Then vectors would be $\overrightarrow{R1}(111000)$ $\overrightarrow{R2}(110011)$ $\overrightarrow{R3}(001111)$ which could be used to calculate similarities. However, TF alone inherits a problem, gives more importance to words/terms occurring frequently while ignoring the importance of rare words/terms. This is not an ideal situation as rare words contain more importance or signal. This problem is resolved by IDF, which logorithmic normalization: IDF = ln (Total number of documents / Number of documents with term t in it). Further, TF-IDF weight is calculated: TF-IDF = TF (t,d) * IDF(t,D), where t is term, d is specific document and D is document space. Lastly, cosine similarity is used to find similar documents [QA18]. However, this technique was generally replaced by word embeddings.
- Text Semantics Based Similarity (Word Embeddings). Semantic Similarity or Semantic Textual Similarity, is a task in the area of Natural Language Processing (NLP) that scores the relationship between texts or documents using a defined metric. For eg. we know that the words 'tiger' and 'leopard' are related, also 'zebra' and 'strips' related. Word2Vec (W2V) and its variantions (e.g. AvgW2V, IDF Weighted W2V) are popular techniques that learns meaningful relations and encodes the relatedness into vector similarity [Rav18]. Some more recent approaches use BERT language transformers to learn word embedding between item contents [DCL+19; SLW+19].
- Weighted Similarity Using Variety of Features. For each item which has inherited/latent/engineered features, separate feature vectors are constructed using One-Hot-Encoding techniques or numeric values. Once constructed, these vectors can be concatenated with vectors of other features. Similarity metrics used mostly:
  - Cosine Similarity: The Cosine angle between the vectors.
  - Dot Product: The cosine angle and magnitude of the vectors also matters.
  - Euclidean Distance: The element-wise squared distance between two vec-

tors

  - Pearson Similarity: The ratio between the covariance of two variables and the product of their standard deviations

- Model-based rating predictors. In this category individual models are built for each user which predict ratings for items one has not seen. Models used are Linear regression, Lasso, KNN, RFR, SVR. However, such approaches are rare, both in literature and in practical examples.

Table 3: Strengths and weaknesses of Content-based Filtering [Agg16; SS16]

Content-based Filtering	
Strengths	**Weaknesses**
1. CBF methods are advantageous in making recommendations for newly added items when sufficient rating data is yet not available. The supervised user-specific model is able to leverage his rating pattern in conjunction with the new item attributes to make recommendation. 2. Does not rely on the other users' preferences. 3. Comparison between items is possible.	1. In most cases, CBF methods provide obvious recommendations due to the use of content attributes. As a result, the user will be recommended with similar items at best. Whereas items with a particular set of keywords, which the user has never consumed, have no chance of being recommended. This is because the community knowledge from similar users is not leveraged, hence, the constructed model is user-specific. This reduces the diversity of the recommended items, which is generally undesirable. 2. Even though CBF methods effectively provide recommendations for new items, they are ineffective at providing recommendations for new users (cold-start problem). This is due to the fact that the training model for the target user requires a history of his ratings. In addition, having a large number of ratings is important in order to make robust recommendations without overfitting.

1.2.3. Knowledge-Based Recommender Systems

Knowledge-based (further, KB) RS are particularly useful: 1) When items are not purchased very often. Such examples include items such as real estate, vehicles, financial services, or expensive luxury goods; 2) When item domain tends to be complex in terms of its varied properties, and it is hard to combine sufficient ratings with a large number of combinations.

The vehicles for sale search domain illustrates such RS well. Vehicles are bought rarely and the vehicle domain has a wide variety of combinable properties. Cars have a wide variety of makes, models, colors, engine options, transmission options, interior options, and user interests may be a very specific combination of the options.

KB systems can be classified by the type of interface (and corresponding knowledge):

1. Constraint-based RS: Users specify requirements or constraints (e.g., lower or upper bounds) on the domain-specific items attributes. For example, user sets constrains: Cars with manual transmission which have electronically adjustable seats. Users may relax some of their constraints if too few results are returned, or may add more to further limit results. This search process is interactively repeated until the user retrieves desired

results.

2. Case-based RS: Specific cases are specified by the user as targets or anchor points. Domain-specific similarity measures are carefully defined to allow retrieval of similar items to the anchor cases. The returned results may further be used as new target cases with some interactive modifications by the user. For instance, a user could set the anchor point: houses in 5 miles range similar to the house on Lake street 15 but with exactly 3 bedrooms.

Interactivity in KB RS is achieved by guiding user through one or more of the following methods:

1. Conversation-based: the user preferences are determined iteratively in the context of a feedback loop. This method is mainly used when item domain is complex and the user preferences can by determined only in the process of an iterative conversation.
2. Search-based: the user preferences are collected by a set of questions in specific search interfaces (checkboxes, dropdown lists, etc.).
3. Navigation-based: the user specifies change requests to the item being currently recommended. Through an iterative set of change requests, the user arrives at a desirable item. For example, when a specific house is being recommended, the user-specified change requests "Show similar house about 10 miles west of the currently recommended house".

It is worth noting that both KB-RS and CBF-RS systems depends significantly on the attributes of the item. The main difference is that CBF-RS learn from past user behavior, whereas KB-RS recommends based on active user specification of their interests.

1.2.3.1. Utility-Based Recommender Systems

Utility-based recommender systems compute the probability of a user liking the item based on a function that is known in advance. The functions can be viewed as a kind of external knowledge, thus such RS is considered a specific case of KB-RS. In addition, utility (or perceived value) functions are used frequently in various ways for ranking recommended items.

1.2.4. Demographic Recommender Systems

Demographic recommender systems (further, D-RS) leverages the demographic information of the user to learn classifiers that can map specific demographics to ratings or buying habits. Demographics comprise an array of socioeconomic information, including the breakdown of a population by gender, age, ethnicity, area, education, etc [Agg16].

Some early D-RS recommended books based on the manually assembled stereotypes. Other works observed that the same demographic groups used in marketing research can be used to recommend items. However, there is relatively less research in this particular technique, some surveys do not even mention it [SS16].

The main difference is that KB-RS recommends items according to user-specified constraints, whereas D-RS maps user to a demographic profile and relates it to items of interest. Such recom-

mender systems are not very different from the vanilla classification and regression modeling problem, where feature variables correspond to the demographic profiles and the dependent variables correspond to the ratings or buying behavior [Agg16].

1.2.5. Popularity-Based Recommendation System

Popularity-Based (also referred as Baseline) RS is a type of recommendation system which works on the principle of popularity and or anything which is in trend. These systems check about the product or movie which are in trend or are most popular among the users and directly recommend those.

For example, if a product is often purchased by most people then the system will get to know that that product is most popular so for every new user who just signed it, the system will recommend that product to that user also and chances becomes high that the new user will also purchase that.

It does not suffer from cold start problems and does not require user's historical data, thus could provide business value from day one. On the other hand, it is not personalized at all.

1.2.6. Hybrid Recommender Systems

The aforementioned recommendation systems use different sources of input, and they may work well in appropriate scenarios. However, these different systems have different strengths and weaknesses. Some, such as KB-RS are more effective in cold-start settings when a significant amount of data is not available. Whereas, other RS, especially collaborative methods, are more effective when a lot of data is available [Agg16; SS16]. Research on hybrid recommender systems mainly focuses on cold-start, data sparsity and accuracy problems [ÇM19].

Where a wider variety of inputs is available, one has the flexibility of applying different types recommender systems for the same task. This brings many opportunities for hybridization, where the various aspects from different types of RS are combined to achieve the best of all worlds [Agg16].

Hybrid recommender systems (further, H-RS) are closely related to the field of ensemble learning which use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. Not only ensemble methods allow to combine inputs of multiple data sources, but it also allows improving the effectiveness of a particular class of recommender systems (e.g., collaborative system) by combining multiple models of the same type [Agg16].

Hybrid model can be classified as pipeline-based hybrid and design-based hybrid, further subsections describe these techniques in detail.

The main types of hybrid recommender systems are:

- Weighted. In the weighted recommendation system multiple recommender systems are employed in parallel. The weighted recommendation system will take the outputs from each of the models and combine the result in static weightings. For example, we can combine a content-based model and a item-item collaborative filtering model, and each takes a weight of 50% toward the final prediction. The benefit of using the weighted

hybrid is that we integrate multiple models to support the dataset on the recommendation process in a linear way. Weighted hybrid is the most recurrent, obviously because of the simplicity and dynamicity it offers. [ÇM19].

- Switching. The switching hybrid selects a single recommendation system based on the situation. The model is used to be built for the item-level sensitive dataset, we should set the recommender selector criteria based on the user profile or other features.
  The switching hybrid approach introduces an additional layer upon the recommendation model, which select the appropriate model to use. The recommender system is sensitive to the strengths and weakness of the constituent recommendation model [ÇM19].
- Mixed. Mixed hybrid approach first takes the user profile and features to generate different set of candidate datasets. The recommendation system inputs different set of candidate to the recommendation model accordingly, and combine the prediction to produce the result recommendation.
  The mixed hybrid recommendation system is able to make large number of recommendations simultaneously, and fit the partial dataset to the appropriate model in order to have better performance [ÇM19].
- Feature combination. In feature combination hybrid, a virtual contributing recommendation model is added to the system, which produces features towards the original user profile dataset [ÇM19].
  For example, we can inject features of a collaborative recommendation model into an content-based recommendation model. The hybrid model is capable to consider the collaborative data from the sub system with relying on one model exclusively [ÇM19].
- Feature Augmentation. In this class of hybrids, one of the combined techniques is used to produce an item prediction or classification which is then comprised in the operation of the other recommendation technique. Feature augmentation hybrids are order-sensitive as the second technique is based on the output of the first. For example an association rules engine can generate for any item, similar items which can be used as augmented item attributes inside a second recommender to improve its recommendations [ÇM19].
- Cascade. Cascade hybrid defines a strict hierarchical structure recommendation system, such that the main recommendation system produce the primary result, and we use the secondary model to resolve some minor issues of the primary result, like breaking tie in the scoring. Successor's recommendations are restricted by predecessor, thus it may not introduce additional items as a result producing very precise results [ÇM19].
- Meta-Level. Meta-level hybrid is similar to the feature augmentation hybrid, such that the contributing model is providing augmented dataset to the main recommendation model. Different from the feature augmentation hybrid, meta-level replaces the original dataset with a learned model from the contributing model as the input to the main recommendation model. In short, successor exploits a model delta built by predecessor. Unlike in Cascade hybrid, in Meta-level subsequent recommender has no restrictions

on items [ÇM19].

1.3. Privacy in Recommender Systems

Users provide implicit and explicit feedback to the RS, however, this feedback contains significant information about the user. Analyzing feedback information may relatively easily reveal information about political affiliation, sexual orientation, and other personal preferences [FPG+18; SS16]. As this information is highly sensitive, it leads to privacy concerns, which in turn highly restrain the release of real historical datasets which are necessary for the advancement of recommendation algorithms [Agg16].

Anonymization is usually a trade-off between utility and privacy, with higher privacy data becomes less accurately represented. As one seeks reasonable utility anonymity in a dataset is compromisable, [Agg16; OOK+15].

The two classes of techniques are used to preserve privacy [Agg16]:

1. Privacy at data collection time: the data collection stage is modified in a way that individual ratings are not collected. This may be achieved by data perturbation or aggregation techniques. Typically, specialized secure user interfaces and secure data collection plug-ins are required for such an approach. The advantage is that users are assured that no single entity has access to their private data, at least in its exact form. However, in practise, this approach is rarely used mainly because the user is required to gain access to specialized interfaces or infrastructures. In the case of the aggregation approach, it is harder to derive appropriate value from aggregated data.
2. Privacy at data publication time: a trusted entity has access to the raw ratings data. When such a trusted entity wants to publish data for recommendation system advancements, it releases the dataset with de-identification, anonymization and perturbation techniques applied.

Multiple papers exist which show that varying degree of de-anonymization is possible using either structural de-anonymization techniques [JLS+14] or background knowledge-based techniques [LLJ+18; OOK+15].

1.4. Evaluating Recommender Systems

Recommender systems share some conceptual similarities with the classification and regression modeling problem. Unlike the general approach to classification and regression where the missing class variable needs to be predicted, recommender systems need to predict any missing matrix entry from the observed entries in the remaining matrix. Therefore, the recommendation problem can be viewed as a generalized classification problem [Agg16].

Many of the models used for the evaluation of classification and regression can be used for evaluating RS, although with some modifications. However, as typically recommender systems have multiple stages, most commonly: rating prediction and ranking. The evaluation of the stages

varies significantly. The former is closely related to classification and regression modeling. Meanwhile, the latter is related to the evaluation of information retrieval applications [Agg16].

1.4.1. Evaluation Paradigms

There are three main types of evaluation of recommender systems: user studies, online evaluations, and offline evaluations with historical data sets. The first two include the participation of users, however, they differ in a way how the users are recruited for the studies [Agg16].

1.4.1.1. User Studies

In user studies, users are explicitly recruited and asked to perform specific tasks regarding the usage of the recommender system. Feedback is collected from the user and also interaction information may also be collected in an automated way by the system [Agg16; BL15].

An advantage of user studies is that they provide information about the user interaction with the recommender system. However, the awareness of the user of being a test subject can often bias his choices and actions. Also, such recruitment is usually difficult and expensive, and participants may not represent the general population. For instance, when testing music RS, participants are likely to be music enthusiasts. Due to these disadvantages, the results from the user evaluations should be viewed skeptically [Agg16; BL15].

1.4.1.2. Online Evaluation

Unlike in user studies, in online evaluations, the users are real users in a fully deployed production system. Such an approach is less susceptible to the test-subject bias because users are using the system in their natural use-cases [Agg16; BL15].

Typically, users are sampled randomly, and various changes of algorithms are tested against them. A typical metric used to measure the effectiveness of RS is conversion rate, which measures the frequency with which the user selects a recommended item. If desired additional information could be added to the conversion action, such as profits. These methods are also referred to as A/B testing, with the basic idea to compare two algorithms as follows [Agg16]:

1. Segment the users into two groups A and B.
2. Use one algorithm for group A and the other for group B for a period of time while keeping all other conditions as similar as possible.
3. At the end, compare the conversion rates.

The main drawback is that online evaluation requires a large number of users enrolled in the system, thus it is hardly usable during the project start-up phase. Furthermore, such evaluation may only be accessible to the owner of the specific commercial system and with appropriate risk-mitigation measures, for example, only a small fraction of users gets to use a new algorithm. Thus, research may be limited [Agg16].

1.4.1.3. Offline Evaluation with Historical Data Sets

Offline testing uses historical data, such as ratings. In some cases, additional temporal data, such as conversion time-stamp, may also be associated with the ratings. Once a dataset is available, it could be used as a standardized benchmark to compare algorithms across a variety of settings. Furthermore, other datasets from the different domain can be used to check the generalizability of the RS [Agg16; BL15].

In general, offline methods are the most popular techniques for testing RS algorithms, mainly because standardized frameworks and evaluation measures have been developed. Additionally, it is generally cheaper, more accessible to researchers, and less risky than online testing [Agg16; BL15].

The main drawback of offline evaluations is that they do not measure the actual tendencies of the user reaction to recommended items in the future. For instance, as data evolves over time, the current predictions may not reflect the most appropriate predictions for the future. In addition, offline evaluation is primarily focused on accuracy, yet does not capture measures such as serendipity and novelty, which have important long-term effects on conversion rate. Although having disadvantages, offline methods are the most widely accepted techniques, because of statistically robust and easily understandable quantifications [Agg16; BL15].

1.4.2. Evaluation Factors

This subsection overviews the main recommender system evaluation factors. Hence, these metrics highlight the reasons why collaborative filtering recommenders are more widely used. For example, content-based RS would have the lowest score possible score on serendipity because it uses only user-specific conversion history, thus recommendations cannot step out to items that are not known to the user.

1.4.2.1. Accuracy

Accuracy of estimated ratings is the fundamental measure though with recommender systems are evaluated. The accuracy metrics are often similar to those used in regression modeling. Let R be the ratings matrix in which r_{uj} is a known rating of user u for item j, whereas $\hat{r}_{uj}$ is predicted rating by RS [Agg16; SS16].

Therefore, the basic notion of entry-specific error is: $e_{uj} = \hat{r}_{uj} - r_{uj}$

Meanwhile, the overall error over the set E of entries in the ratings matrix on which evaluation is performed can be calculated by:

1. Mean absolute error, MAE. However. it does not severely penalize large errors.
2. Mean squared error, MSE. It does severely penalize large errors. Although due to square, the units are different than the dependent variable.
3. Root mean squared error, RMSE. It gives relatively high weight to large errors. Similar to MSE, the RMSE should be more useful when large errors are especially undesirable. In addition, it has the same units as the dependent variable. Also, the most commonly used.

1.4.2.2. Coverage

Even though a recommender system is highly accurate, it may not be able to recommend a certain portion of the items. This is the measure for coverage. This limitation is inherited due to the ratings matrix sparsity. However, different algorithms tend to have different levels of providing coverage. Some recommender systems use defaults for ratings that are not possible to predict, thus reaching 100% coverage. On the other hand, defaults are lowering accuracy metrics. Therefore, a trade-off between accuracy and coverage needs to be incorporated. There are two types of coverage, user-space coverage and item-space coverage [Agg16; GDJ10; SS16].

User-space coverage measures the fraction of users for which at least k ratings may be predicted, where the value of k is the expected size of the recommendation list. When fewer than k ratings can be predicted for a user, it is no longer possible to present a meaningful recommendation list of size k to the user. An alternative definition of user-space coverage is the minimum amount of profile that must be built for a user until it is possible to make recommendations for that user [Agg16; GDJ10].

Item-space coverage measures the fraction of items for which the ratings of at least k users can be predicted. In practice, this notion is rarely used, because recommender systems generally provide recommendation lists for users, and rarely otherwise [Agg16; GDJ10].

1.4.2.3. Diversity

Diversity implies that recommendations in a single recommended list should be as diverse as possible. For example, if a set of recommendations is not diverse, then if the user does not like the first item then there is a good chance that he might dislike all of them [Agg16].

Diversity can be measured in terms of the content-centric similarity between pairs of items in the set. The average similarity across the recommendation lists of provided for users can be reported as the diversity [Agg16].

1.4.2.4. Serendipity

In simpler terms, serendipity means “lucky discovery” and measures the level of surprise in successful recommendations. In contrast to novelty, novelty merely requires that the use was not aware of the recommendation earlier. Serendipity, on the other hand, is a stronger condition. All serendipitous recommendations are novel, but the opposite in not always true. Serendipity can be viewed as a departure from obvious recommendations. This metric has an important long-term impact on conversion rate [Agg16; GDJ10].

Serendipity can be measured by both online and offline evaluation methods [Agg16; GDJ10]:

1. Online methods: the RS asks a user to provide feedback on the usefulness and obviousness of a recommendation. The fraction of recommendations that are both useful and non-obvious, may be used as a measure for serendipity.
2. Offline methods: a primitive recommender that has a high propensity for obvious recommendations is selected, for example, content-based RS. Then, serendipity is mea-

sured by the fraction of recommended items in the top-k lists that are correct and not recommended by the primitive RS.

Hence, it is insufficient to measure only the fraction of non-obvious items, because the system may be recommending unrelated items. Therefore, the usefulness of recommendations should always be included when measuring serendipity [Agg16; GDJ10].

1.4.2.5. Novelty

Novelty evaluates the plausibility of giving recommendations that the users were not aware of, or that they have not seen before. Another definition is that a novel recommender system is better at providing recommendations on items that are more likely to be selected by the user in the future, rather than at the present. For example, content-based methods tend to recommend obvious items that the user would select at the present, thus novelty in such methods is poor [Agg16].

The straightforward approach to measuring novelty is through online experimentation where users are explicitly asked if they were aware of an item previously [Agg16].

Although, if timestamps are available with the ratings, it is also possible to estimate novelty with offline evaluation, where the idea is that novel RS is better at recommending items that are more likely to be selected by the user in the future, rather than at the present time. Then the steps are performed [Agg16]:

1. All ratings that were created after a certain point in time t_0 are removed from the training data.
2. Some of the ratings occurring before t_0 are also removed.
3. RS is trained with these ratings removed.
4. With the removed items used for scoring purposes, for each item rated before t_0 and correctly recommended, the novelty score is penalized. Hence, RS recommended item, that is likely to be selected in the present time.
5. With the removed items used for scoring purposes, for each item rated after t_0 and correctly recommended, the novelty score is rewarded.

1.4.2.6. Scalability

Collecting large numbers of ratings and a wide array of implicit feedback information is relatively easy. Thus, as sizes of the datasets are continuously increasing, it is essential to design RS that performs effectively and efficiently in the presence of large amounts of data. These are the main measures for scalability [Agg16]:

1. Training time: the overall time required for a training phase. Typically the training is done offline in most cases. Therefore, a training time of a few hours is acceptable in most realistic scenarios.
2. Prediction time: once a model is trained, time is measured on how long it takes to determine the top recommendations for a particular user. Hence, this determines the latency with which the user receives responses from RS, thus it is crucial for prediction time to be low.

3. Memory requirements: when ratings matrices are large, sometimes it is a challenge to hold the entire matrix in the main memory. To prevent difficulties of using such RS in large-scale settings, the algorithm should be designed to minimize memory requirements.

1.5. Attack-Resistant Recommender Systems

Recommender systems are typically open "Garbage in, garbage out" - like any other data-mining system, recommender system effectiveness exclusively depends on the quality of the data available to it [Agg16].

However, in the context of recommender systems, there are significant motivations to submit incorrect feedback about the items for malicious reasons or personal gain. Also, some attacks may be designed to cause misbehavior of the underlying RS. Although such attacks are relatively rare compared to ones motivated by personal gain. The type of motivation behind the attack is categorized as follows [Agg16; LCL+20]:

- Push attacks: fake positive feedback is submitted in order to maximize the conversion rate of an item. For example, the adversary may use push attack to maximize sales of a book published by attacker.
- Nuke attacks: fake negative feedback is submitted in order to minimize the conversion rate of an item. For example, attacked seeks to lower the presence of a competitor item in the recommendation lists.

1.5.1. Types of Attacks

An attack that requires a smaller number of injected (fake) profiles is referred to as efficient attack because they are harder to detect. Whereas an inefficient attack is one that requires a large amount of injected users, that is easily detectable. As a result, well-designed attacks are able to influence the recommender predictions with a small amount of injected profiles. Meanwhile, a carelessly thought attack may have no effect on the predicted ratings at all [Agg16].

Furthermore, attacks are described by the amount of knowledge about the ratings distribution required to mount a successful attack. An attack that requires only limited knowledge are referred to as low-knowledge attacks, and vice versa for high-knowledge attacks. With better knowledge attacker is able to attack more efficiently, however obtaining knowledge about the ratings database is often difficult. Such knowledge is embedded in the ratings of additional items (other than the target items) in the injected profile. These items are referred to as filler items [Agg16].

Attack models have trade-offs between the efficiency of the attack and the amount of knowledge required to mount a successful attack. In addition, the effectiveness of a particular attack varies depending on the specific recommendation algorithm, in other words, some RS algorithms are more prone to attacks than others [Agg16].

1.5.1.1. Random attack

Random attack requires knowledge of the global mean of all ratings across all items, obtaining which is typically not very difficult in most settings [Agg16; LCL+20].

Then, filler items are assigned ratings that are drawn from a probability distribution that is distributed around the global mean. Meanwhile, the target item is either set to maximum possible rating r_max or minimum possible rating, respectively for push attack or nuke attack. Hence, filler items should be randomly selected from each injected profile, thus making the attack less suspicious [Agg16; LCL+20].

1.5.1.2. Average attack

From the perspective of filler item selection, average attack is similar to random attack. The main difference is how the ratings are assigned to the filler items. In this case, ratings of filler items are assigned to approximately the average of the specified item [Agg16; LCL+20].

This method requires a greater amount of knowledge than the random attack, because one must know the average ratings of each item in the filler items set. Some platforms directly advertise average ratings for items, whereas on the other average values can be easily computed. Hence, one could use a small set of candidate items with a calculated item-specific rating average, from which one would randomly draw filler items [Agg16].

1.5.1.3. Bandwagon attack

The problem with the aforementioned is that an inherently sparse ratings matrix prevents injected profiles from being sufficiently similar to the real profiles. However, if the adversary would select too many items in order to increase similarity, then the attack becomes conspicuous. In the case of collaborative filtering, a fake profile does not impact recommendations when it does not have any commonly rated items with the target real user [Agg16].

The bandwagon attack leverages the fact that a small number of items are very popular in terms of the number of high ratings they receive. As a result, if these items are rated highly by the fake user, then the chance of a fake user being similar to the target user is increased [Agg16; LCL+20].

1.5.1.4. Probe attack

Probe attack uses a seed profile created by the adversary, and the predictions generated by the RS are used to acquire knowledge about related items and their ratings. The seed profile can then be extended further by observing the operation of a user-based collaborative filtering algorithm when the seed profile is used as the target user. Depending on whether it is push or nuke attack, the rating of the target item is set to r_{max} or r_{min}, respectively [Agg16].

1.5.2. Attack counter-measures

From the point of view of maintaining a robust recommender system, the best way to counter attacks is to detect them and apply corrective actions, such as the removal of fake user profiles. Although, one must take into account, that the removal of fake profiles is a mistake-prone process, in which genuine profiles might be removed. This results in a natural trade-off between the precision and recall of fake profile removal. Correspondingly, attack detection algorithms are often measured in terms of this precision and recall. An alternative way of evaluating the effectiveness of attack is by measuring the impact of profile removal on recommender system accuracy [Agg16].

Supervised attack detection algorithms are often used. Individual user profiles are characterized as feature vectors, where features could be extracted accordingly to the attack being countered. Such features could include: number of similar users, average of overall ratings, number of impacted recommendations, etc [Agg16].

Another robust strategy is to prevent automated attacks with CAPTCHAs used on profile creation. As attacks typically require a significant number of fake profiles, depending on the RS algorithm, it may require between 3% to 5% fake profiles to initiate an attack. Thus, if the user-base of RS is in the number of millions, thus CAPTCHAs force the adversary to manually create profiles which is hard [Agg16].

1.6. Literature overview conclusions

The following conclusions could be made after reviewing the literature:

1. Model-based Collaborative Filtering techniques are the most commonly used. Whereas other techniques may be used only in specific domains or as supporting algorithms in a hybrid approach with Collaborative Filtering.
2. Model-based Collaborative Filtering has multiple implementations, although variants of deep learning are considered to be the state-of-the-art.
3. Publicly released historic ratings dataset should be robustly anonymized. If used privately, the data minimization principle should be taken into account.
4. Offline evaluation methods are among the most popular techniques and can robustly evaluate a recommender system. Standardized frameworks and evaluation measures have been developed for such cases.
5. In order to counter attacks on the recommender system, tools need to be implemented either for fake profile detection or embedded in recommender system design.

2. Dataset preparation

The user feedback on bibliographic records resides in two separate projects: LIBIS and METIS (Modern Digital Content Storage and Dissemination; acronym for Lithuanian: „Modernaus elektroninio turinio išsaugojimas ir sklaida“). In short, LIBIS is responsible for library back-office processes, whereas, METIS is the portal for readers (ibiblioteka.lt). Both modernized projects have been launched to production in July 2021. The next subsections describe available data, dataset design process.

Code for dataset preparation is provided at gitlab.com/senvaitis/libis-rs-dataset-preparator.

2.1. LIBIS data

The following subsections describe the original data structure, data quantities and properties, and the process of composition of the dataset.

2.1.1. Original data structure

The original entity–relationship model of LIBIS database in regards to the user reading information is shown in Figure 5. The data model has limitations on extracting record properties as they are serialized and stored in JSON format, also such parsing is resource and time-intensive and would require sophisticated SQL queries. However, record properties could instead be extracted from Solr index, where all records are indexed for searching, see Figure 6. Unlike in database, Solr allows for relatively easy querying of record unimarc attributes with close to instant result retrieval.

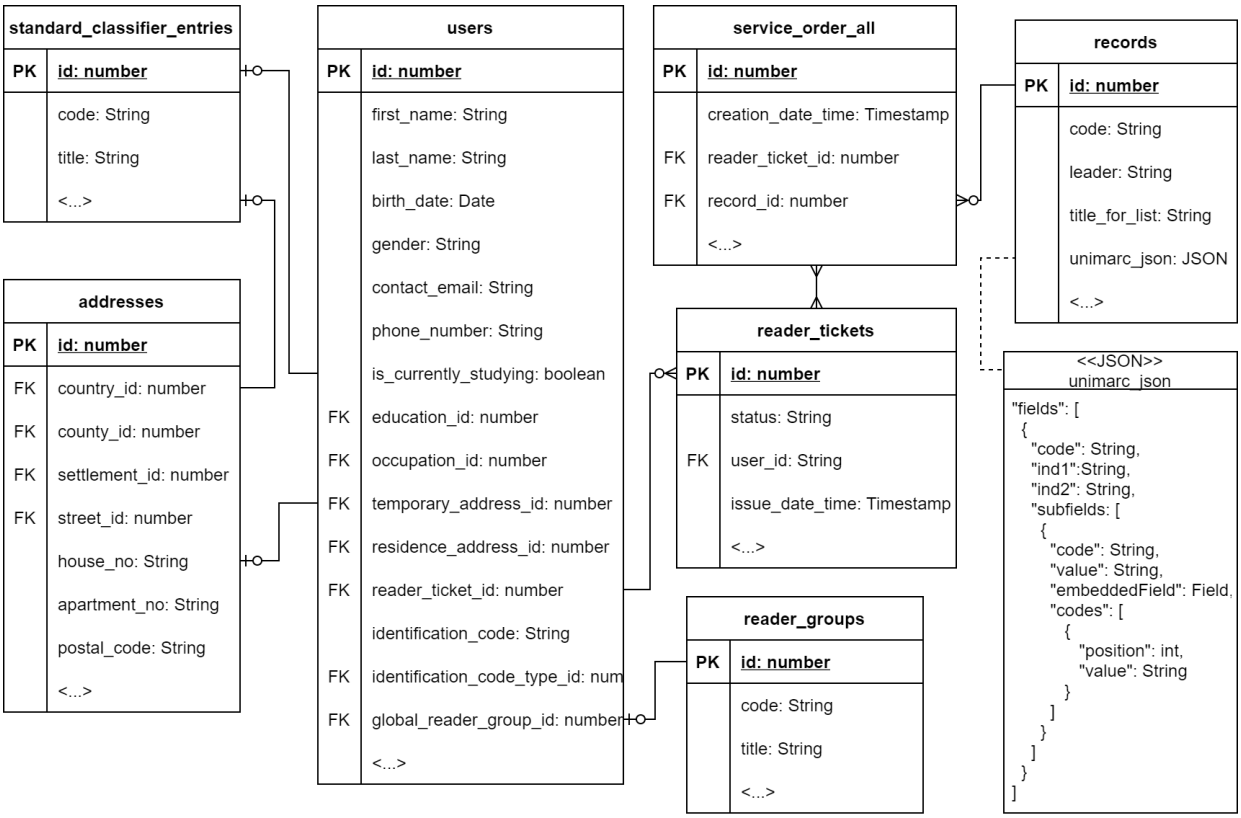


Figure 5: LIBIS: database structure in regards to user reading history

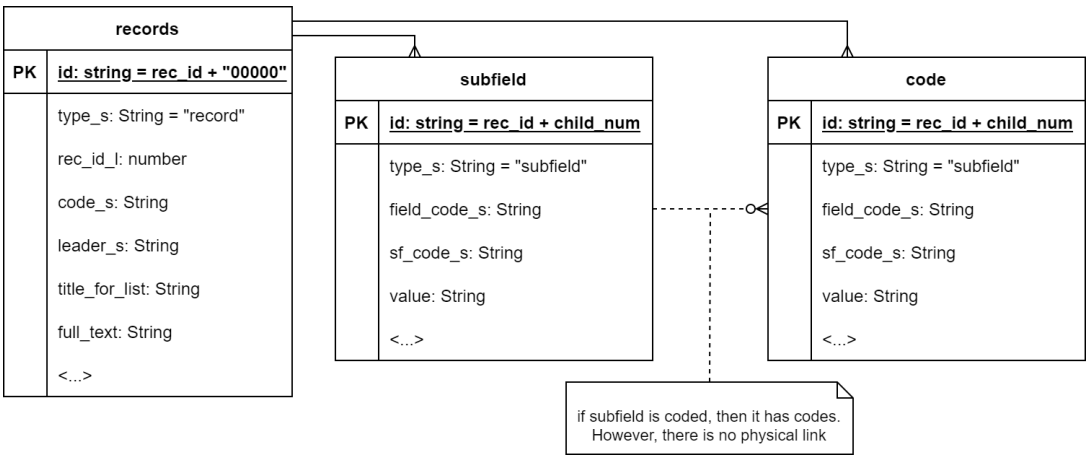


Figure 6: LIBIS: Solr index structure in regards to record properties

2.1.2. Data quantities and properties

LIBIS data on its own only provides implicit feedback on records in form of orders. Although LIBIS data could be combined with METIS data which has explicit feedback on records, however amount of such explicit feedback is far lower, more-in depth analysis is provided in METIS data section. In comparison, from July 2021 to January 2022 total of 1.75M orders were issued, meanwhile, only 1.2K ratings were given, which estimates that roughly 0.1% of record orders can be related to record ratings.

As mentioned in the introduction, this paper focuses only on books (am0-records) because other types of records (newspapers, magazines, video material, etc) is less tangible and recommendations on such records are in different format thus may cause unexpectedness.

LIBIS orders by year and the relationship between all orders and book orders are visualized in Figure 7. Hence, until 2015 orders were issued on physical paper reader cards, thus are not available. Having roughly 4M of implicit feedback every year is a sufficient amount of data for recommender system machine learning algorithm training. Table 4 shows total counts of data related to recommender system training in the upper part, whereas the lower part filters counts in regards to books (am0-records).

Some implicit feedback collaborative filtering algorithms use repeated user-item interaction count to infer confidence in positive feedback. In LIBIS case, over 90% of order cases are not repeated, whereas only 7% of order cases are 2-times orders, see Figure 8. This makes it harder to develop a robust confidence metric. Hence, extended orders are counted as repeated orders.

Hybrid recommender systems, which mainly solve the cold start problem, require contextual information. In LIBIS case, contextual user information is available for the majority of users as depicted in Figure 9, also distribution of information is adequate. For example, Figure 10 depicts the distribution of users according to their age group. On the other hand, record information required more sophisticated feature engineering as one has to have an understanding of both unimarc and domain knowledge.

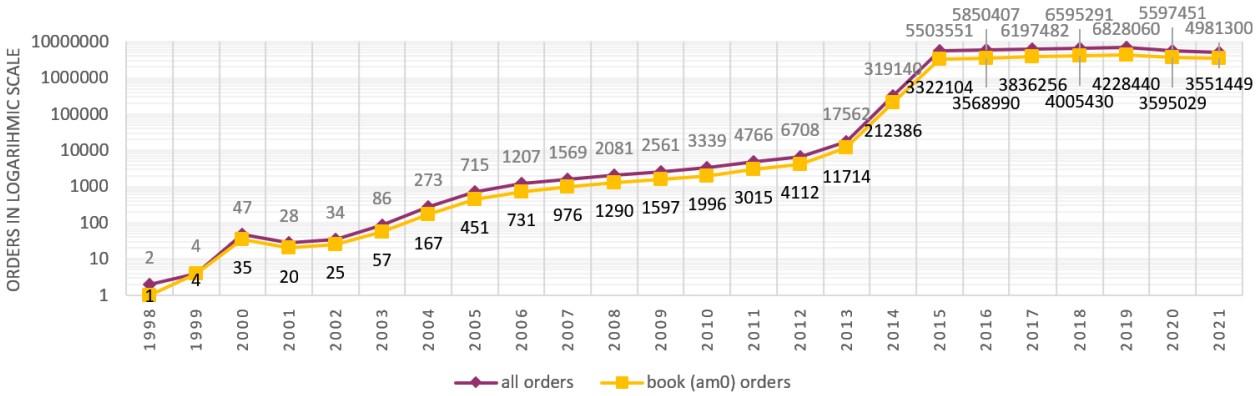


Figure 7: LIBIS: order information by year, hence most of the libraries issued orders on paper cards until 2015, thus are not available. 63% of all orders are book orders. Note, that orders are provided in a logarithmic scale.

Table 4: LIBIS data related to recommendation systems

Available data from LIBIS	
Data & Description	**Count**
Records. Bibliographic records which represent books, magazines, articles, music notes, video material, sound material, etc.	8.3M
Records with at least one order. Bibliographic records which represent books, magazines, articles, music notes, video material, sound material, etc.	0.57M
Users. Users of public library.	1.09M
Users with at least one order. Users of public library.	0.69M
Orders. Order of exemplars which are directly related to bibliographic record.	42.3M
Available book (am0 records) data from LIBIS	
Data & Description	**Count**
Books (am0-records). Records which represent printed books: leader (24-symbol code) has “am0” in positions from 7 to 9.	1.1M
Books (am0-records) with at least one order. Bibliographic records whose leader (24-symbol code) has “am0” in positions from 7 to 9.	0.3M
Users with at least one book order. Users of public library.	0.66M
Users with at least 20 book order. Users of public library.	0.27M
Book orders. Order of exemplars which are directly related to bibliographic record.	26.3M

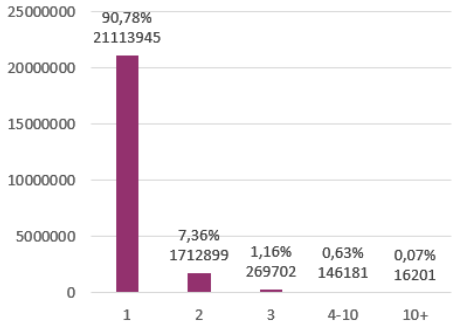


Figure 8: LIBIS: distribution of repeated reads. Orders with 10+ reads are likely to be the case of incorrect data)

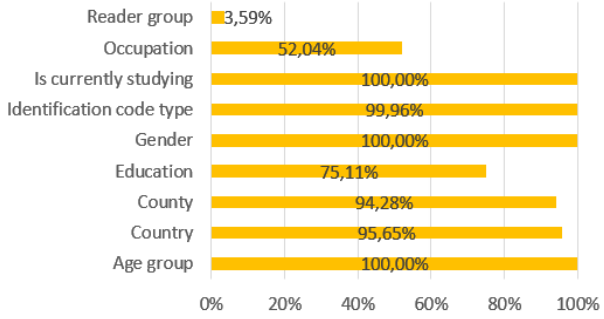


Figure 9: LIBIS: user contextual information availability (who have ordered at least one book)

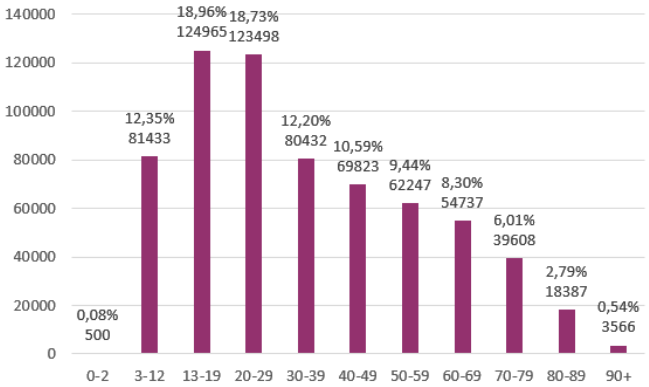


Figure 10: LIBIS: distribution of age groups of users who have ordered at least one book

2.2. METIS data

During the modernization project, METIS scope introduced reader feedback functionalities on bibliographic records, which are depicted in Figure 11, whereas a more detailed model is provided in Appx. no. 2. However, as shown in Table 5 the amount of information is insufficient for training any valuable recommender system. On the other hand, implicit feedback in LIBIS is abundant as presented in the previous section. In the period from June 2021 to January 2022 METIS collected 1,744,245 orders whereas collected 1209 ratings.

In addition, METIS does not have a timestamp attribute on ratings table, which is critical when evaluating the recommender system. For these reasons, METIS data is completely excluded from further research.

Table 5: Available feedback data and their counts from METIS production environments. Hence, feedback features in METIS were available from June 2021. Thus the data amounts in this table represent data of approximately 6 months.

Available data from LIBIS and METIS (ibiblioteka.lt)	
Data & Description	**Count**
Ratings. 5-star ratings. From May 2021 to January 22 (6 months period) only 1209 ratings have been collected, thus using explicit rating feedback for the training of recommendation system is not an option. Thus only implicit feedback systems may only be considered.	1209
Comments. From May 2021 to January 22 (6 months period) only 45 comments have been collected, thus no ML processes could be trained. In the future, if the situation changes, one could attempt to process comments with natural language processing (NLP) ML-based algorithms to extract positive/negative feedback from comments, which could then be used for RS-training, or other purposes.	45
Personal shelves. Users can add bibliographic records to personal shelves. By default user is provided with the following shelves: “I plan to read”, “Would recommend”, “Currently being read”, “Finished reading”. Users manage their shelves themselves and may add additional shelves.	80K
Saved searches. Users may save searches.	385

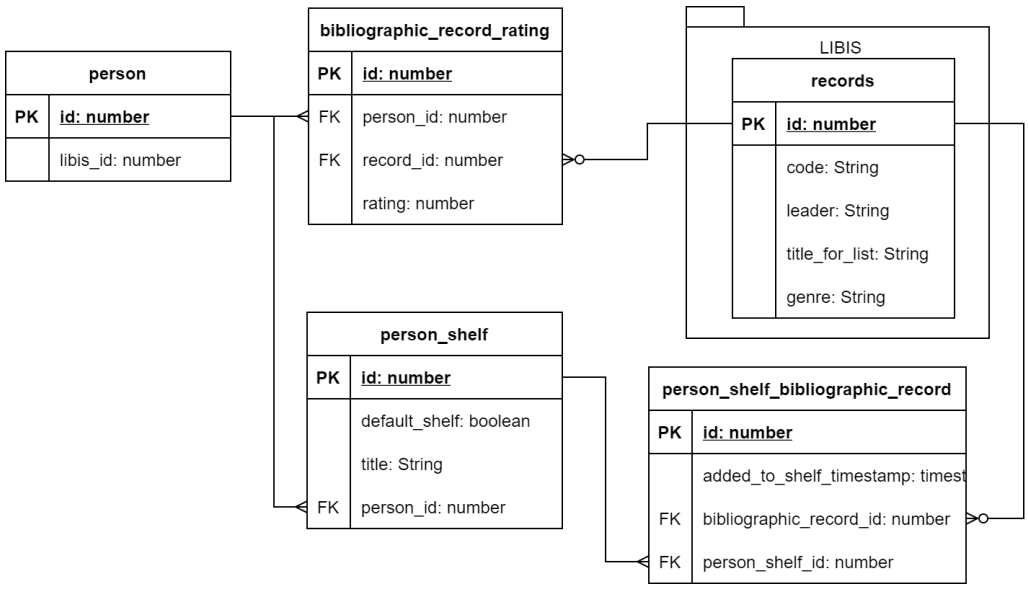


Figure 11: De-identified dataset for developing recommender system. Hence, comments and saved searches are omitted from the dataset because their use in developing RS would significantly inflate the scope of this research.

2.3. Privacy protection measures

As described in the original data structure section, the data contains a variety of sensitive, that would cause extreme privacy leaks if accessed by an adversary. Thus, privacy protection measures must be applied.

Sensitivity was assessed according to direct-identifiers and quasi-identifiers. Appx. no. 1 provides full database structure with the assessment of sensitivity and usefulness. Hence, quasi-identifiers are pieces of information that are not of themselves unique identifiers but are sufficiently well correlated with an entity that they can be combined with other quasi-identifiers to create a unique identifier. Also, the data was selected similarly to other ML datasets for recommendations, particularly the MovieLens dataset [HK15].

The main principles used to protect privacy:

1. Data-minimization principle. Selecting only the data that is required for the goal.
2. Statistical bucketing. Bucketing generalizes the values into broader categories which mitigates re-identification by quasi-identifiers.
3. Psuedo-id generation. Hiding the original id, to mitigate de-anonymization in case the original user list is accessed by an adversary.

2.4. UNIMARC-based feature engineering

This section describes features that are engineered using UNIMARC domain knowledge.

2.4.1. Record genres (675$a)

Record genres are highly beneficial when developing a content-based filtering model, which is typically used in a hybrid model alongside collaborative filtering mode. Genres are calculated

according to numeric codes in unimarc subfield 675$a [Wol10], which stands for Universal Decimal Classification (UDC) code. For instance, UDC code 821.172-84 is translated to the genre “Grožinė literatūra” based on regexp ^8.∗. Mappings between UDC codes and genres are provided in Appx no. 3. UDC along with record ids are exported from Solr due to limitations described in Original data structure. Solr query for genres is depicted in Figure 12.

```
q: type_s:code AND field_code_s:675 AND sf_code_s:a AND value:[* TO *]
fl: id, value
```

Figure 12: LIBIS Solr: query for release date which is under subfield 100$a Position 9.

As depicted in Figure 13, there is total of 62 genres with a moderate imbalance in distribution. One could perform a sanity check, on how would the recommender system behave if a user orders only books of a narrow niche genre. A record may also have multiple genres, although 75% of records have 1 genre, see Figure 16.

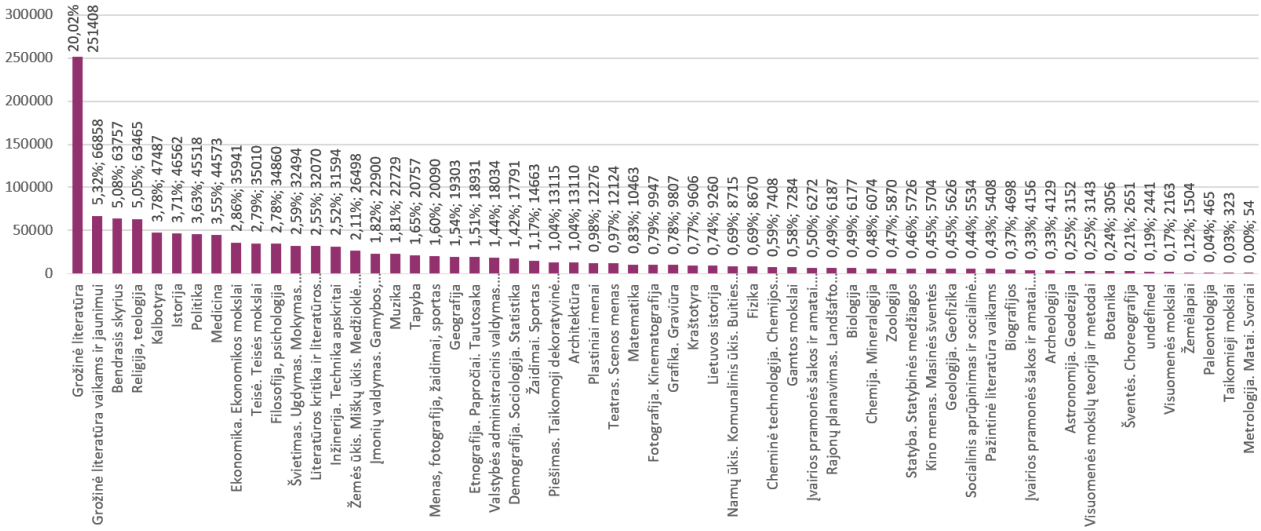


Figure 13: LIBIS: distribution of genres

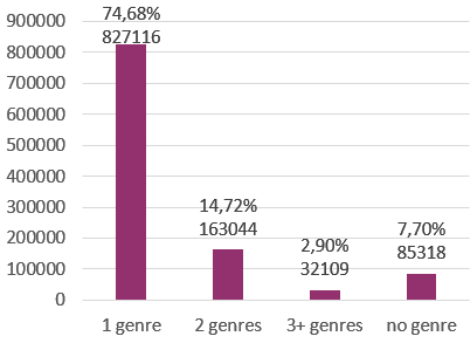


Figure 14: LIBIS: distribution of genres per book

2.4.2. Release date (100$aPos9)

Record release data is not as significant attribute as record genre, although it could further improve the Content-based filtering model, which in turn would improve hybrid RS performance. According to UNIMARC standard, resource language is stored in code at position 9 of subfield $a under field 100 [Wol10]. Solr query for genres is depicted in Figure 15.

```
q: type_s:code AND field_code_s:100 AND sf_code_s:a position_i:9 AND value:[0 TO 2100]
fl: id, value
```

Figure 15: LIBIS Solr: query for release date which is under subfield 100$a Position 9.

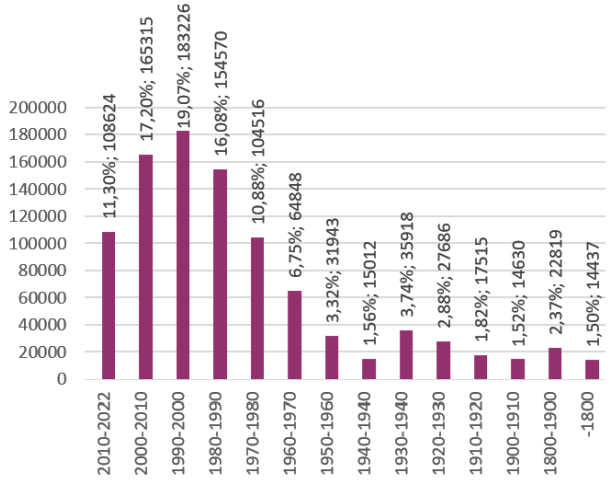


Figure 16: LIBIS: distribution genres per book

2.4.3. Resource language (101$aPos0)

Similar to the release data attribute, this attribute could further improve the Content-based filtering model, which in turn would improve hybrid RS performance. According to UNIMARC standard, resource language is stored in code at position 0 of subfield $a under field 101 [Wol10]. Solr query for languages is depicted in Figure 17. This feature raises a pragmatic test case for RS, suppose a user has been ordering books solely in the Russian language, intuitively RS should recommend Russian books. Language distribution across books is shown in Figure 18. Language count per book is depicted in Figure 19. Books that have 2 languages constitute over 5% of all books, thus one should not drop other than first language when training a model.

```
q: type_s:code AND field_code_s:101 AND sf_code_s:a position_i:0 AND value:[* TO *]
fl: id, value
```

Figure 17: LIBIS Solr: query for release date which is under subfield 101$a Position 0.

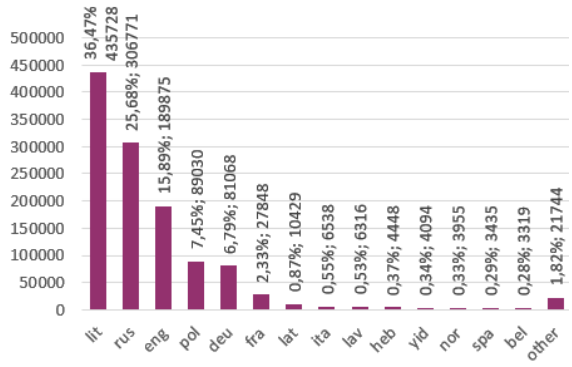


Figure 18: LIBIS: language distribution across books

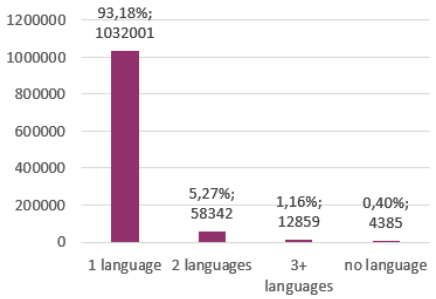


Figure 19: LIBIS: language count per book

2.5. Datasets

As described in the previous section, METIS data does not provide enough ratings to be used for RS training, thus only LIBIS data will be used to produce the datasets. The data model of the datasets is visualized as entity-relationship diagram in Figure 20. Summary statistics for the LIBIS datasets are shown in Table 6. libis-100k dataset is available at kaggle.com/senvaitis/libis100k.

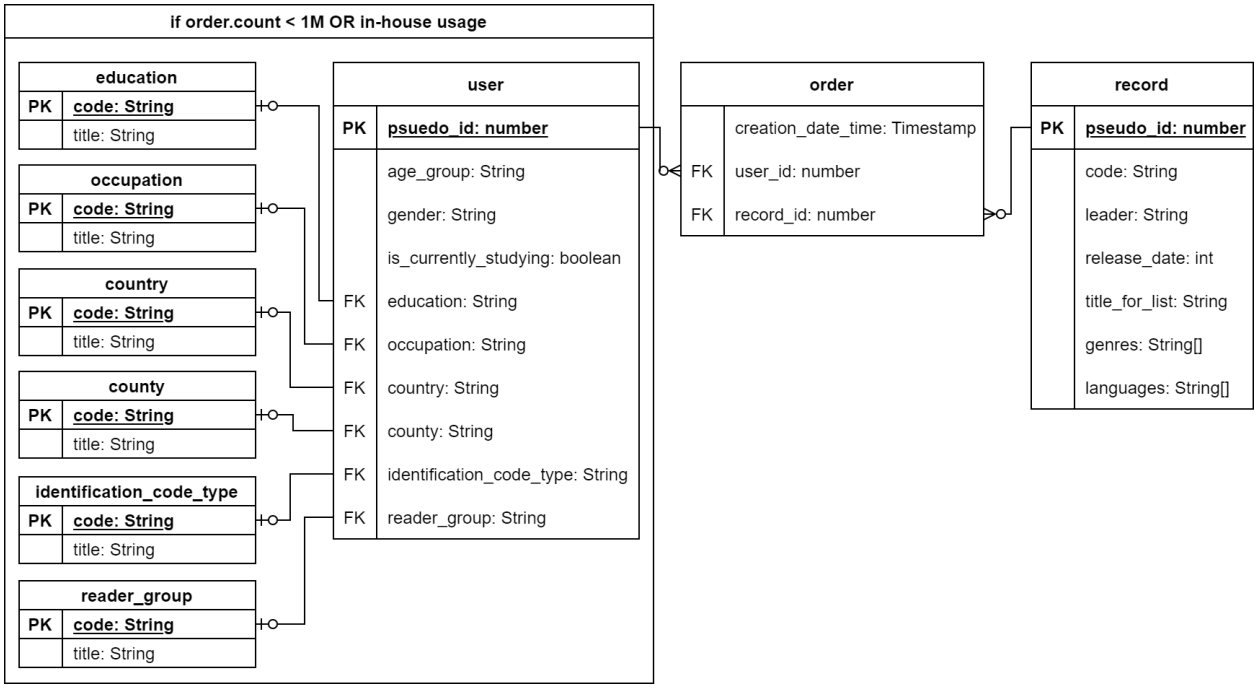


Figure 20: De-identified dataset for developing the recommender system.

LIBIS datasets share several characteristics. They each represents order tuples of the form <user, record, timestamp>. Orders and other data are attributed to anonymized user ids (these ids do not map across datasets). Records are listed along with their titles, leaders from LIBIS/METIS systems, along with zero or more genres. Only users with at least 20 ratings are included (with an exception for libis-1m-dense dataset).

These datasets differ in their sampling methods. Though all four required a minimum of 20 ratings to include a user, 100k also required complete demographic data. In 1m dataset sampled

users required all demographic data except for reader_group (due to scarcity as depicted in Figure 9). 10m and 20m users with the latest activity regardless of their demographic data completeness.

Table 6: Privacy preserving libis datasets

Name	Filters	Users	Records	Orders	Density
libis-100K	20≤orders per user≤500	1130	2000	100,319	4,44%
libis-1m	20≤orders per user≤500	12930	9000	1,000,582	0,86%
libis-5m	20≤orders per user≤500	67200	12000	5,004,161	0,62%
libis-10m	20≤orders per user≤500	131,500	17,000	10,011,681	0,45%
libis-20m	20≤orders per user≤500	256,250	60,000	20,018,036	0,13%

2.6. Dataset extraction

The process of dataset extraction is shown in Figure 21. The main difficulties of the process were:

1. Querying LIBIS PROD database for such large amounts of data is unwanted. Thus, raw data was exported to CSV files which were then loaded to local database for composition of various dataset.
2. When querying LIBIS PROD database data-minimization and statistical bucketing principles were applied to prevent storage of highly sensitive information on local filesystem.
3. Record genres are calculated according to numeric codes in unimarc subfield 675$a of bibliographic record. As record unimarc properties are stored as JSON in database, selection of such value is impractical due to sophisticated JSON parsing and parsing performance. Solution was found to extract 675$a from Solr indexes, which took only a few seconds and was easily queried.
4. As this research is focusing on recommendation of books (records with “am0” in leader), libis_raw_am0 schema was made to only contain orders, users, and record that are related to books. This resulted in faster querying performance and correctness, because various joins could be ommited.
5. Datasets were prapared in such a way, that thay consist of the most recent data in order to ensure realistic environment of training.
6. De-identified user information was omitted in datasets which have more than 1M orders, because large dataset containing quasi-identifiers is still vulnerable to de-anonymization attacks. Same has been done in MovieLens dataset [HK15].
7. Psuedo-id generation not only improved data protection, but also highly reduced the sizes of dataset CSV files.
8. Dependence on order time is inevitable because time allows for deterministic results. In case of MovieLens, dataset was exported from selected date, whereas in LIBIS dataset case orders were taken from most recently active users. The purpose of such approach, was to catch the newest book reading trends. If one was to pick random users, then resulting order dataset size would vary, thus achieving even close to selected dataset

size is hard and requires multiple runs. Also, if dataset is regenerated differently each time, it makes data debugging complex.

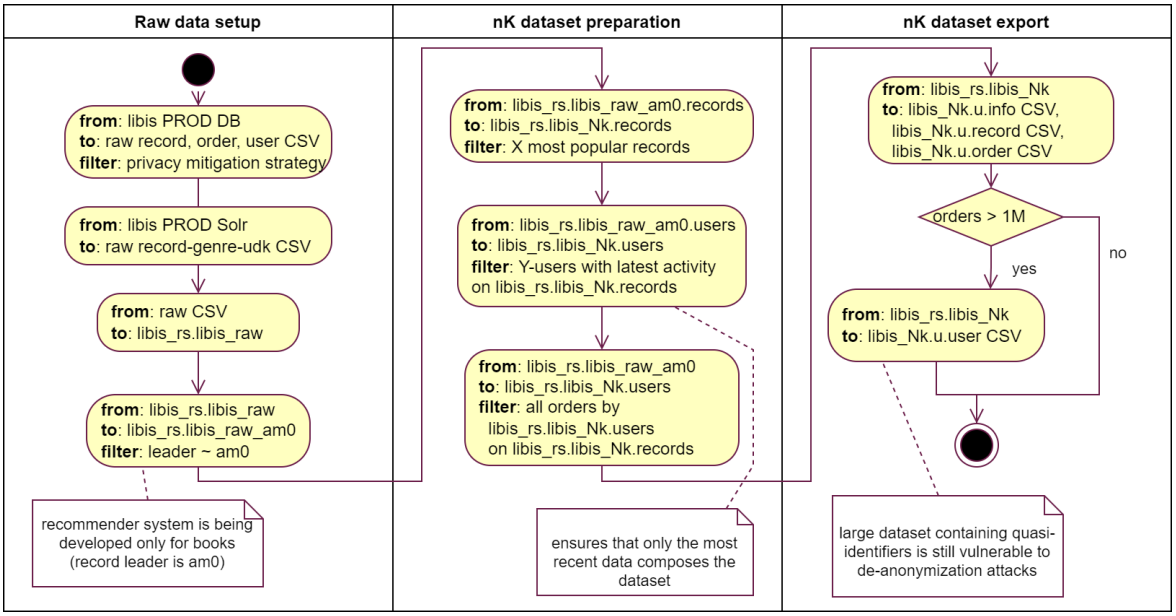


Figure 21: De-identified dataset for developing recommender system

3. Recommender System Research & Development

This chapter describes conducted experiments and presents LIBIS recommender system prototypes. Throughout further sections there are links to source code files, hence links are also provided concisely in Source Code section at the end of the document. Experiments were conducted using Jupyter Notebooks, thus execution results are visible and also variety of examples are provided.

Experiments were conducted on a desktop PC with: Intel i7-6700k CPU, NVidia GTX 1070 GPU, and 32GB RAM. However, insufficient RAM size slowed down process of training with large datasets.

3.1. Demographics Filtering

This section describes demographics based recommender system which was built using k-means algorithm which is a machine-learning based clustering algorithm.

This technique leverages attributes of users and orders of users. K-means algorithm assigns users to clusters by learning interactions between attributes of users and items which they interact with. By leveraging explicit user attributes this technique allows to resolve new user cold-start problem.

In the prototype version (provided at gitlab.com/senvaitis/libis-rs/-/blob/main/libis-rs-df-kmeans-clustering.ipynb), age_group, education and gender were used as user attributes. Typically optimal number of clusters would be determined using Elbow method, where optimal number for all libis datasets would be 5 or 6, as shown in Figure 22. However, such low number of clusters is over-generalizes and does not match realistic groups of users. Thus, in LIBIS case, on datasets

over 1m, number of clusters should far exceed optimal number of clusters. Practical experiments showed that cluster number of 16 stops assigning children of adjacent age groups into same groups, thus making more appropriate recommendations.

Clustering accuracy has been measured using Mean Absolute Error (MAE) which calculates the average absolute deviation between the actual and the predicted order count scores. A lower MAE represent good recommendation quality. The formula of MAE refer to Equation 1, where N is the total of predicted records, r_{xp} and $\hat{r}_{xp}$ denote the actual and predicted order counts of the user x to record p, respectively.

$$MAE = \frac{1}{N} \sum_{x \epsilon U, p \epsilon P} |\hat{r}_{xp} - r_{xp}| \tag{1}$$

Size of the dataset has effect on MAE as shown in Figure 23, however recommender system must provide coverage for all users, thus there was little that could be done. On the other hand, as LIBIS datasets are of implicit feedback, number of negative samples has significant effect on MAE as shown in Figure 24. Yet negative sampling samples is a trade-off between accuracy and computing resource usage. For instance, 20m dataset with 4 negatives results in 100m dataset, resulting in roughly 20GB memory usage while training K-means model.

Initially, statistical buckets for young people ages were wide, namely 0-2, 3-12 and 13-19 years. However, with such groups K-means algorithm was failing to provide adequate recommendations for children, especially teenagers. MAE of all clusters was 0.25 for libis-20m with 4 negatives. For example, teenager of age 15 would be recommended book “Kakė makė”, which, according to book publishers, is recommended for 6 year olds. This forced to rework statistical buckets more precisely. Based on [CDC], buckets of children ages were adjusted to 0-2, 3-5, 6-8, 9-11, 12-14, 15-17. This resulted in more reasonable recommendations.

Even after adjustment of statistical buckets, recommendations for child groups were poor. This, however, was resolved by taking only recent book orders for training, because we should not recommend books that a child has grown out of. This further reduced MAE as shown in Figure 25, hence longer order recency results in higher MAE.

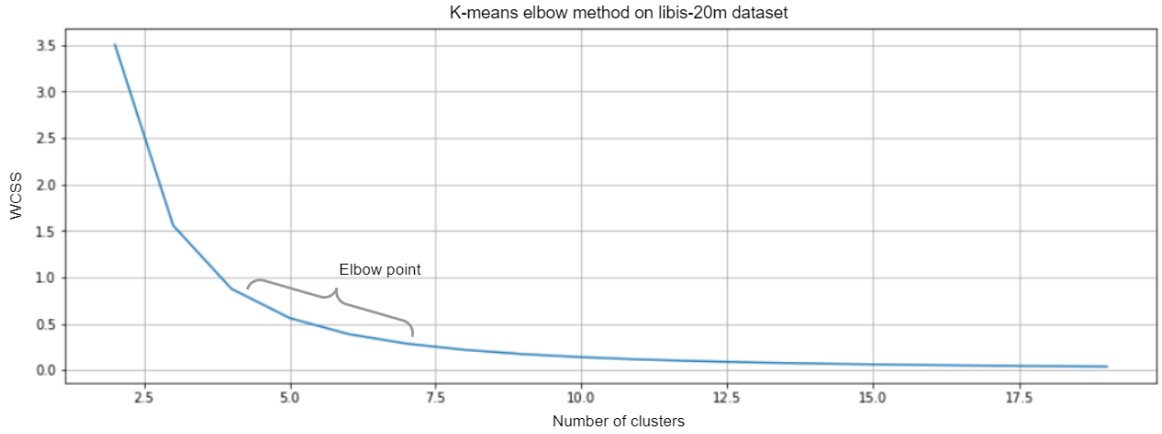


Figure 22: Elbow method graph to determine optimal number of clusters

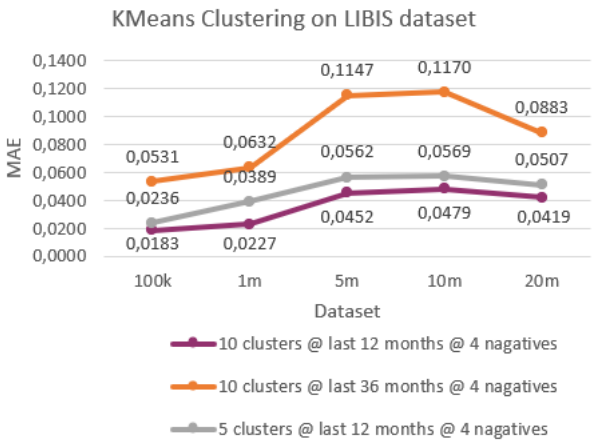


Figure 23: Mean Average Error by dataset size

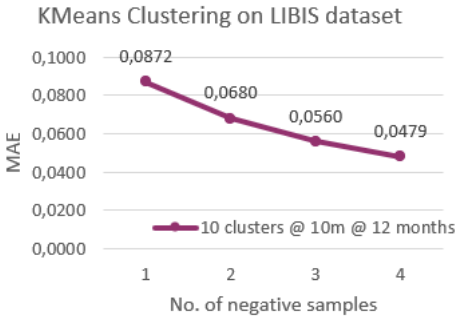


Figure 24: Mean Average Error by number of negative samples

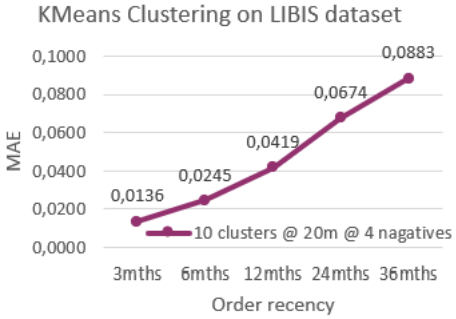


Figure 25: Mean Average Error by order recency

3.2. Collaborative Filtering

This chapter aims to evaluate the suitability of LIBIS datasets for the development of the recommender system. As our dataset is implicit and does not have a robust confidence metric, algorithm implementations and available examples are limited as the majority focuses on datasets with explicit feedback or at least a strong confidence metric (inferred feedback).

Different datasets yield different results, for example, He et al. [HLZ+17] achieved up to 20% higher HR@10 on Pinterest dataset compared to MovieLens dataset. The further sections describe experiments performed on libis datasets with multiple algorithms and their variations.

Experiments with CF were also run on Kaggle's Jupyter Notebook with NVIDIA TESLA P100 GPUs support, however no major training time difference was noticed when comparing the aforementioned environments, most likely due to the high demand for GPUs at Kaggle.

Hence, the current state of this chapter only describes research on Collaborative Filtering, thus such algorithms would severely suffer from a cold-start problem for both new users and new items. In the future, algorithms of Content-based Filtering will be implemented and evaluated and joined to the CF model to form a hybrid recommender system. As a result, contextual information about user or record is absent in training.

3.2.1. Train/test split

Unlike in regular train/test splitting, the data in RS training is split according to the time factor using the leave-one-out technique. Based on the timestamp column, the very last feedback of each user composes the test dataset, while the rest will be used as training data.

Doing a random split would not be fair, as it could potentially use a user's recent orders for training and orders reviews for testing. This introduces data leakage with a look-ahead bias, and the performance of the trained model would not be generalizable to real-world performance.

3.2.2. Evaluation

The predominantly used metrics in evaluating recommender performance are Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG), especially HR@10 and NDCG@10 variants [HLZ+17; PE21; RKZ+20]. These metrics fall under accuracy evaluation factors.

Hit Ratio (HR). Taking HR@10 for instance, for each test sample, random 99 items are selected which user has not interacted with. Then a ground truth item is added to the list, resulting in 100 items. Lastly, RS model is used to generate ranking scores between 0 and 1. Top-k items (in this case 10 items) are selected. If the ground truth item is included in top-k, then it is a success. Finally, HR is calculated as a mean of successes and failures [LLJ+20].

Normalized Discounted Cumulative Gain (nDCG). NDCG solves the shortcoming of the Hit Ratio metric where the position of the ground truth item in top-k ranking is not measured [WWL+13]. The equation is derived from metrics Cumulative Gain (CG) and Discounted Cumulative Gain DCC. CG (see Equation 2) is the sum of the graded relevance values of all results in a search result list, where rel_i is the graded relevance of the result at position i. DCC (see Equation 3) ensures that highly relevant documents appearing lower in a search result list should be penalized as the graded relevance value is reduced logarithmically proportional to the position of the result. Finally, nDCG (see Equation 3) is computed by dividing DCG by ideal DCG. In case of implicit datasets (where graded relevance can only be 0 or 1), nDCG equation simplifies as depicted in Equation 5.

$$CG_p = \sum_{i=1}^{p} rel_i \qquad (2)$$

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{log_2(i+1)} \qquad (3)$$

$$nDCG_p = \frac{DCG_p}{IDCG_p} \qquad (4)$$

$$nDCG_p = \frac{1}{log_2(i_{gt}+1)} \qquad (5)$$

3.2.3. "Implicit" library

As a benchmark model training algorithms were used from a popular Python library "Implicit" [Benb], which as the name implies is for implicit datasets. This project provides fast Python implementations of several different popular recommendation algorithms: Alternating Least Squares,

Bayesian Personalized Ranking, Logistic Matrix Factorization, Even though the algorithms in this library are based on research papers from 2008-2014, this library (GitHub repo: [Bena]) is actively maintained, has over 2.5K stars and over 500 usages.

This paper performs experiments only on the ALS and BPR models because only these models support custom CUDA kernels - enabling fitting on compatible GPUs. However, libraries required for GPU training are only available in Linux operating systems. Furthermore, these algorithms require high amounts of memory, which limited conducting their evaluation on libis-20m dataset.

Experiments conducted with Implicit library are provided at: kaggle.com/senvaitis/libis-rs-als-bpr-gpu. Results are visualized in section 3.2.5 Comparison.

Alternating least squares Matrix Factorization with Alternating Least Squares was introduced by [HKV08]. The principle of ALS is to minimize the entire loss function at once, but, only optimizing half the parameters. As a result, if half your parameters are fixed, the optimization becomes an easy algebraic solution. After recomputing one half of the parameters, it recomputes other half and repeats. There is no gradient in the optimization step since each optimization problem is convex and does not need an approximate approach.

Bayesian Personalized Ranking Matrix Factorization with Bayesian Personalized Ranking was introduced by Rendle et al. [RFG+09]. This algorithm is considered standard and strong CF baseline for recommender systems even more than decade after [TTZ+19]. It models the user-item representation using the inner product and explores the triplet objective to rank items.

3.2.4. Neural Collaborative Filtering (NCF)

Neural Collaborative Filtering (NCF) is a neural network based method for collaborative filtering which models nonlinear user-item interaction. The framework was proposed in [HLZ+17]. Adapted model depicted in Figure 26 was implemented for libis RS on Pytorch Lightning framework. Experiments conducted with this model are provided at: gitlab.com/senvaitis/libis-rs/-/blob/main/libis-rs-cf-ncf-gpu.ipynb.

The inputs to the model are the one-hot encoded user and record vector, which are fed to the user embedding and record embedding respectively, which results in a smaller, denser user and record vectors. The embedded user and record vectors are concatenated before passing through a series of fully connected layers, which maps the concatenated embeddings into a prediction vector as output. Finally, we apply a Sigmoid function to obtain the most probable class.

Unlike algorithms from "Implicit" library, training of neural network for implicit data requires negative sample. Negative samples were marked with label 0, whereas positive samples (orders from dataset) were grouped and summed by repeated read counts. For instance, if a reader ordered book once, label is 1, where ordering it twice would result to 2. This is considered as confidence metric, however experiments showed that using only 0 or 1 for labels yields very close results, which differentiated only by 1% in best case.

NCF model has been trained for 5 epochs using the GPU on each dataset and their evaluation depicted in Figures 28, 29. Running more epoch is unlikely to improve evaluation results as training loss step curve is beginning to straighten as depicted in Figure 27.

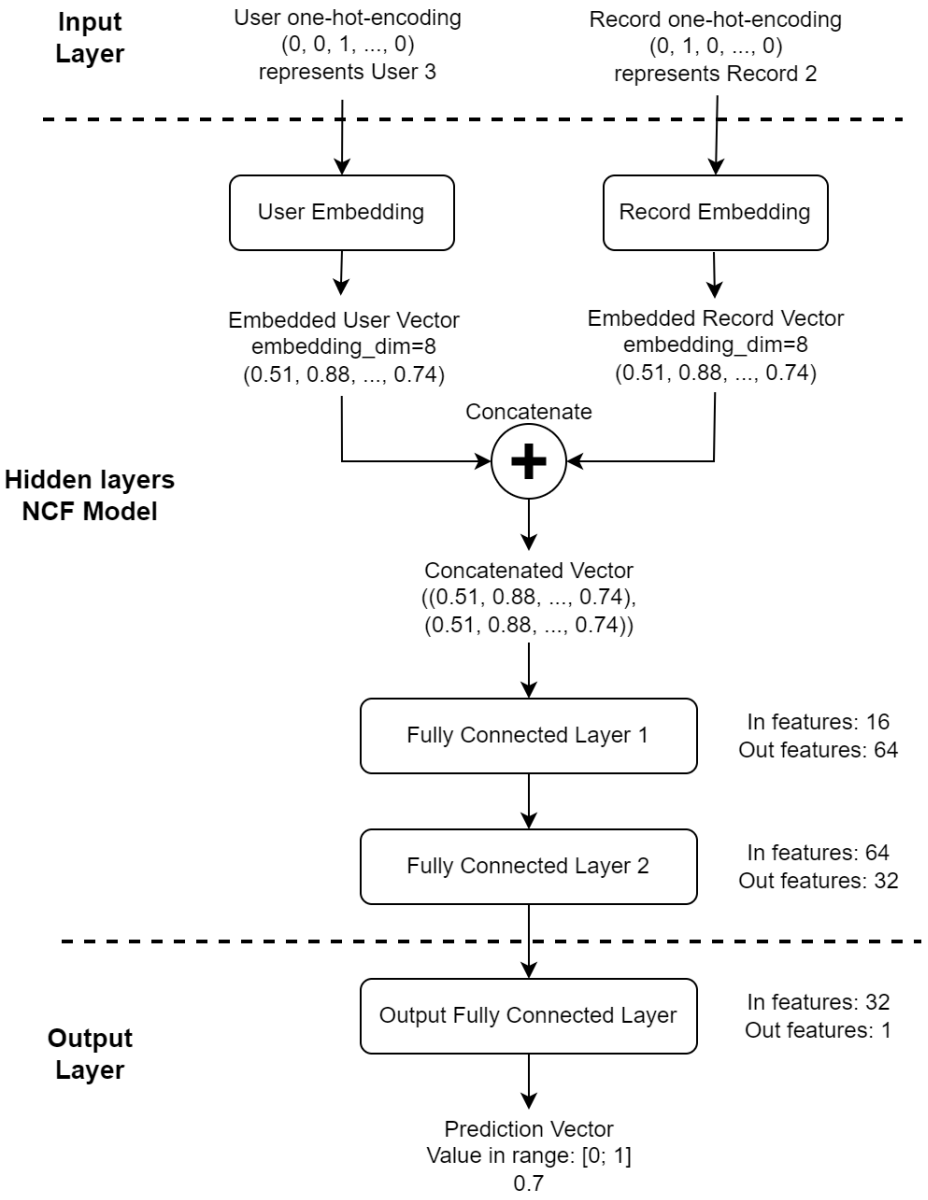


Figure 26: Neural network model of Neural Collaborative Filtering algorithm

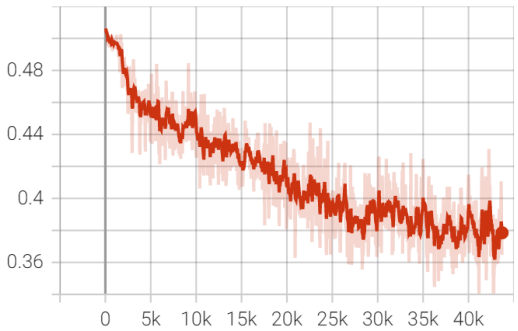


Figure 27: NCF: loss step of training model on libis-20m dataset

3.2.5. Comparison

In the context of recommendation systems, results vary highly from dataset to dataset. For instance, Tran et al. [TTZ+19] were able to achieve 0.87 HR@10 and 0.64 nDCG@10 on MovieLens dataset

with their state-of-the-art algorithm, but in other dataset, were able to reach as low as 0.27 HR@10. Other algorithm also face it [HYC+17; TAC18]. From their results, and results in this paper, trend is visible that recommendation systems are highly dependent on dataset size.

With HR@10 of 0.66 and nDCG@10 of 0.38 (see Figures 28 29), results from NCF algorithm on libis-20m dataset are promising as they are close to the results on MovieLens dataset which is considered industry benchmarking dataset.

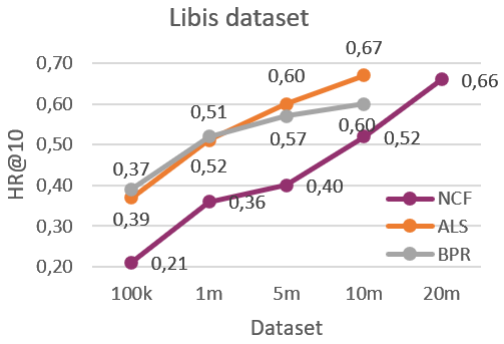


Figure 28: HR@10 in relation to dataset size

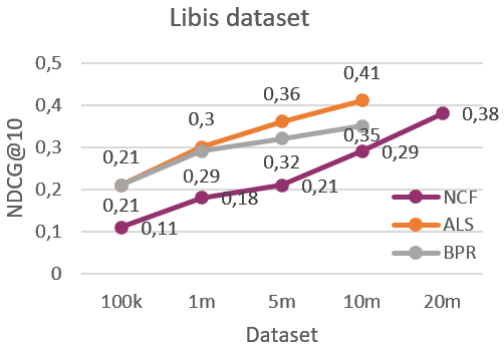


Figure 29: nDCG@10 in relation to dataset size

3.3. Content-based Filtering

This section describes content-based filtering recommender system which was built using TF-IDF (term frequency-inverse document frequency) machine learning-based information retrieval algorithm.

TF-IDF works by increasing proportionally to the number of times a word appears in a document, but is offset by the number of documents that contain the word. So, words that are common in every document, rank low even though they may appear many times, since they do not mean much to that document in particular.

In the prototype version (provided at gitlab.com/senvaitis/libis-rs/-/blob/main/libis-rs-cbf-tf-idf.ipynb), TF-IDF was adapted to use record genres instead of words. Cosine similarity matrix was used to calculate similarities between records. The most frequent genres are 'Grožinė literatūra', 'Grožinė literatūra vaikams ir jaunimui' and 'Filosofija, psichologija', as depicted in Figure 30. Then there are less frequent genres, such as 'Medicina', 'Politika' or 'Geografija'. As previously pointed out, less popular genres should have highest leverage when recommending.

As an example let's consider new user wants to find records similar to "Dievų miškas", which is a mixture of 'Istorija|Valstybės administracinis valdymas. Karyba|Grožinė literatūra' genres. Say now that among the available records to recommend we have a collection of 'Grožinė literatūra' records and some other 'Istorija' records too. In this case, it seems reasonable to assume that the less generic genre, 'Valstybės administracinis valdymas. Karyba', will be the most relevant in terms of characterising the record.

Figure 30: Popularity of genres in libis-20m visualized as word cloud

3.4. Hybridization

The main purpose of hybridization is to overcome short-comings of selected recommender techniques. Typically single technique inherits one or multiple of short-comings: new user cold-start issue, new item cold-start (coverage), lack of diversity and novelty, absence of serendipity. This section introduces solution of pipelined hybrid and describes attempts to leverage Wide & Deep learning technique.

3.4.1. Pipelined hybrid

In the prototype version (provided at gitlab.com/senvaitis/libis-rs/-/blob/main/libis-rs-hybrid.ipynb), three recommender systems were joined in Switching and Mixed type of hybridization.

This hybrid prototype joins Collaborative filtering, Demographics filtering, and Content-based filtering recommenders, respectively Neural Collaborative Filtering (NCF), K-means clustering and TF-IDF machine learning algorithms.

Switching hybridization is responsible to enabling or disabling Collaborative filtering recommender and Content-based filtering. CF recommender is disabled if user has less than 5 orders in his order history as CF provide poor recommendations on cold-start. Similarly, CBF recommender cannot recommend without knowing what content user likes. Thus, on cold-start recommendations are provided based on demographic information of the user using Demographic filtering based on K-means clustering.

Mixed hybridization mixes recommendations that came from separate recommenders. If user has long order history, then combined result recommendations primarily consist of recommendations provided by collaborative filtering as this recommender is considered in industry to provide most valuable recommendations.

Collaborative filtering robustly ensures diversity, novelty, serendipity, however it inherits new user cold-start and new item cold-start issues. Meanwhile, Demographics filtering resolves new user cold-start, provides mediocre diversity, novelty and serendipity. Lastly, Content-based filtering

resolves new item cold-start, but fails at new user cold-start and at providing diversity, novelty and serendipity. These techniques, if joined correctly cancel out short-comings of each.

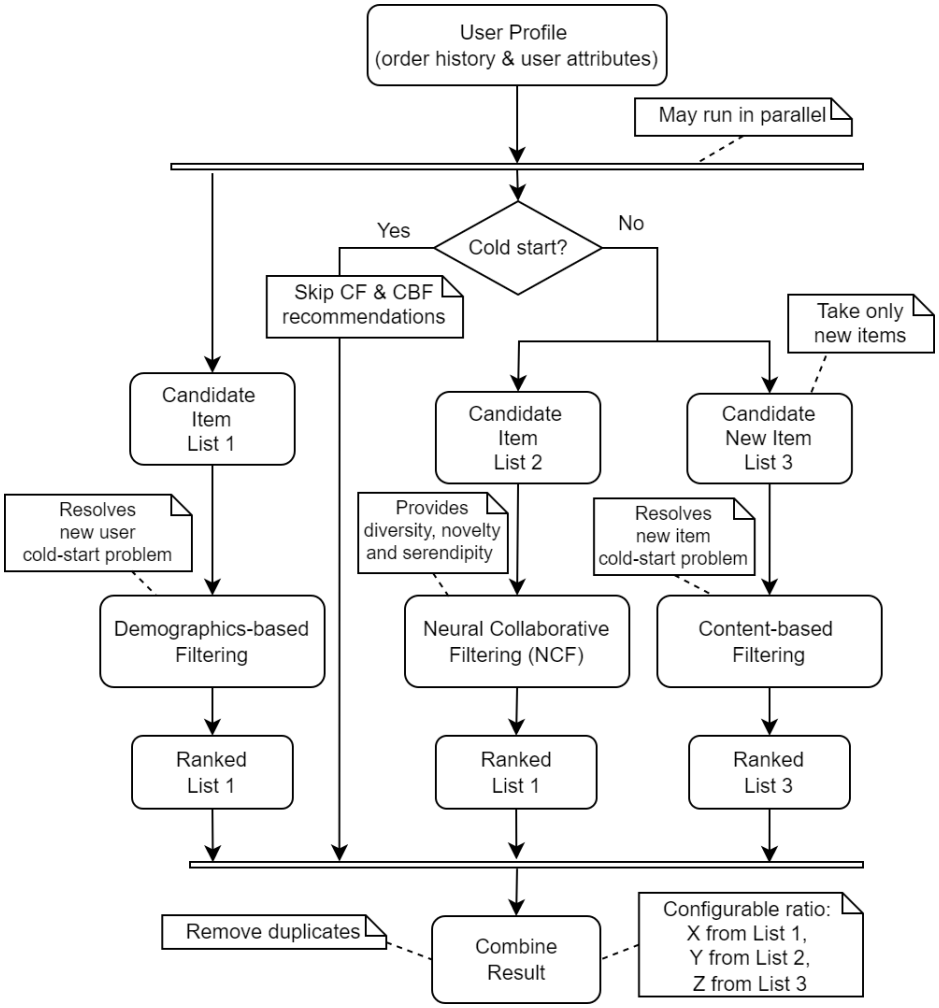


Figure 31: Pipelined hybrid solution: combination of Switching and Mixed types of hybridization with focus on new user and new item cold-start issues

3.4.2. Wide & Deep learning

Wide & Deep learning technique for recommender systems was presented by Google [CKH+16] and has been productionized and evaluated on Google Play, a commercial mobile app store with over one billion active users and over one million apps. It jointly trains wide linear models and deep neural networks - to combine the benefits of memorization and generalization for recommender systems as shown in Figure 32. This technique aims to tackle both new user and new item cold-start problems.

Memorization of feature interactions through a wide set of cross-product feature transformations are effective and interpretable, while generalization requires more feature engineering effort. With less feature engineering, deep neural networks can generalize better to unseen feature combinations through low-dimensional dense embeddings learned for the sparse features. However, deep neural networks with embeddings can over-generalize and recommend less relevant items when the user-item interactions are sparse and high-rank. In short, it is a deep learning algorithm that can

memorize feature interactions and generalize user features.

Wide & Deep learning framework combines strengths of both types of model and led to significant improvements on app acquisitions over wide-only and deep-only models. This algorithm is considered CF and CBF weighted hybrid however the weights are implicit and non configurable in this algorithm.

However, conducted experiments on Wide & Deep technique adapter for implicit feedback and libis datasets presented issues even with libis-100k dataset. Althrough prediction on a all user-item cross joined pairs is quick enough taking 5s, however prediction for single user on given item set is slow, for instance, prediction for 100 items for single user take between 500 and 1500ms. Furthermore, memory usage training on libis-1m exceeded 10GB, which would make it impractical to train on libis-20m dataset.

Due to slow training and slow predictions, this method was eliminated from further research as quick predictions are required for Hit Ratio evaluation. In addition, nDCG@10 was successfully evaluated on libis-100k dataset, however it performed nearly twice worse than NCF. Most likely Wide & Deep technique is built on the basis of explicit feedback, whereas libis datasets only provide implicit feedback.

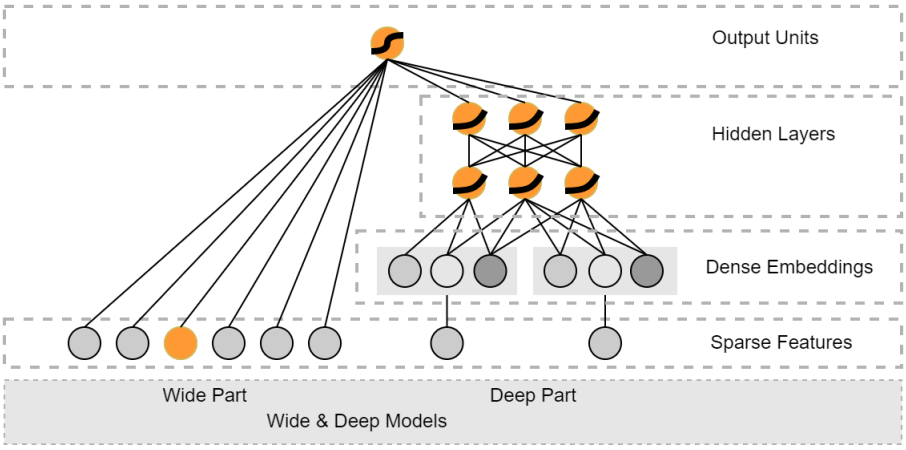


Figure 32: Wide & Deep Learning in Recommender Systems [CKH+16]

4. Production Use

Much of the research (outside of the few tech giants) cited in this paper, remains just that - research, and not part of a production application. Deploying and utilizing machine learning models in production remains a significant challenge. This chapter seeks to explore possibilities and challenges of deploying and maintaining deep-learning based recommendation system in production environment.

4.1. Use cases

LIBIS recommender system (further LIBIS-RecSys) could serve in both LIBIS and METIS projects. The main objectives are: 1) provide assistance for librarian to provide recommendations

for readers 2) to provide automated recommendation in self-service portal (METIS) in similar fashion to Youtube, Netflix, etc. Uses cases in LIBIS ecosystem are depicted in Figure 33.

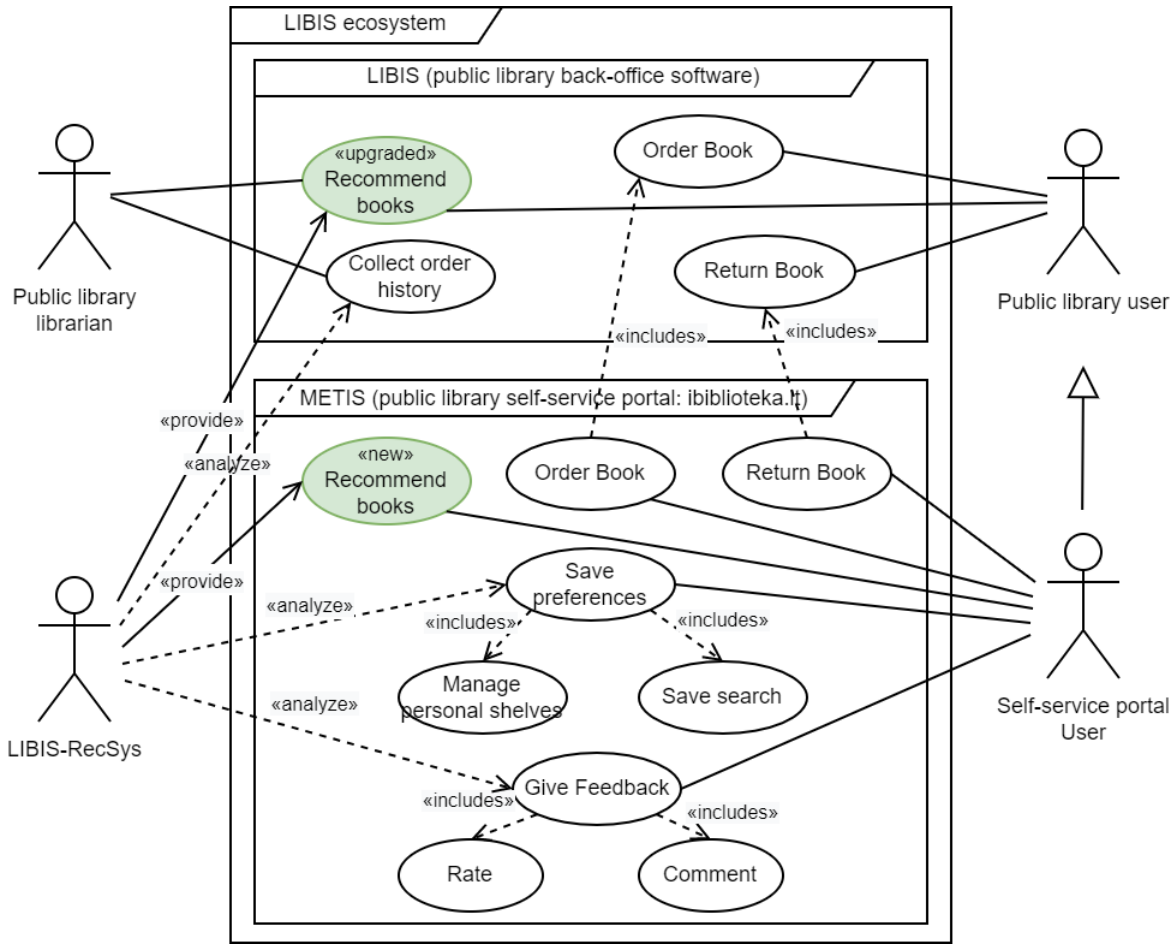


Figure 33: Use cases in LIBIS ecosystem in relationship to recommender system

4.2. Integration

This section describes one way of integration of LIBIS Recommendation System (further LIBIS-RecSys) within the ecosystem of LIBIS as seen in Figure 34. Prototype version of LIBIS-RecSys web service provided at gitlab.com/senvaitis/libis-rs/-/blob/main/libis-rs-server.ipynb.

LIBIS-APP is monolith application based on Java, developed with SpringBoot framework. It exposes multiple API endpoints for front-end application LIBIS-NGX and for other systems in the ecosystem.

The main concerns to recommender system integration:

1. LIBIS-RecSys should not have direct access to sensitive information on LIBIS-DB, thus should read sanitized data through intermediary dataset preparator service (further, LIBIS-RecSys-DataPreparator). As a result, this would allow outsourcing responsibility of LIBIS-RecSys away from LIBIS development team. LIBIS-RecSys-DataPreparator should provide API to access on processed data whose attributes should be carefully coordinated with Lithuanian Public Library (LNB).
2. Lowest possible LIBIS DB performance degradation. LIBIS-RecSys-DataPreparator

should be allowed read-only statements on LIBIS DB and store its data in its own database. In turn, this would prevent accidental data corruption.

3. Allow LIBIS-RecSys-DataPreparator direct data retrieval from LIBIS DB. Practical experiments showed that selecting data from LIBIS DB for RS training takes hours, thus data should be allowed to be read directly from DB.
4. Allow METIS data usage in the future. LIBIS-RecSys should be prepared to include METIS data, for example, "Not interested" flags, thus it should also be prepared to have read access to METIS DB.
5. LIBIS-RecSys-DataPreparator should be able to return time-framed data so that LIBIS-RecSys could do retraining on model on demand.
6. APIs should be strictly accessable only for coordinated applications using host whitelisting:

The following subsections describe APIs that LIBIS-RecSys-DataPreparator and LIBIS-RecSys-DataPreparator should expose. Sequential interactions between LIBIS/METIS systems, LIBIS-RecSys and LIBIS-RecSysDataPreparator are shown in Figure 35. Hence refreshing of dataset and retraining of model could be run as scheduled tasks.

4.2.1. LIBIS-RecSys-DataPreparator APIs

GET `/dataset-status`

This API is responsible for reporting current status of the dataset, including date of the currently available dataset and status of dataset refreshment (possible states: OK, IN_PROGRESS, FAILED).

POST `/refresh-dataset`

This API initializes refreshment of dataset. Status returned from `/dataset-status` changes to IN_PROGRESS, on error changes to FAILED, and on successful refreshment to OK.

GET `/dataset/{dtFrom}/{dtTill}`

This API is responsible for returning processed data based on given time frames. As time frames can be long and dataset size be large, it may require solution for large-size data transfer.

4.2.2. LIBIS-RecSys APIs

GET `/model-status/`

This API is responsible for reporting current status of the RS model, including date of dataset on which it was trained and status of model retraining (possible states: OK, IN_PROGRESS, FAILED).

POST `/retrain-for-all-users/`

This API is responsible for retraining ML model. The metadata of model should store date until which order history entries have been used while training. With such attribute, it could retrain model, for example, every midnight with day's order history. On call, status returned from `/model-status` changes to IN_PROGRESS, on error changes to FAILED, and on successful model retraining to OK.

POST `/generate-recommendations-for-all-users/`

This API would pre-generate and store top N candidate recommendations for all users in order to provide quick recommendations as real-time predictions may slow down user experience.

POST `/generate-recommendations-for-user/{userId}`

This API would regenerate recommendations for user. As each regeneration provides different recommendations, it could be employed as refresh button. It is worth noting, that with every added item to user's history we could retrain ML model, however it is not as straightforward to mark which history items were used in training.

GET `/get-recommendations-for-user/{userId}/{count}`

This API would load and return specified number of recommendations from top N candidates for specified user.

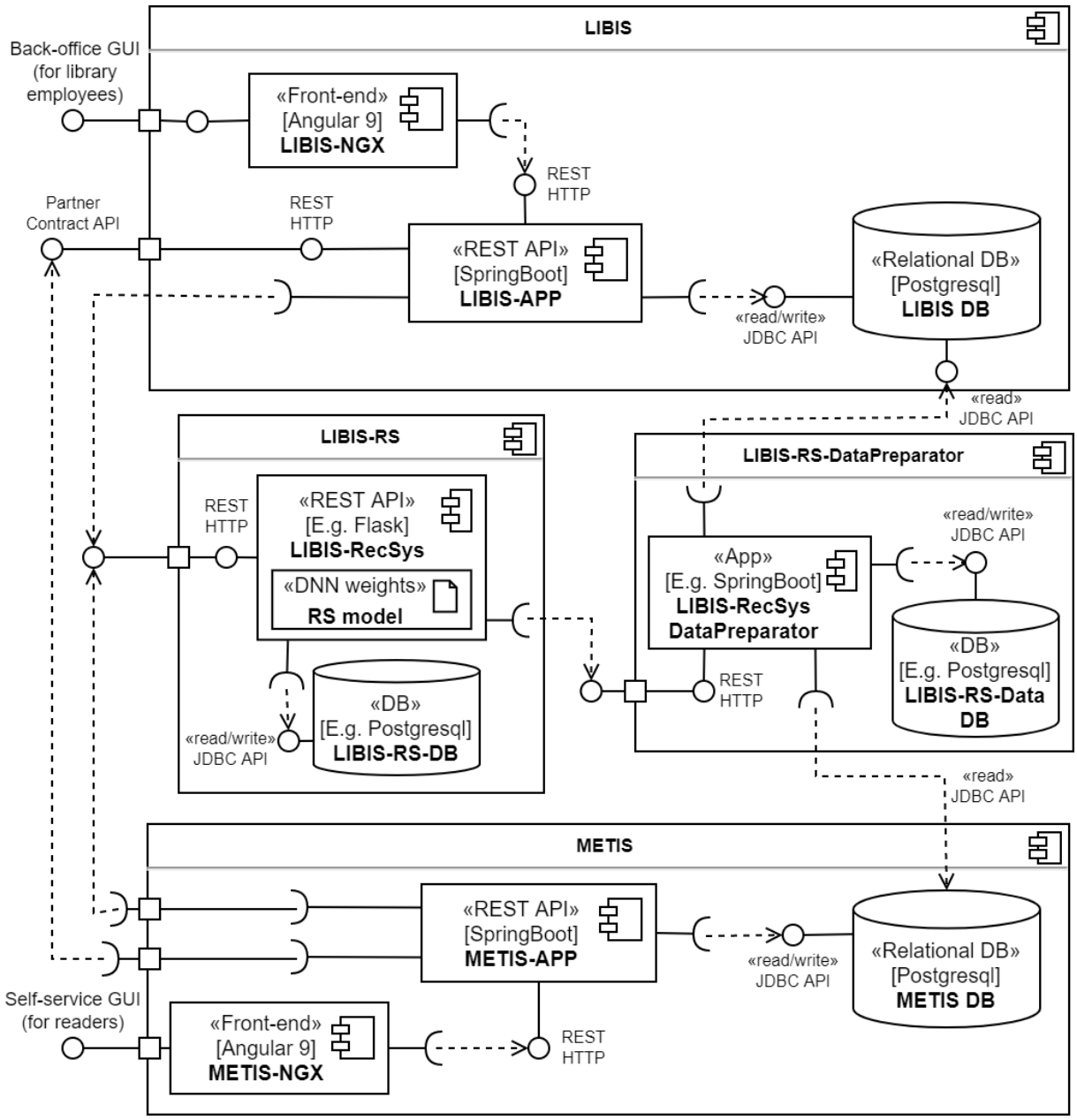


Figure 34: Component diagram of integration of LIBIS-RecSys and LIBIS-RecSes-DataPreparator into LIBIS ecosystem

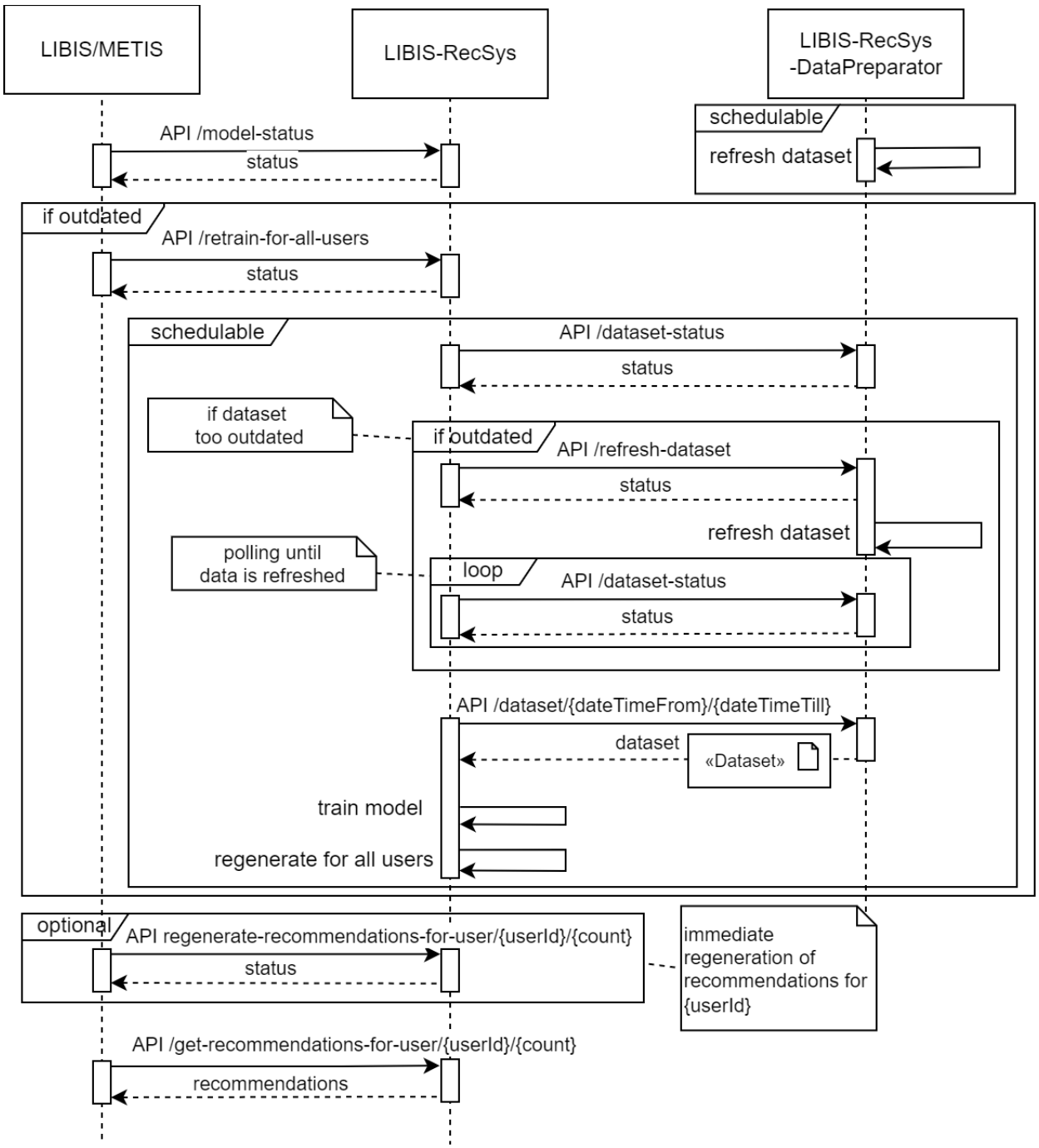


Figure 35: Sequence diagram of integration of LIBIS-RecSys and LIBIS-RecSes-DataPreparator into LIBIS ecosystem

4.3. Deployment

Deployment of LIBIS-RecSys-DataPreparator and LIBIS-RecSys could follow industry standard Continuous Integration/Continuous Deployment (CI/CD) practice, with following notes:

1. LIBIS-RecSys should train and perform evaluation on small static dataset in test phase to ensure appropriate ML model setup is used. Tests should fail if acceptable HR and nDCG metrics are not reached.
2. Having TEST environment could allow to perform online experiments with test users. One could initialize users which tend to like specific genres and see if recommendations

are sensible.

3. Deployment to PROD environment should be manual and only performed after online experiments in TEST environment.

The pipelines of LIBIS-RecSys-DataPreparator and LIBIS-RecSys CI/CD are shown in Figures 36 and 37 respectively.

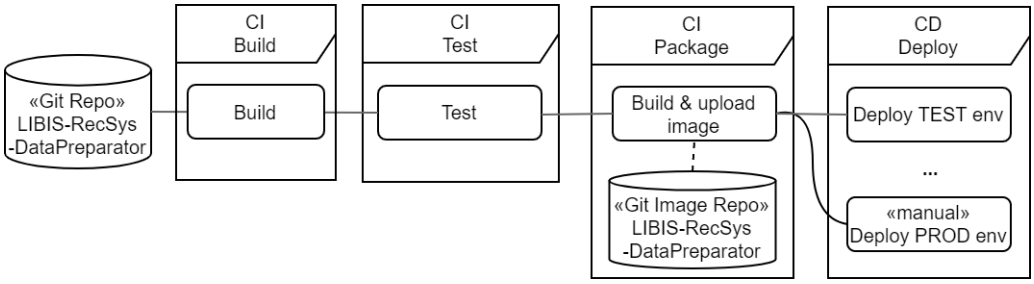


Figure 36: CI/CD pipeline of LIBIS-RecSys-DataPreparator

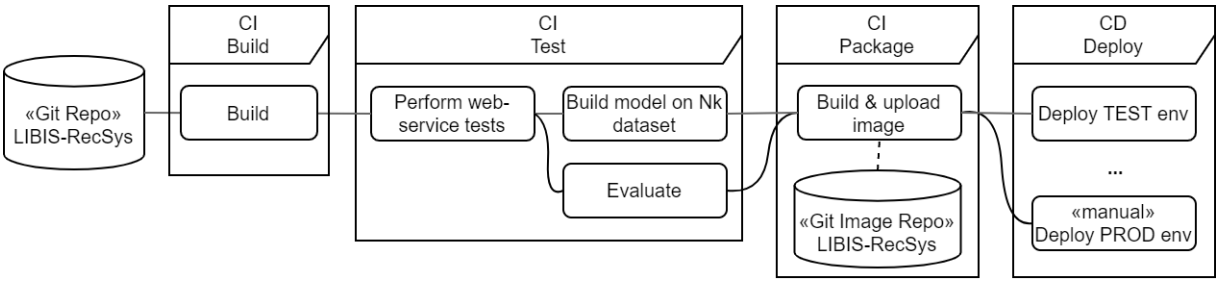


Figure 37: CI/CD pipeline of LIBIS-RecSys

Results

1. Analysis of data available in LIBIS and METIS projects showed that METIS scope does not have enough data for recommender system training, which forced to develop recommender system based on implicit feedback. Comparing order amount in LIBIS to rating amount in METIS, LIBIS provides 1500 times more feedback information compared to METIS.
2. Created LIBIS implicit datasets for recommender system training: libis-100k, libis-1m, libis-5m, libis-5m-dense, libis-10m, libis-20m. Additionally, datasets with negative samples were generated as typically training on implicit datasets require negative data. For each book order an additional 4 orders of unseen books are added as non-ordered. This resulted in respectively approximately 500k, 5m, 25m, 50m and 100m datasets. All the datasets have been refreshed at 2022-05-12. Only 100k dataset has been made publicly available. Access to larger datasets may be given if asked.
3. Created LIBIS corresponding datasets for hybrid recommender system training, providing demographics attributes for users and content attributes for records.
4. Implemented and evaluated Neural Collaborative Filtering technique on LIBIS implicit dataset. Evaluation on libis-20m dataset, resulted in 0.66 HR@10 and 0.38 nDCG@10. This is poorer performance compared to algorithms provided by "Implicit" library. Although, NCF algorithm allows for further tuning and development of a hybrid model.
5. Experiments with Wide & Deep technique on libis-100k implicit dataset yielded only 0.036 nDCG@10, whereas other algorithms yield between 0.11 and 0.21 nDCG@10 on this dataset. Meanwhile, training on libis-1m dataset used more than 20GB of RAM, and could not finish training in time interval of 1 hour. This technique was ruled out from further research due to poor evaluation results, slow trainining and impractical resource usage .
6. Implemented Switching & Mixed hybrid recommender which primarily runs Neural Collaborative Filtering (NCF). Although to resolve new user cold-start problem, the hybrid runs Demographics filtering recommender based on K-means clustering. Moreover, to resolve new item cold-start problem, the hybrid runs Content-based filtering recommender based on TF-IDF algorithm.
7. Provided in-depth suggestions on integration, deployment and production use of a recommender system in LIBIS/METIS ecosystem.
8. Implemented Flask-based LIBIS-RecSys prototype web service which serves hybrid recommender and additionally serves Collaborative filtering, Demographics filtering and Content-based filtering for using separately.

Conclusions

1. LIBIS and METIS feedback analysis confirm the statement that implicit feedback is more available. Thus, until METIS project starts to generate more feedback, METIS

scope should be dropped from recommender system research and development.
2. Datasets contain variety of personal information even if they are applied privacy preservation techniques.
3. METIS project developers should add timestamp on when the rating was given because it is a critical attribute for evaluating a recommender system, otherwise their collected ratings may be useless when developing the recommender system on their data.
4. Collaborative filtering implicit feedback recommendation system trained on LIBIS datasets would provide accurate recommendations for LIBIS/METIS users.
5. Datasets and attributes for hybrid recommender contain data and consistency errors made by public library employees. As a result, clustering techniques, for example demographics filtering based on KMeans clustering, may cause unreliable recommendations when unreliable attributes are used.
6. Privacy-preserving statistical bucketing for age groups of young people negatively affects recommendations from clustering techniques, thus youth should be split in narrower buckets than people above 18.
7. In order to save computing resources and reduce training time, implicit datasets should contain pre-generated negative samples. Number of negative samples is a trade-off between training time and accuracy.
8. Even though implicit feedback is more available in industry, academic papers and examples still mainly focuses on explicit feedback. This, in turn, narrowed algorithm selection for implementing recommender system for LIBIS as its feedback is implicit.

Further work directions

1. Once LIBIS-RecSys is deployed, explicit feedback on recommendations, such as "Not interested" or "Do not recommend this author" could be analyzed and used as negative feedback into recommender model.
2. If METIS started to generate sufficient amount of data, one could investigate use of multi-relational (or multi-behavior) recommender, which aims to leverage multiple user behavior data (such as view book, order book, rate book, etc) to improve the recommendation performance on the target behavior.
3. One could attempt to use different combination hybrid recommender system, there are tens if not hundreds possible combinations
4. UNIMARC data of records contain wide variety of information, which could be used to engineer more features. In addition to UNIMARC-based features, more statistical features could be engineered, for instance, average reader age for every record.
5. Content-based recommender could be extended to use more features than just genres. Sklearn TfidfVectorizer has FeatureUnion for this purpose.
6. More thorough research on errors in libis datasets is required, as there is a variety of errors and inconsistencies made by librarians.

References

[ACF21] M. Mehdi Afsar, Trafford Crump, and Behrouz Far. Reinforcement learning based recommender systems: A survey, 2021. URL: http://arxiv.org/abs/2101.06286.

[Agg16] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer International Publishing, Cham, Switzerland, 1st ed., 2016, pp. 1–498. ISBN: 978-3-319-29657-9. DOI: 10.1007/978-3-319-29659-3.

[BAH20] Jesus Bobadilla, Santiago Alonso, and Antonio Hernando. Deep learning architecture for collaborative filtering recommender systems. *Applied Sciences (Switzerland)*, 10(7), 2020. ISSN: 20763417. DOI: 10.3390/app10072441.

[Bena] Ben Frederickson. Implicit github repository. https://github.com/benfred/implicit. Accessed: 2021-01-12.

[Benb] Ben Frederickson. Implicit. Fast Python Collaborative Filtering for Implicit Datasets. https://implicit.readthedocs.io/en/latest/index.html. Accessed: 2021-01-12.

[Bie19] Dirk Bieresborn. *The impact of the General Data Protection Regulation on Social Security*, vol. 20 of number 2. 2019, pp. 285–306. ISBN: 9789284667710. DOI: 10.1007/s12027-019-00565-x.

[BL15] Joeran Beel and Stefan Langer. A comparison of offline evaluations, online evaluations, and user studies. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9316(September):153–168, 2015. ISSN: 16113349. DOI: 10.1007/978-3-319-24592-8{_}12.

[Bri21] Brian Dean. Netflix Subscriber and Growth Statistics: How Many People Watch Netflix in 2021? https://backlinko.com/netflix-users, 2021. Accessed: 2021-06-20.

[CAS16] Paul Covington, Jay Adams, and Emre Sargin. Deep Neural Networks for YouTube Recommendations. *RecSys 2016 - Proceedings of the 10th ACM Conference on Recommender Systems*:191–198, 2016. DOI: 10.1145/2959100.2959190.

[CDC] CDC. Positive Parenting Tips. https://www.cdc.gov/ncbddd/childdevelopment/positiveparenting/middle.html. Accessed: 2022-05-15.

[Cen19] Centrinis viešųjų pirkimų portalas. Skelbimas apie pirkimą. https://cvpp.eviesiejipirkimai.lt/Notice/Details/2019-616760, 2019. Accessed: 2021-06-15.

[CKH+16] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, et al. Wide & Deep Learning for Recommender Systems, 2016. URL: http://tensorflow.org..

[ÇM19] Erion Çano and Maurizio Morisio. Hybrid recommender systems: A systematic literature review. *Intelligent Data Analysis*, 21(6):1487–1524, 2019. ISSN: 15714128. DOI: 10.3233/IDA-163209.

[CMA16] Davy Cielen, Arno Meysman, and Mohamed Ali. *Introducing Data Science: Big Data, Machine Learning, and More, Using Python Tools*. Manning Publications Co., USA, 1st ed., 2016. ISBN: 1633430030.

[DCL+19] Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 1:4171–4186, 2019. URL: https://github.com/tensorflow/tensor2tensor.

[DSD17] Debashis Das, Laxman Sahoo, and Sujoy Datta. A Survey on Recommendation System. *International Journal of Computer Applications*, 160(7):6–10, 2017. DOI: 10.5120/ijca2017913081.

[FPG+18] William R. Frey, Desmond U. Patton, Michael B. Gaskell, and Kyle A. McGregor. Artificial Intelligence and privacy. *Social Science Computer Review*, (January):089443931878831, 2018. ISSN: 0894-4393. DOI: 10.1177/0894439318788314. URL: http://journals.sagepub.com/doi/10.1177/0894439318788314.

[GDJ10] Mouzhi Ge, Carla Delgado, and Dietmar Jannach. Beyond accuracy: Evaluating recommender systems by coverage and serendipity. In *RecSys'10 - Proceedings of the 4th ACM Conference on Recommender Systems*, pp. 257–260, 2010. DOI: 10.1145/1864708.1864761.

[Gjo13] Hristijan Gjoreski. Hybrid Recommender System for Personalized Poi Selection:19–22, 2013.

[HK15] F Maxwell Harper and Joseph A Konstan. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.*, 5(4), 2015. ISSN: 2160-6455. DOI: 10.1145/2827872. URL: https://doi.org/10.1145/2827872.

[HKV08] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative Filtering for Implicit Feedback Datasets, 2008.

[HLZ+17] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat Seng Chua. Neural collaborative filtering. *26th International World Wide Web Conference, WWW 2017*:173–182, 2017. DOI: 10.1145/3038912.3052569.

[HYC+17] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. Collaborative Metric Learning. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pp. 193–201, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee, 2017. ISBN: 9781450349130. DOI: 10.1145/3038912.3052639. URL: https://doi.org/10.1145/3038912.3052639.

[Int20] International Publishers Association and Norwegian Publishers Association. Reading matters Surveys and campaigns: How to keep and recover readers. Tech. rep., 2020.

[JLS+14] Shouling Ji, Weiqing Li, Mudhakar Srivatsa, and Raheem Beyah. Structural Data De-anonymization: Quantification, Practice, and Implications, 2014. DOI: 10.1145/2660267.2660278. URL: http://dx.doi.org/10.1145/2660267.2660278..

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. Tech. rep. 8, 2009, pp. 30–37. DOI: 10.1109/MC.2009.263.

[LCL+20] Chen Lin, Si Chen, Hui Li, Yanghua Xiao, Lianyun Li, and Qian Yang. Attacking Recommender Systems with Augmented User Profiles. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020. DOI: 10.1145/3340531.3411884. URL: http://dx.doi.org/10.1145/3340531.3411884.

[Lie] Lietuvos nacionalinė Martyno Mažvydo biblioteka. Suvestinio katalogo paskirtis. https://www.libis.lt/content/info/purpose.jsp. Accessed: 2021-06-10.

[LLJ+18] Wei-Han Lee, Changchang Liu, Shouling Ji, Prateek Mittal, and Ruby Lee. Blind De-anonymization Attacks using Social Networks, 2018. URL: https://arxiv.org/abs/1801.05534.

[LLJ+20] Dong Li, Zhi Liu, Ruoming Jin, and Jing Gao. On Sampling Top-K Recommendation Evaluation. 12, 2020. DOI: 10.1145/3394486.3403262. URL: https://doi.org/10.1145/3394486.3403262.

[Mar21] Maryam Mohsin. 10 Youtube Statistics That You Need to Know in 2021. https://www.oberlo.com/blog/youtube-statistics, 2021. Accessed: 2021-06-20.

[MB19] Sushma Malik and Mamta Bansal. Recommendation system: Techniques and issues. *International Journal of Recent Technology and Engineering*, 8(3):2821–2824, 2019. ISSN: 22773878. DOI: 10.35940/ijrte.C5211.098319.

[Ngo18] Lionel Ngoupeyou Tondji. Web Recommender System for Job Seeking and Recruiting. *IEEE Internet Computing*, 60(12):169–182, 2018. DOI: 10.13140/RG.2.2.26177.61286.

[OLL+14] Kyo Joong Oh, Won Jo Lee, Chae Gyun Lim, and Ho Jin Choi. Personalized news recommendation using classified keywords to capture user preference. *International Conference on Advanced Communication Technology, ICACT*:1283–1287, 2014. ISSN: 17389445. DOI: 10.1109/ICACT.2014.6779166.

[OOK+15] Burcu Okkalioglu, Murat Okkalioglu, Mehmet Koc, and Huseyin Polat. A survey: deriving private information from perturbed data. *The Artificial Intelligence Review*, 44(4):547–569, 2015. ISSN: 02692821. URL: http://search.proquest.com/docview/1728123846/.

[PE21] Vasileios Perifanis and Pavlos S Efraimidis. Federated Neural Collaborative Filtering, 2021.

[QA18] Shahzad Qaiser and Ramsha Ali. Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents. *International Journal of Computer Applications*, 181(1):25–29, 2018. DOI: 10.5120/ijca2018917395.

[Rav18] Ravindra Parmar. Word vector representations - Word2Vec. (January), 2018. DOI: 10.13140/RG.2.2.27845.65766. URL: https://medium.datadriveninvestor.com/word-vector-representations-word2vec-87b6bf0a755e.

[RFG+09] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: Bayesian Personalized Ranking from Implicit Feedback, 2009.

[RKZ+20] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. Neural Collaborative Filtering vs. Matrix Factorization Revisited, 2020.

[SGB16] Lipi Shah, Hetal Gaudani, and Prem Balani. Survey on Recommendation System. *International Journal of Computer Applications*, 137(7):43–49, 2016. ISSN: 0975-8887. DOI: 10.5120/ijca2016908821.

[SLW+19] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. 11, 2019. DOI: 10.1145/3357384.3357895. URL: https://doi.org/10.1145/3357384.3357895.

[Soe15] Budi Soediono. A Student-Centered Hybrid Recommender System to Provide Relevant Learning Objects from Repositories. *Journal of Chemical Information and Modeling*, 9192(August):160, 2015. ISSN: 1098-6596. DOI: 10.1007/978-3-319-20609-7. URL: http://link.springer.com/10.1007/978-3-319-20609-7.

[SS16] Richa Sharma and Rahul Singh. Evolution of recommender systems from ancient times to modern era: A survey. *Indian Journal of Science and Technology*, 9(20), 2016. ISSN: 09745645. DOI: 10.17485/ijst/2016/v9i20/88005.

[TAC18] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Latent Relational Metric Learning via Memory-based Attention for Collaborative Ranking:11, 2018. DOI: 10.1145/3178876.3186154. URL: https://doi.org/10.1145/3178876.3186154.

[TCC21] Chieh Yuan Tsai, Yi Fan Chiu, and Yu Jen Chen. A two-stage neural network-based cold start item recommender. *Applied Sciences (Switzerland)*, 11(9), 2021. ISSN: 20763417. DOI: 10.3390/app11094243.

[TTZ+19] Lucas Vinh Tran, Yi Tay, Shuai Zhang, Gao Cong, and Xiaoli Li. HyperML: A Boosting Metric Learning Approach in Hyperbolic Space for Recommender Systems, 2019.

[VOC+19] Priscila Valdiviezo-Diaz, Fernando Ortega, Eduardo Cobos, and Raul Lara-Cabrera. A Collaborative Filtering Approach Based on Naïve Bayes Classifier. *IEEE Access*, 7:108581–108592, 2019. ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2933048.

[Wol10] Judith A Wolfe. UNIMARC Manual Bibliographic Format. *Library Collections, Acquisitions, & Technical Services*, 34(4):133, 2010. DOI: 10.1080/14649055.2010.10766282. URL: https://doi.org/10.1080/14649055.2010.10766282.

[WWL+13] Yining Wang, Liwei Wang, Yuanzhi Li, Wei Chen, et al. A Theoretical Analysis of NDCG Ranking Measures:1–30, 2013.

[ZYS+19] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys*, 52(1):1–35, 2019. ISSN: 15577341. DOI: 10.1145/3285029.

Abbreviations

Term	Description
RS	Recommendation System
LIBIS	Lithuanian Integral Information System of Libraries (Lithuanian: Lietuvos integrali bibliotekų informacijos sistema)
METIS	METIS (Modern Digital Content Storage and Dissemination; acronym for Lithuanian: „Modernaus elektroninio turinio išsaugojimas ir sklaida“). METIS is the portal for readers (ibiblioteka.lt).
UNIMARC/B	Unified Machine-Readable Cataloging / Bibliographic
UDC	Universal Decimal Classification
HR	hit ratio
nDCG	normalized discounted cumulative gain

Source Code

Repository	URL	Description
libis-rs-dataset-preparator	gitlab.com/senvaitis/libis-rs-dataset-preparator	This project is responsible for the preparation of datasets for libis recommendation system research. Project is based on SQL scripts and Windows Powershell scripts (ps1).
libis-rs-als-bpr-gpu	kaggle.com/senvaitis/libis-rs-als-bpr-gpu	Allows training & evaluating Alternating Least Squares or Bayesian Personalized Ranking Collaborative Filtering algorithms. Hence, GPU training in implicit library requires libraries that are available ONLY on Linux OS!
libis-rs	gitlab.com/senvaitis/libis-rs	• libis-rs-cf-ncf-gpu.ipynb Trains, evaluates, exports Neural Collaborative Filtering (NCF) model. • libis-rs-df-kmeans-clustering.ipynb Trains, evaluates, exports K-means clustering model for Demographics filtering. • libis-rs-cbf-tf-idf.ipynb Trains, exports TF-IDF model for Content-based filtering. • libis-rs-hybrid.ipynb Runs Switching & Mixed pipelined hybrid recommender combining CF, DF, CBF recommenders. • libis-rs-server.ipynb Runs Flask web-service which serves Hybrid, CF, DF, CBF recommenders as well as endpoints for debugging.
libis-rs-wide-and-deep	https://github.com/senvaitis/recommenders/blob/main/examples/00_quick_start/wide_deep_libis.ipynb	MicroSoft's wide_deep_movielens.ipynb example adapted for LIBIS, however training is ultimately slow and highly demanding on resources. Has been run on libis-100k dataset.

Appendix no. 1

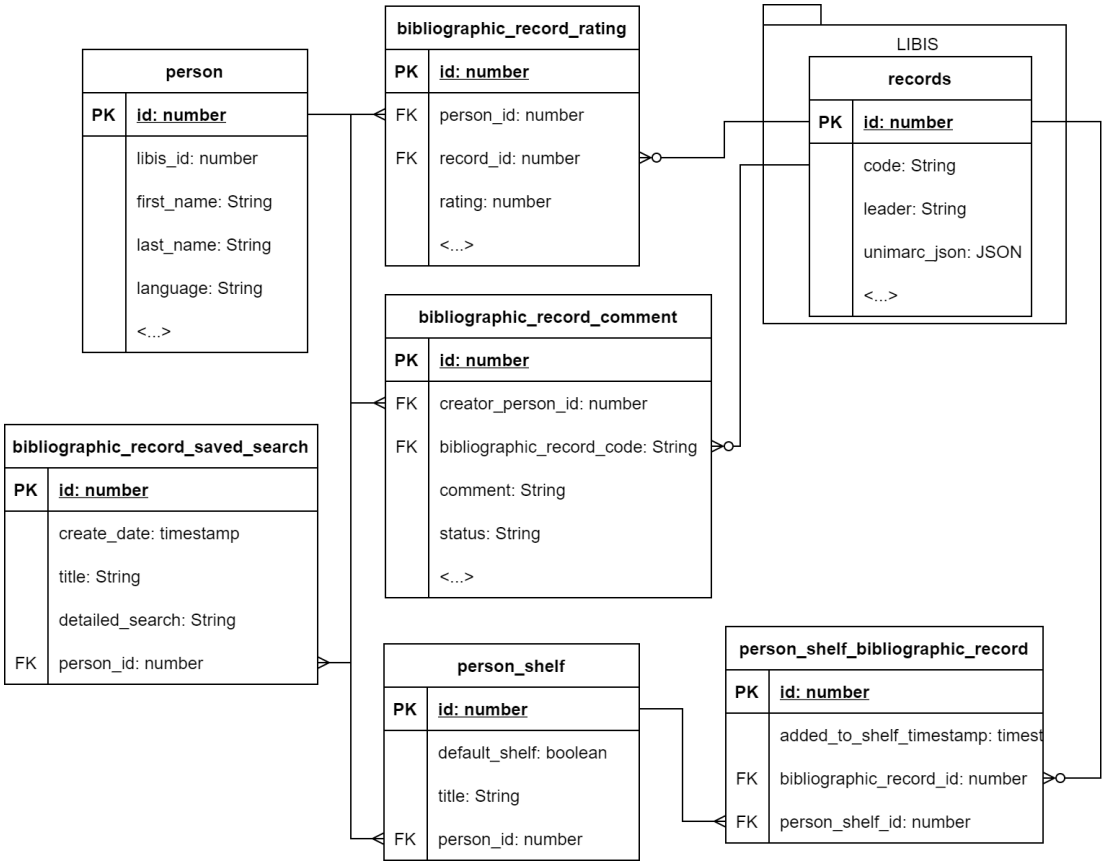
LIBIS: assessment of user table information sensitivity

This table provides full structure of users table and maps sensitivity and utility to each column. Only rows with 'high' utility and are composed into dataset, where rows with high 'sensitivity' are applied mitigation strategy.

Column	Type	Example	Technical	Sensitivity	Utility	Mitigation
id	Number			High	High	Pseudo id generation
academic_degree_id	Classifier	Daktaras		High	Med	
academic_degree_other	Text			High	None	
academic_name_id	Classifier	Docentas		Med	Med	
academic_name_other	Text			High	None	
active	Boolean		TRUE			
birth_date	Date			High	High	Statistical
contact_email	Text			High	None	
debtor	Boolean			Low	None	
education_id	Classifier	Aukštasis		Med	High	
education_other	Text			High	None	
educational_faculty_id	Classifier	Ekonomikos fakultetas		Med	Med	
educational_faculty_other	Text			High	None	
educational_institution_id	Classifier	Vilniaus universitetas		Med	Med	
educational_institution_other	Text			High	None	
email	Text			High	None	
facebook_user_id	Number			High	None	
first_name	Text			High	None	
gender	Categorical			Low	High	
global_reader_group_id	Classifier	Pensininkai		Med	High	
google_user_id	Number			High	None	
hashed_password	Text		TRUE			
identification_code	Text			High	None	
identification_code_type_id	Classifier	Lietuvos pilietis		Med	High	
identity_state	Categorical	UNVERIFIED		Low	None	
is_currently_studying	Boolean			Med	High	
last_name	Text			High	None	
level_in_edu_institute_id	Classifier	Magistrantas 2 k.		Med	Med	
level_in_edu_institute_other	Text			High	None	
modified	Date		TRUE			
note	Text			High	None	
occupation_id	Classifier	Elektrikas		Med	High	
occupation_other	Text			High	None	
old_id	Number		TRUE			
old_password	Text		TRUE			
parent_permission_granted	Boolean		TRUE			
parent_permission_id	Number		TRUE			
parent_permission_info	Text		TRUE			
participates_in_newsletters	Boolean		TRUE			
participates_in_polls	Boolean		TRUE			
password_change_date_time	Timestamp		TRUE			
password_reset_date_time	Timestamp		TRUE			
password_reset_key	Text		TRUE			
password_valid_till_date	Timestamp		TRUE			
phone_number	Text			High	None	
profile_photo_file_id	Number			High	None	
reader_age_group	Categorical	S		Med	Mid	
reader_ticket_id	Number			High	None	
registration_type	Categorical	DATA_IMPORT		Low	None	
residence_address_id	FK addresses			High	High	Select Country & County
temporary_residence_address_id	FK addresses			High	High	Select Country & County
usage_agreement_date_time	Timestamp		TRUE			
user_name	Text			High	None	
user_type	Categorical	READER		Low	None	
version_num	Number		TRUE			

Appendix no. 2

METIS: database structure in regards to user feedback



Appendix no. 3

LIBIS: assessment of user table information sensitivity

This table provides full structure of users table and maps sensitivity and utility to each column. Only rows with 'high' utility and are composed into dataset, where rows with high 'sensitivity' are applied mitigation strategy.

regexp	genre	regexp	genre
^087.*	Pažintinė literatūra vaikams	^71.*	Rajonų planavimas. Landšafto formavimas
^0.*	Bendrasis skyrius	^72.*	Architektūra
^1.*	Filosofija, psichologija	^73.*	Plastiniai menai
^2.*	Religija, teologija	^74.*	Piešimas. Taikomoji dekoratyvinė dailė. Dizainas
^30.*	Visuomenės mokslų teorija ir metodai	^75.*	Tapyba
^31.*	Demografija. Sociologija. Statistika	^76.*	Grafika. Graviūra
^32.*	Politika	^77.*	Fotografija. Kinematografija
^33.*	Ekonomika. Ekonomikos mokslai	^78/.089.*	Natos
^34.*	Teisė. Teisės mokslai	^78.*	Muzika
^35.*	Valstybės administracinis valdymas. Karyba	^791.*	Kino menas. Masinės šventės
^36.*	Socialinis aprūpinimas ir socialinė pagalba. Draudimas	^792.*	Teatras. Scenos menas
^37.*	Švietimas. Ugdymas. Mokymas. Laisvalaikio organizavimas	^793.*	Šventės. Choreografija
^389.*	Metrologija. Matai. Svoriai	^79[4-9]*	Žaidimai. Sportas
^39.*	Etnografija. Papročiai. Tautosaka	^79.*	Renginiai. Žaidimai. Sportas
^3.*	Visuomenės mokslai	^7.*	Menas, fotografija, žaidimai, sportas
^51.*	Matematika	^8.*-93.*	Grožinė literatūra vaikams ir jaunimui
^52.*	Astronomija. Geodezija	^8.*\.09.*	Literatūros kritika ir literatūros mokslas
^53.*	Fizika	^8.*\:929.*	Literatūros kritika ir literatūros mokslas
^54.*	Chemija. Mineralogija	^8.*\(09\).*	Literatūros kritika ir literatūros mokslas
^55.*	Geologija. Geofizika	^8.*\(09[1-4]\).*	Literatūros kritika ir literatūros mokslas
^56.*	Paleontologija	^8[0-1].*	Kalbotyra
^57.*	Biologija	^8.*	Grožinė literatūra
^58.*	Botanika	^90[2-4].*	Archeologija
^59.*	Zoologija	^908.*	Kraštotyra
^5.*	Gamtos mokslai	^912.*	Žemėlapiai
^61.*	Medicina	^91.*	Geografija
^62.*	Inžinerija. Technika apskritai	^929.*	Biografijos
^63.*	Žemės ūkis. Miškų ūkis. Medžioklė. Žuvininkystė	^947\.45.*	Lietuvos istorija
^64.*	Namų ūkis. Komunalinis ūkis. Buities	^9[3-9].*	Istorija
^65.*	Įmonių valdymas. Gamybos, prekybos ir transporto organizavimas	^9.*	Istorija
^66.*	Cheminė technologija. Chemijos pramonė		
^67.*	Įvairios pramonės šakos ir amatai. Mechaninė technologija		
^68.*	Įvairios pramonės šakos ir amatai. Tikslioji mechanika		
^69.*	Statyba. Statybinės medžiagos		
^6.*	Taikomieji mokslai		