FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

# Assignment of master's thesis

| | |
|---|---|
| **Title:** | Recommendation system for Data Dictionary application. |
| **Student:** | Bc. Valeriy Lyalin |
| **Supervisor:** | Ing. Michal Peroutka |
| **Study program:** | Informatics |
| **Branch / specialization:** | Knowledge Engineering |
| **Department:** | Department of Applied Mathematics |
| **Validity:** | until the end of summer semester 2022/2023 |

## Instructions

The aim of the work is to develop a recommendation tool for the Data Dictionary application. This tool should recommend based on historical data of user interactions and based on object(item) similarities.

Instructions for elaboration:
1. Get acquainted with Data Dictionary(DD) use cases.
2. Familiarise yourself with the dataset of the DD application, which will be provided by one unnamed data warehouse (after applying data anonymization).
3. Search for recommendation algorithms.
4. Design and implement a recommendation system.
5. Test and evaluate your implemented solution.

*Electronically approved by Ing. Karel Klouda, Ph.D. on 20 October 2021 in Prague.*

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

# Recommendation system for Data Dictionary application

***Bc. Valeriy Lyalin***

Department of Applied Mathematics
Supervisor: Ing. Michal Peroutka

April 30, 2022

# Acknowledgements

I would like to thank everybody who directly or indirectly contributed to the master thesis. My special thanks go to Ing. Michal Peroutka, the supervisor of the thesis, who mentally and physically supported me during the whole process of working on the thesis, and who shared his rich experience and always was there when I needed. Also, I want to thank my parents and Julia for their mental support during the whole study process.

# Declaration

In Prague on April 30, 2022 . . . . . . . . . . . . . . . . . . . .

## Citation of this thesis

Lyalin, Valeriy. *Recommendation system for Data Dictionary application.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

# Abstrakt

Práce si klade za cíl analyzovat a navrhnout prototyp doporučovácího systému pro aplikaci Data Dictionary. Práce se skládá ze tří částí. První část začíná nastíněním problématiky datového slovníku a případami užití aplikace. Poté poskytuje přehled doporučovácích systémů. Druhá část obsahuje návrh, implementaci, ladění a vyhodnocení různých doporučovacích technik, jako je content-based recommendation, collaborative filtering a session-based recommendation. Finální část analyzuje dosažené výsledky a zaměřuje se na výběr modelu pro aplikaci.

**Klíčová slova** doporučovácí systémy, content-based recommendation, collaborative filtering, session-based recommendation

# Abstract

The thesis aims to analyze and build a prototype of a recommendation system for a Data Dictionary application. This work consists of three parts. The first part starts with outlining Data Dictionary and uses cases of the application. Then it provides an overview of recommendation systems. The second part contains the design, implementation, tuning, and evaluation of different recommendation techniques, such as content-based recommendation, collaborative filtering, and session-based recommendation. The last part analyzes achieved results and focuses on a model selection for the application.

**Keywords** recommendation systems, content-based recommendation, collaborative filtering, session-based recommendation

# Contents

# List of Figures

# List of Tables

Chapter 1

# Introduction

Before the rise of e-commerce, goods were sold solely in shops. Store's inventory was limited by its physical space, and products that did not sell well were unprofitable. Due to the fixed physical space of the shops, merchants were motivated to sell only the most popular mainstream products.

The development of internet marketplaces in the 1990s revolutionized the whole retail business. New digital spaces provided storage for an unlimited number of inventories. Some enterprising merchants decided not to lose this lucrative opportunity and gradually expanded the range of suggested products, which started to include more and more niche items. As a result, some less known pieces gained so much popularity that they unavoidable changed the merchants' vision on the possible ways how to make skyrocketing profits.

Inventory extension and a growing amount of niche products in reality, however, does not necessarily imply a profit increase. In 2000, two psychologists conducted a study [1] to prove that. They constructed two supermarket booths. Each of them was offering jams. The first one had 24 different types, whereas the second one was offering only 6. Two researching assistants worked as store employees and invited passing-by people to come in and try their jams. After a few days of "selling", they concluded the following: while the booth with more samples brought in more customers, the booth with fewer samples had higher conversation rates. Meaning, that people were more likely to purchase in the booth with a lower amount of jams than in one with a wider variety of tastes. The phenomenon is known as the "choice overload": when a person is given too many options to choose from, he is less inclined to purchase.

The e-commerce was not the only area exposed to the choice overload. When there is too much information on one's "plate", it becomes harder for a person to differentiate between suggested things and to make a final choice. A concept of information overload is encountered everywhere: in social networks, entertainment systems, businesses, etc. One of the possible ways of tackling this information saturation issue is by using a recommender system, which reduces the search space and identifies the most relevant items for all users.

Recommendation systems are not new. They evolved as an independent research area in the mid-1970s in the Duke University. The systems started to gain attention among businesses. In 1998 Amazon.com, Inc. was the first big company to launch an item-based collaborative filtering, that allowed large-scale recommendations for millions of customers and millions of catalog products. Since then, recommendation systems based on Collaborative Filtering have become widely popular and have been implemented by numerous e-commerce and online systems.

As of 2022, most e-commerce platforms have already included recommendation systems in their businesses. However, there are still areas where the problem of information overload is acute and recommender systems have just started to be adopted there. An example of such areas can be corporate systems designed for the internal needs of companies. As companies grow, so does the amount of their company data. Systems that were originally created and tested for a small amount of data become more and more difficult to use over time. An example of such a system is the data dictionary application, which is used in one metadata warehouse of a company in the finance sector. The application is primarily utilized for designing, managing, and documenting systems and databases used in a variety of companies' projects. The users of the system, such as data analytics or data engineers, might have issues with finding a table with a certain name since it can exist in multiple systems simultaneously. In that case, a recommender system may assist with finding the most relevant table for the particular user. This master thesis aims to build a recommendation system for the data dictionary application to help users to find the most relevant information in a more efficient way.

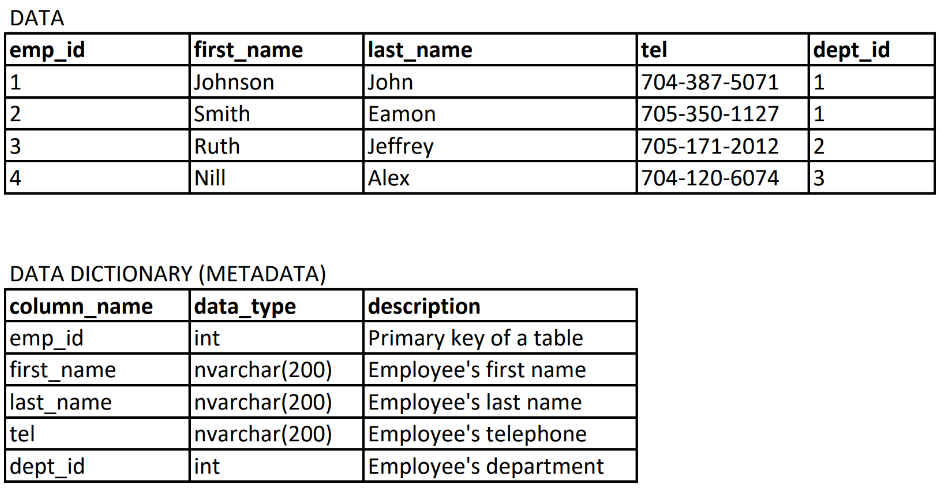
Chapter 2

# Introduction to data dictionary

Term Data Dictionary, according to IBM Dictionary Of Computing [2], means a centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format. A data dictionary does not store the data but rather metadata (data about data). An example of a simple Data Dictionary is shown in figure 2.1.

## 2.1 Sources of information in a data dictionary

Data in a relational database is structured. The structure of data is defined in a schema. A database schema represents the logical configuration of all or part of a relational database [3]. This schema includes information about stored tables, columns, views, indexes, constraints, procedures, functions, and more.



DATA

| emp_id | first_name | last_name | tel | dept_id |
|---|---|---|---|---|
| 1 | Johnson | John | 704-387-5071 | 1 |
| 2 | Smith | Eamon | 705-350-1127 | 1 |
| 3 | Ruth | Jeffrey | 705-171-2012 | 2 |
| 4 | Nill | Alex | 704-120-6074 | 3 |

DATA DICTIONARY (METADATA)

| column_name | data_type | description |
|---|---|---|
| emp_id | int | Primary key of a table |
| first_name | nvarchar(200) | Employee's first name |
| last_name | nvarchar(200) | Employee's last name |
| tel | nvarchar(200) | Employee's telephone |
| dept_id | int | Employee's department |

Figure 2.1: An example of a simple Data Dictionary application

The data itself can be treated as a simple data dictionary, but often it is one of many sources of an enterprise data dictionary application. Other sources are models from data modeling tools, metadata repositories, data catalogs, etc.

## 2.2 Typical attributes in a data dictionary

Typical attributes of a data dictionary that are imported from a database schema are item names, nullability, data types, default values, length, precisions, views and procedure definitions, incoming and outgoing references, child-parent relationships (e.g. a relation between a column and a table that it belongs to). Apart from that data, a database can be queried for data profiling information, such as minimum, maximum, median values in columns, number of nullable values, row count, uniqueness, and more.

Data models from modeling tools enhance the information by adding descriptions, definitions, annotations, and more. Besides the documenting information, it may provide mapping information. This kind of information is crucial for understanding sources of information and capturing usages of the data, such as data reports and other systems.

## 2.3 Functions of a data dictionary

A data dictionary plays a crucial role in a process of designing, managing, and documenting a database. Thanks to search it may allow to quickly and effectively access desired data. It enables analysts to understand overall system design and data flow. The application assists in reducing inconsistencies across systems. It helps in defining conventions that are used across the whole project and enforces users to follow them.

Chapter 3

# Defining goals for a recommendation system

## 3.1 What is a recommendation system?

A recommender system or a recommendation system is a subclass of information filtering systems that seeks to predict the "rating" or "preference" that a user would give to an item [4]. In other words, recommender systems are active information filtering systems that personalize the information shown to a user depending on his/her interests, the item's relevancy, and other factors. The recommendation is mainly done by building and training a model based on a comparison of characteristics between users (user-based recommendation), items (item-based recommendations), and users' domain (collaborative filtering). Different algorithms are used to implement the recommendation system. Some of them will be covered and implemented in this thesis.

## 3.2 Functions of a recommendation system in a data dictionary

Enterprise data dictionaries contain tons of information. Orientation and data discovery in this area is complicated. Tools like Elasticsearch [5] might partially solve the issues. The engine allows developers to store, search and analyze huge volumes of data in milliseconds. It can produce quick search results because it searches an index rather than querying the text directly. Given a query, it computes a relevance score for each item. Then the ones with the highest score are returned to a user. Developers may modify the score function by assigning weights to each attribute of stored data. Though, there is a problem with the approach. It handles duplicates poorly. Suppose a user searched for a table called Party. He/she types the keywords. Based on the keyword, the system outputs, for instance, thousands of results with the top ten results of tables named Party. One of the solutions would be adding additional filters to queries, such as a department name. But what if the user does not know the name of the department? He/she will have to look at each of the ten results. Going through ten results is bearable, but what if a data dictionary contains thousands of tables called Party? Thankfully, there is another solution to the problem. The data dictionary system can personalize results for users by collecting information about user activities and eventually adding a recommender system.

Despite the enormous amount of items in an enterprise data dictionary, users have repeated patterns in the item interaction history. For instance, data analytics have a subset of information that interests only them. A portion of the items may be accessed solely by a single person. Another set of items may be visited more frequently and by multiple individuals. Based on the knowledge, a recommendation system can forecast the next visited items.

## 3.3 Goals for building a recommendation system

The aim of the thesis is to build a recommendation system. As described in the previous section, the recommendation engine will be used in an enterprise data dictionary to improve search and potentially improve navigation by recommending the most relevant items to the user even without searching.

The item recommendation problem in an enterprise data dictionary is complex due to the large number of items. For that reason, users do not expect high recalls from the recommender system. However, narrowing the search results based on relevance score from a search engine will decrease the number of items and more than likely will improve the recall.

Chapter 4

# Content-based filtering

A content-based (CB) recommendation is a technique that predicts the following interactions based on user profiles. The process of building a recommender model with CB consists of three parts: building item vectors, developing user profiles based on item vectors, and eventually making predictions based on a similarity between a user profile and an item vector. Items with the highest similarity scores are recommended to a user.

## 4.1 Generating item embeddings

The first step in building the recommender system is to create item embeddings. Items usually have not only number features but also text attributes, such as name, description, category name, etc. To feed these features to a machine learning model, we need to create embeddings. Embeddings are essentially a set of numbers. More specifically, it is a low-dimensional space in which we translate high-dimensional vectors, ideally, with preserving semantics. Techniques for translating item features into embeddings are described in the following sections.

### 4.1.1 TF-IDF model

One way of creating item vectors is called TF-IDF, which stands for term frequency and inverse document frequency. It is calculated as a product of term frequency $tf$ and inverse document frequency $idf$ (4.1).

$$\begin{aligned} tfidf(t,d,D) &= tf(t,d) * idf(t,D) \\ tf(t,d) &= \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \\ idf(t,D) &= log\frac{N}{|\{d \in D : t \in d\}|} \end{aligned} \tag{4.1}$$

Term frequency $tf$ measures how often a word $t$ occurs in a document $d$. A term that frequently appears in a document is likely to be crucial to its meaning. Document frequency $df$ measures how often a word occurs in an entire set of documents $D$. If a word is frequent in a provided document $d$ and common in all of the given documents, it will get a low score. Examples of these words are “the”, “a”, “is”, etc. These words do not carry information about a context.

TF-IDF is a decent baseline technique for embedding generation. However, it has several limitations. It does not take into consideration similarities between words, word positioning, and semantics. The method assumes that word frequency provides independent evidence of similarity. Last but not least, word counting may be slow for extensive vocabularies.

### 4.1.2 Word2vec model

Word2vec is not a singular algorithm. Rather, it is a family of model architectures and optimizations that can be used to learn word embeddings from large datasets. Embeddings learned through word2vec have proven to be successful on a variety of downstream natural language processing tasks [6]. There are two approaches for constructing word embeddings with word2vec: the continuous bag-of-words model and the continuous skip-gram model.

**Continuous bag-of-words model** of word2vec predicts the middle word based on surrounding context words (defined in the equation 4.2).

$$P(w_t|w_{t-C}, w_{t-C+1}, ..., w_{t-1}, w_{t+1}, ..., w_{t+C}) \tag{4.2}$$

Consider the following sentence: *today is a great day for a walk*. Given the neighborhood words $\{a, great, for, a\}$ the method will try to predict the target word *day*. As described in study [7], the approach uses the one-hot encoding of the input words and measures the output error compared to the one-hot encoding of the target word. It converts input one-hot encodings of words into a lower-dimensional feature vector and reconstructs the original one-hot representation from the vector. The architecture is shown in figure 4.1. In addition, the output layer does not have an activation function but only soft-max. More precisely, it uses hierarchical soft-max, which is computationally cheaper compared to a regular one. Then the method includes word subsampling to reduce the training set.

**Continuous skip-gram model** predicts words within a certain range before and after the current word in the same sentence (defined in the equation 4.3).

$$P(w_{t-C}, w_{t-C+1}, ..., w_{t-1}, w_{t+1}, ..., w_{t+C}|w_t) \tag{4.3}$$

The continuous skip-gram model is the opposite of the continuous bag-of-words model. It takes a word one-hot representation as input and outputs

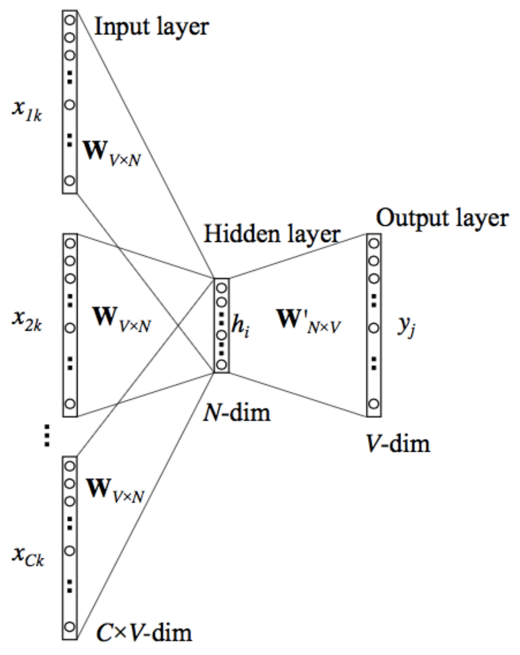


Figure 4.1: Continuous bag-of-words model architecture. Source: [8]

$C$ vectors, each representing the probabilities of words. The architecture is shown in figure 4.2.

The word2vec method had shown good performance, according to the paper [10], compared to competitors. Howbeit, it has some cons. The approach processes only words it has seen during the training process. The next problem is that embeddings generated with word2vec are content-independent. For instance, the word *bank* in the following sentences will have the same embedding: “We went to the river bank”, and “I need to go to the bank to pay the bill”. The third problem is that word2vec does not take into account the word position. All of the previously mentioned problems are solved by a method called Bidirectional Encoder Representations from Transformers.

### 4.1.3 BERT model

Bidirectional Encoder Representations from Transformers (BERT) is a machine learning technique for natural language processing (NLP) pre-training developed by Google [11]. Since its introduction in 2018, BERT obtained state-of-the-art results in numerous benchmarks and is still a must-have baseline [12].

BERT is essentially a stack of Transformer encoder layers made up of several self-attention heads. Each head computes key $K$, value $V$, and query $Q$ vectors for each input token in a sequence to produce a weighted repre-

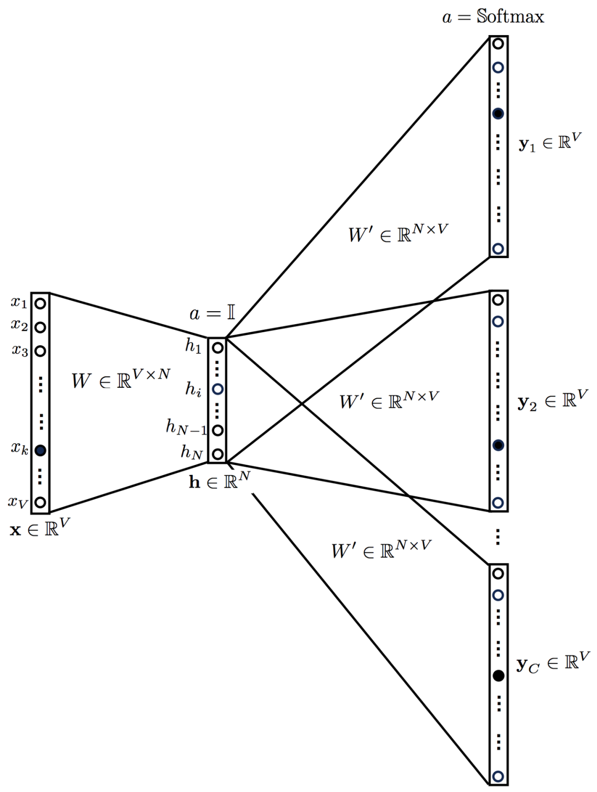

$a = \text{Softmax}$
$\mathbf{y}_1 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
$a = \mathbb{I}$
$x_1$
$x_2$
$x_3$
$x_k$
$x_V$
$\mathbf{x} \in \mathbb{R}^V$
$W \in \mathbb{R}^{V \times N}$
$h_1$
$h_i$
$h_{N-1}$
$h_N$
$\mathbf{h} \in \mathbb{R}^N$
$W' \in \mathbb{R}^{N \times V}$
$\mathbf{y}_2 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
$\mathbf{y}_C \in \mathbb{R}^V$

Figure 4.2: Continuous skip-gram model architecture. Source: [9]

sentation. All heads in the same layer have their outputs combined and run through a fully connected layer. Each layer is wrapped with a skip connection and then normalized. Transformer architecture is shown in figure 4.3.

BERT has several advantages over the previously mentioned methods for creating word embeddings. Studies have shown that the model has syntactic knowledge [13]. Syntactic awareness monitors the relationships between words in order to understand the meaning. BERT also has some knowledge of semantic roles [14], which involves the meaning found in the actual text. Another advantage is that BERT is available and pre-trained in over 100 languages. That can be useful for projects that are not English-based.

## 4.2 Building user profiles

The next phase of building a content-based recommendation system after generating item embeddings is building user profiles. Given a set of items $I$, ratings $R$ and item embeddings $V$ we can calculate user profiles $P$ (4.4).

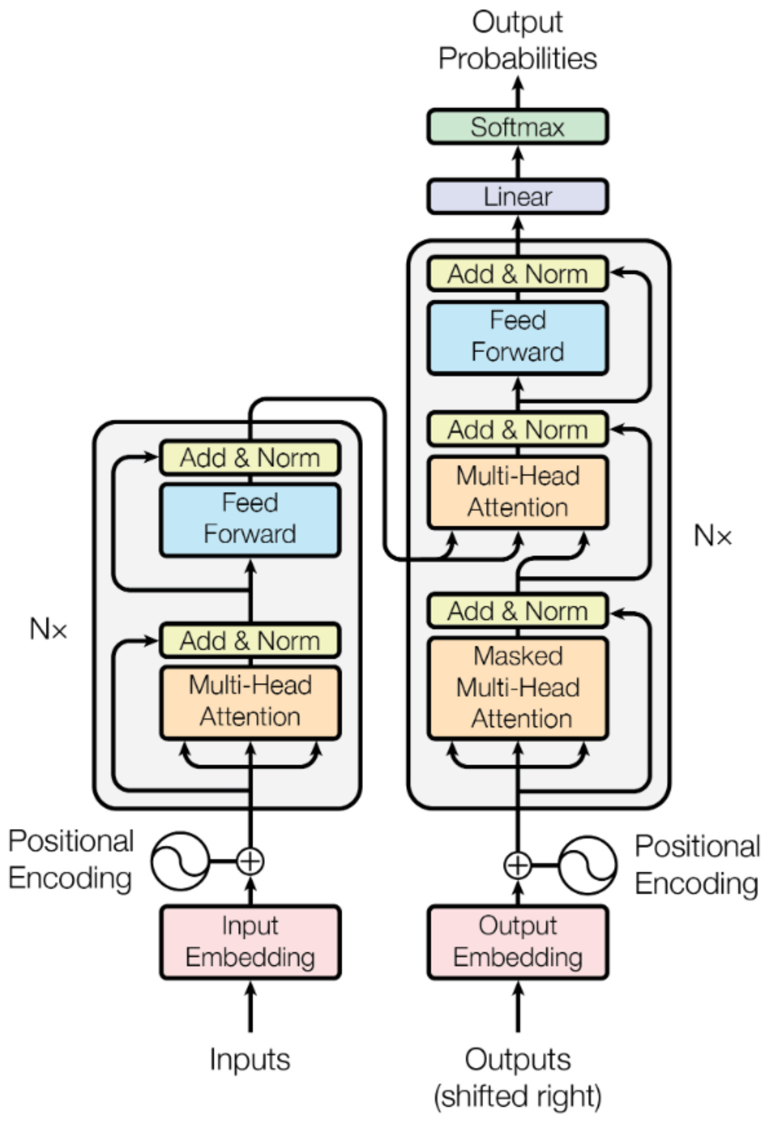


Figure 4.3: The Transformer - model architecture. Source: [15]

$$P_j = \sum_{i \in I} r_{ij} * v_i \tag{4.4}$$

When we have multiple features $F$, each having its embedding, we can sum up all the embeddings, or we can assign weights $W$ to attributes and then sum up (4.5).

$$P_j = \sum_{i \in I} \sum_{f \in F} r_{ij} * w_f * v_{if} \tag{4.5}$$

To speed up computation, we can use matrix multiplication on GPU.

## 4.3 Next item prediction using a content-based model

The last step in building a content-based recommendation model is choosing a similarity function. The function is used when making predictions. For any given user profile, it calculates the similarity between each item and the user vector. Then *top-k* items are recommended to a user.

One of the most popular ways of measuring similarity between two vectors $A$ and $B$ is cosine similarity (4.6).

$$sim(A,B) = cos(\Theta) = \frac{A * B}{|A| * |B|} \tag{4.6}$$

There are other ways of measuring vectors' similarities, such as Euclidian, Manhattan, Minkowski distances, Jaccard Similarity, and others.

## 4.4 Cons and pros of a content-based recommendation system

Content-based recommendation system has the benefit of recommending new items that haven't been visited yet. Another advantage of the model is that it can satisfy the unique tastes of a user. It is not biased towards the most popular items. Despite the outlined benefits, it also has some problems. It can only recommend objects similar to those that the user has interacted with (e.g. visited). The model cannot detect changes in user tastes over time. The last issue, however, can be somewhat solved by adding aging to the rating generation process.

Chapter **5**

# Collaborative filtering

Collaborative filtering is the process of filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc [16]. While content-based recommendation focuses on recommending items that are the most similar to the ones that a user has interacted with, collaborative filtering tries to find similarities in historical data of user interactions. It assumes that users that had similar tastes in the past will have similar tastes in the future. There are different types of collaborative filtering, but all of them work with rating matrices to produce the model.

## 5.1 User-based collaborative filtering

User-based collaborative filtering makes predictions based on similarities between users. Based on a similarity function *sim(u,v)* it finds a set $N_k = \{v_1, v_2, ..., v_k\}$ of $k$ users that is the most similar to a given user $u$. Then using ratings of the most similar users it calculates ratings estimations (5.1).

$$\hat{r}_{u,i} = \begin{cases} \dfrac{\sum\limits_{\substack{v \in N_k \\ r_{v,i} \neq ?}} sim(u,v) * r_{v,i}}{\sum\limits_{\substack{v \in N_k \\ r_{v,i} \neq ?}} sim(u,v)} & \text{if } \exists v \in N_k : r_{v,i} \neq ? \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

To measure similarities between users $sim(u,v)$ we can use cosine similarity $C(u,v)$ (5.2), Jaccard Similarity $J(u,v)$ (5.3), Pearson correlation coefficient $P(u,v)$ (5.4), and others.

$$C(u,v) = \frac{r_{u,*} * r_{v,*}^T}{\|r_{u,*}\|_2 * \|r_{v,*}\|_2} = \frac{\sum_{\substack{i \in I \\ r_{u,i} \neq ? \\ r_{v,i} \neq ?}} r_{u,i} * r_{v,i}}{\sqrt{\sum_{\substack{i \in I \\ r_{u,i} \neq ?}} r_{u,i}^2} \sqrt{\sum_{\substack{i \in I \\ r_{v,i} \neq ?}} r_{v,i}^2}} \tag{5.2}$$

$$J(u,v) = \frac{r_{u,*} * r_{v,*}^T}{\|r_{u,*}\|_2^2 * \|r_{v,*}\|_2^2 - r_{u,*} * r_{v,*}^T} = \frac{\sum\limits_{\substack{i \in I \\ r_{u,i} \neq ? \\ r_{v,i} \neq ?}} r_{u,i} * r_{v,i}}{\sum\limits_{\substack{i \in I \\ r_{u,i} \neq ?}} r_{u,i}^2 + \sum\limits_{\substack{i \in I \\ r_{v,i} \neq ?}} r_{v,i}^2 - \sum\limits_{\substack{i \in I \\ r_{u,i} \neq ? \\ r_{v,i} \neq ?}} r_{u,i} * r_{v,i}} \tag{5.3}$$

$$P(u,v) = \frac{\sum\limits_{\substack{i \in I \\ r_{u,i} \neq ? \\ r_{v,i} \neq ?}} (r_{u,i} - \overline{r_{u,*}}) * (r_{v,i} - \overline{r_{v,*}})}{\sqrt{\sum\limits_{\substack{i \in I \\ r_{u,i} \neq ?}} (r_{u,i} - \overline{r_{u,*}})^2} * \sqrt{\sum\limits_{\substack{i \in I \\ r_{v,i} \neq ?}} (r_{v,i} - \overline{r_{v,*}})^2}} \tag{5.4}$$

## 5.2 Item-based collaborative filtering

Item-based collaborative filtering processes the rating matrix in a way complementary to the User-based method. In the approach, the sim function calculates item similarity rather than assessing similarity between users. However, formulas used for calculating similarities are the same. After computing a set of $k$ the most similar items for every item in the rating matrix, we can estimate missing ratings as similarity-weighed average (5.5).

$$\hat{r}_{u,i} = \begin{cases} \dfrac{\sum\limits_{\substack{j \in I \\ i \in N_k \\ r_{u,j} \neq ?}} sim(i,j) * r_{u,j}}{\sum\limits_{\substack{j \in I \\ i \in N_k \\ r_{u,j} \neq ?}} sim(i,j)} & \text{if } \exists j \in I : i \in N_k(j) \land r_{u,j} \neq ? \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

## 5.3 Rating generation

There are two sources of ratings: explicit and implicit. Explicit ratings are essentially ratings given to items by users. To obtain explicit feedback from users the system must ask the user to rate an object. The collection process can be implemented in the form of likes and dislikes, stars, or even writing comments as text. The last one gives a great opportunity to learn user opinion, but it is not easy to obtain and hard to evaluate.

Explicit feedbacks are usually biased toward popular items. People tend to give higher ratings to more prevalent items. Another issue with explicit ratings is that people rate differently. For instance, in a movie streaming service with ratings of one to five, two persons may rate the same film as four, but the first person gives ratings of four to films that he does not like, and for the second individual, the rating of four is the highest one. These issues should be solved during preprocessing.

Implicit ratings are generated from user interactions. These days most of the systems are collecting viewing, and searching data, adding to bookmarks, time spent on reading or watching a particular item, and others. Each interaction type is weighted based on its importance and then summed up. For instance, when a user buys an item, it will get higher ratings compared to a case when the user only clicked on the object.

Most modern systems collect both implicit and explicit ratings. If a system collects only explicit ratings, the rating matrix will probably be sparse. Applications may also collect only implicit ratings, but to maximize prediction quality, we need as much data as possible.

## 5.4 Matrix factorization for collaborative filtering

The core of item-based and user-based methods of collaborative filtering is to find a set of nearest neighbors using the similarity function. Calculating similarities requires a lot of computational power. Another problem is that the result estimation matrix of the rating matrix is sparse because users (or items) in a set of nearest neighbors will only have a few ratings. One way of partially solving these problems is to use a matrix factorization technique.

Matrix factorization means the decomposition of an original matrix into two or more matrices with specific non-trivial properties. The main idea behind the method is the following: given a rating matrix $R \in \mathbb{R}^{m,n}$ find matrices $P \in \mathbb{R}^{m,k}$ and $Q \in \mathbb{R}^{n,k}$ so that known ratings are approximated by $\hat{R} = P * Q^T$, where $k$ is given as a hyperparameter. These matrices are obtained by solving the optimalization task 5.6, where $r_{u,i}$ is known rating of $u$th user and $i$th item, $p_u$ is $u$th row of $P$, $q_i$ is the $i$th row of $Q$ and $\lambda$ is a hyperparameter.

$$argmin_{U,V} \sum_{\substack{\forall (u,i) \in (U \times I) \\ r_{u,i} \neq ?}} (r_{u,i} - p_u^T q_i)^2 + \lambda (\sum_i q_i^T q_i + \sum_u q_u^T q_u) \tag{5.6}$$

The bottom line of the method is that when we have already found the matrices $P$ and $Q$, then we can multiply $\hat{R} = PQ^T$. Known entities in the matrix $R$ will be similar to ones in $\hat{R}$, and unknown elements in $R$ will be estimated by known. As a result, the $\hat{R}$ matrix will have all the entities known.

Matrix factorization for collaborative filtering was made famous by Simon Funk, who used the technique to place third in the 2006 Netflix competition.

The method is sometimes called FunkSVD because it can be understood as an approximation of the approximation given by Singular Value Decomposition (SVD). Using Eckart-Young Theorem, it can be shown that the result matrix of Simon Funk's method corresponds to the matrices from SVD using only $k$ biggest singular values.

## 5.5 Cons and pros of collaborative filtering

On the one hand, collaborative filtering does not require items information. This may be considered as an advantage because we do not need to process complex item structures, such as music, videos, or images. On the other hand, the method works only with a rating matrix, and we cannot simply add an item or user information, such as item category or user's age and country. Another significant problem with collaborative filtering is that it cannot recommend items that no one hasn't interacted with yet. The last issue with collaborative filtering is that it tends to suggest more popular objects, meaning users with unique tastes will probably be disappointed.

CHAPTER **6**

# Session-based recommender systems

Recommendations systems based on collaborative filtering or content-based are generally tended to operate with historical user-item interactions to learn a user's long-term preferences. The fundamental premise of both of these systems is that all previous interactions are equally important in determining the user's current choice. However, in reality, this may not be the case. For instance, recently viewed or purchased items may be more appropriate than others. Furthermore, user preferences toward certain items tend to be dynamic rather than static. As a result of these issues, a new class of recommendation algorithms has been developed: known as session-based recommendation algorithms (SBRS). These algorithms rely strongly on the user's most recent interactions rather than the user's prior preferences.

## 6.1 Session properties and components

A session-based recommendation system works with user-item interactions user and item data. These entities are the core components of sessions, which is an essential element of every session-based recommendation model. In the section, the author of the thesis outlines the session properties and its key components.

### 6.1.1 User and user properties

In an SBRS, a user $u$ is the subject who performs actions on objects, such as clicks or purchases, and receives the recommended results. Users $U = \{u_1, ..., u_n\}$ are associated with a unique identification and a set of attributes that describes them. As identification is considered *user_id* or *cookie_id* in the case of an anonymous person. In entertainment systems, user attributes may include gender, age, and location. For example, a boy may watch more action movies, while a girl may watch more love-story movies. In business domains, user information may include department numbers. One department may use only a part of a system and have a few intersections with another department. However, user information may not always be available because it may not be recorded due to privacy protection, or the system may allow anonymous users to use it.

### 6.1.2 Item and item properties

In a session-based recommender system, an item $p_i$ is a recommendation entity, such as a product or service. Items $P = \{p_1, ..., p_m\}$ are associated with an identification number and with a set of attributes that describes them, such as name, price, category, description, etc.

### 6.1.3 Action and action properties

In SBRM, an action $a_i$ is an operation performed by a user. Action has a unique identification based on an action type. Examples of action types are the following: click, search, like/dislike, add to a cart, buy an item, etc.

### 6.1.4 Interaction and interaction properties

Interaction in an SBRS is a tuple $o = \langle u, p, a, t \rangle$ of a user $u$, an item $p$, an action type $a$ and a timestamp $t$. Based on the item recency, the model may prefer newer items and not recommend older objects that only appeared in a certain period. For instance, in an online grocery store, products such as ice cream have time periodicity. People buy more ice cream during the hot seasons and almost do not buy it during cold seasons.

### 6.1.5 Session and session properties

A session is a non-empty determinate list of interactions $s = \{o_1, ..., o_k\}$ created over a period of continuous-time and associated with a certain user through userId or a cookie. Note that a session is not a set, but a list, meaning that it may have duplicates. For instance, a person may listen to a song multiple times. A session, like its components, is commonly associated with attributes, such as session duration, time, day of the occurrence, and others. A session is a core element of any session-based recommendation system. The process of session generation is complex and depends on a specific domain. In the next subsections, the author describes common session properties that may have a great impact on an SBRS.

#### 6.1.5.1 Session length

The length of a session is determined by the total number of interactions it contains. Sessions can be divided into three groups based on their length: long sessions, medium sessions, and short sessions. Note that the particular definitions for long, medium and short sessions may vary depending on the data set.

Long sessions contain a relatively large number of interactions. In some domains, a session might be considered to be long when it has more than ten or fifteen interactions. Generally, long sessions give more contextual information for more accurate recommendations. Nevertheless, a long session is more likely to contain random interactions that are unrelated to the rest of the session. As a result, there is more noise in the data, which reduces the performance of recommendations. Furthermore, long sessions generally contain more complex dependencies, such as long-range dependencies between two interactions that are far apart in the session.

Medium-long sessions contain an average number of interactions, e.g. from five to ten. Medium size sessions are the most common ones in the e-commerce industry [17]. Compared to long and short sessions, the medium session is more likely to have fewer unnecessary interactions while still containing minimal contextual information. For instance, in an e-shop selling electronics with an average session length of bought items of three, a person who buys a cell phone and a charger for the phone are likely to buy a case for the phone.

A session is considered to be short when it has significantly fewer interactions than average. In e-commerce, it may be sessions of less than four interactions. When working with this number of sessions, we should not expect high results from a recommendation system.

#### 6.1.5.2 Session action types

A session can be a single-type-action session or a multi-type-action session. A single-type-action session includes only one action type, e.g. click, search, purchasing, etc. Therefore, one only form of action comes from the same set of activities, which may be easier to learn, but potentially miss some patterns that may increase the overall performance of the model.

A multi-type-action session contains more than one action type. In e-commerce domains, users may first click on multiple items to compare them and then buy an item or search for an item and purchase it straight away. Therefore, there are complex dependencies inside a session: one action type may lead to the same action type or a different one.

#### 6.1.5.3 Internal order

There are three different types of interaction ordering within a session: unordered, strict ordered, and flexible ordered. The ordering type depends on a dataset or on a domain.

An unordered session is made up of interactions that are not in any particular order. For instance, in an online grocery store, adding items to a basket is rather unordered. When a session is unordered, the dependencies among its interactions are based on their co-occurrence rather than their sequences. Furthermore, compared with sequential dependencies, co-occurrence-based dependencies are generally weaker, which makes them more difficult to learn.

A session is ordered when it contains multiple interactions with a strict order. This ordering type is present in online educational platforms. For example, a fingerstyle guitar course requires an understanding of guitar basics. An advanced fingerstyle course demands knowledge of fingerstyle basics and so on. Ordered sessions are easier to learn. Nevertheless, it is still challenging because in a long session dependencies become weaker.

A session's interaction ordering is flexible when it has some ordered parts while others are unordered. Flexible ordering takes into account sequentially dependent interactions and assigns less weight to random actions that are not related. For example, when a session consists of adding to a basket a cell phone, a cover, and a charger to the cell phone and a cartridge, the recommender system should be able to figure out that the cartridge is probably out of the order.

#### 6.1.5.4 User information

When a user is allowed to use a service anonymously, almost all of the sessions will not have connections, meaning they will seemingly belong to different users, even if it is not the case. When sessions are anonymous, it is nearly impossible to collect prior knowledge of user interactions. Therefore only contextual information from a current session may be used for generating recommendations.

In a system that requires authorization, many (or most) of the users will have more than one session. This prior information may improve the performance. However, the recommender system will have an additional challenge of capturing changes in the user's taste.

## 6.2 Session-based recommendation problem

Given a session $s = [o_1, o_2, ..., o_i, o_{i+1}, ..., o_k]$ in the session set $S$ and a set of unique items $P = \{p_1, p_2, ...., p_n\}$, we build a model $M$ so that $y = M(s)$, where $y = \{y_1, y_2, ..., y_n\}$ is a ranking set of item's scores over all the next items that can occur in that session. Moreover, for $\forall j \in \{1, 2, ..., n\}$ $y_j$ is a score of item $p_j$. Since a recommendation system needs to give more than one recommendation for a user, therefore top-k items with the largest scores $\{p_1, p_2, ..., p_k\}$ are recommended, where $k \ll n$ and $(\forall i, j \in \mathbb{N})(i < j)(j \leq m)|(y_i < y_j)$.

## 6.3 K nearest neighbors approaches for SBRS

SBRS techniques based on K Nearest Neighbours (KNN) are simple yet proven to be effective [18]. In the approaches, interaction is generally reduced to an item ID. KNN-based techniques for SBRSs might be separated into item-KNN and session-KNN, depending on whether the similarity is determined between items or sessions.

### 6.3.1 Item KNN for session-based recommendation

Given a current session $s = \{p_1, p_2, ..., p_k\}$ from a session set $S = \{s_1, ..., s_m\}$, item-based KNN takes the last item in the session $p_k$ and using a similarity function $sim : V \times V \to \mathbb{R}$ finds $k$ most similar items, where $V_i = \{o_1, ..., o_m\}$ and $(\forall j \in 1, ..., m)(o_j \in \{0, 1\})$ is a binary vector of an item $p_i$ across all sessions $S$. The value of $o_i$ of an item $p_z$ if calculated based on equation 6.1.

$$o_i = \begin{cases} 1 & \text{if } p_z \in s_i \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

The list of similarity functions that can be applied here is the same as it was described in item-based collaborative filtering.

### 6.3.2 Session KNN for session-based recommendation

Given a current session $s_i$ the session-based KNN firstly finds $k$ a set of the most similar sessions $N_s$ and then calculates the score for each candidate item $p$ using the equation 6.2.

$$score(p) = \sum_{s_{nb} \in N(s)} sim(s, s_{nb}) * 1_{s_{nb}}(p) \tag{6.2}$$

In the equation 6.2, $sim$ is a similarity function between two sessions and $1_{s_{nb}}(p)$ is a funciton that returns 1 when the session $s_{n_{nb}}$ contains the item $p$ and 0 otherwise.

Unlike item-KNN, which only analyzes the current item in the session context, session-KNN evaluates the whole session context and therefore captures more information for more accurate suggestions.

### 6.3.3 Cons and pros of K nearest approaches for SBRS

The session-based approach of K nearest neighbors, as mentioned earlier, is proven to be effective. However, the model is not suitable for large-scale applications with millions of sessions due to computation complexity. To make a prediction, the method requires calculating the similarity between a session and all of the other sessions. With the rapidly growing number of sessions, the calculation becomes too slow even for asynchronous calculations of top-k recommendations.

## 6.4 Recurrent neural network approaches for SBRS

Recurrent Neural Networks (RNN) are one of the most popular approaches for a session-based recommendation task. The main idea of RNN consists of two parts: context generation and prediction. Firstly, an RNN model feeds each ordered session through recurrent layers. Secondly, it takes the last hidden state modeling the context representation as the input to predict the next interaction. In the section, the author of the thesis outlines some implementations that use the concept.

One of the relatively popular approaches that used RNN to tackle session-based recommendation problems is called GRU4Rec [19]. As the name implies, the technique uses Gated Recurrent Units (GRU) to predict the probability of the subsequent events, such as clicks given a session beginning. Figure 6.1 shows the architecture of the network, in which the embedding, the feedforward, and additional GRU layers are optional.

The network accepts a single item as input in the form of a one-hot encoded vector that represents the entire item space. As an output, the model yields a similar-shaped vector of a rating distribution for the next item. During the process, the GRU layer keeps track of a hidden state that encodes the previously occurring items in the same session. When the session ends, the hidden state of the GRUs has to be reset. Note that due to the architectural design, the sequence must be fed in the correct order. In terms of the activation functions in the last layer, the authors of GRU4Rec [19] found that the *tanh* and *sigmoid* functions work best for the GRU and the ranking layer. Training is accomplished with stochastic gradient descent using established optimizations like Adam [20].

All of the mentioned above is not new compared to the vanilla RNN. The GRU4Rec approach uses innovatively session-parallel mini-batches to speed up the training phase and ranking-based loss function. Each time a session at a certain position in the batch ends, the corresponding hidden state is reset. Then the next batch update will place a new session at that position. The process is illustrated in figure 6.2.

The approach uses as a loss function generalized version of Bayesian Personalized Rating, which is defined in equation 6.3.

$$L_s(\hat{r}_{s,i}, s_N) = -\frac{1}{|S_N|} \cdot \sum_{j \in S_N} log(\sigma(\hat{r}_{s,i} - \hat{r}_{s,j})) \tag{6.3}$$

In the loss function 6.3, $\hat{r}_i$ is the predicted rating for the actual item $i$ and $S_N$ is a set of negative samples. Functions, such as sigmoid and logarithm are

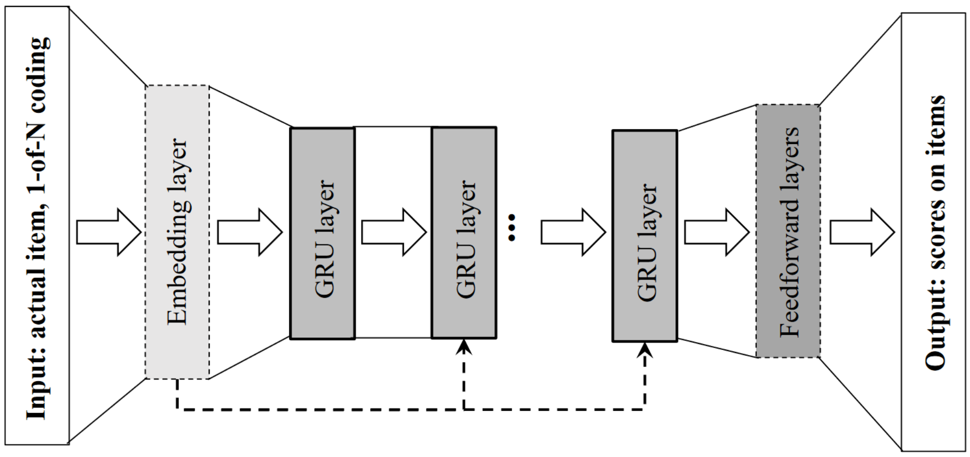


Figure 6.1: Architecture of the GRU4Rec neural network. Source: [19]

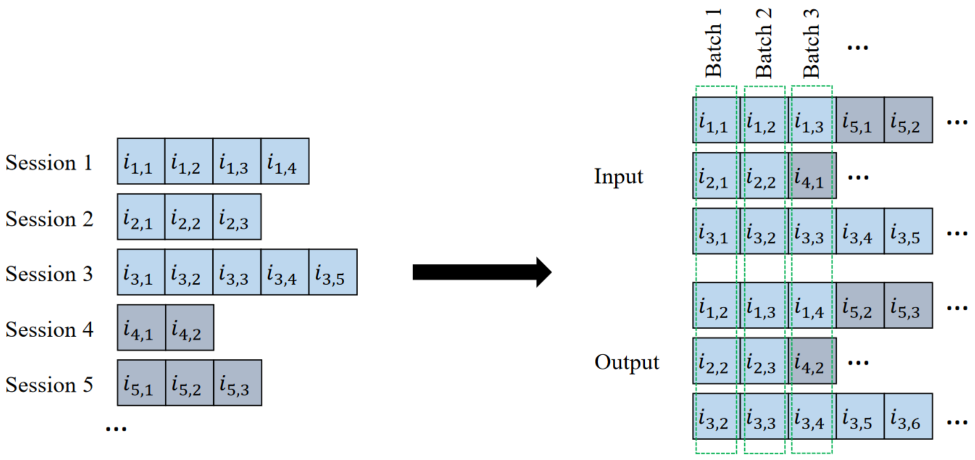


Figure 6.2: Illustration of the session-parallel mini-batch scheme of GRU4Rec. Source: [19]

applied to represent the proportion between the ranking of the negative and the positive example. The goal of the optimizer is set to maximize the loss function, in other words, to maximize the difference between the predicted rating for the actual item and the non-actual one.

## 6.5 Transformer4Rec approach for SBRS

The Transformer architecture was first presented in 2017 as an effective alternative to the RNN-based sequential encoder-decoder network [15]. The self-attention compared to RNN favors parallel processing and scales well for long sequences. At first, Transformers have been used mainly in Natural Language Processing (NLP) tasks. However, it has also proven to beat RNNs in sequential recommendation tasks, even when user sessions are shorter than NLP sequences [21].

One of the latest and influential approaches that uses Transformers in the field of session-based recommendation systems is Transformers4Rec. The method was developed by the NVIDIA research team and presented at RecSys 2021 conference [21].

The Transformer4Rec library was designed for large-scale applications. The library contains modules that form the whole pipeline for a session-based recommendation task. It includes its data preprocessing library NVTabular [22] which provides GPU-accelerated preprocessing of terabyte-sized RecSys datasets. Then it has training and evaluation modules. The Transformer4Rec supports PyTorch and TensorFlow frameworks. The pipeline overview is illustrated in figure 6.3.

### 6.5.1 Data preprocessing

Data preprocessing is generally a bottleneck in the session-based recommendation pipeline. It is the focus of the GPU-accelerated NVTabular library that is co-developed with Transformer4Rec. The data preprocessing library offers common and advanced feature engineering techniques. In addition, it includes specialized operations for the sequential and session-based recommendation, such as feature categorizing, grouping time-sorted interactions by user or session, and truncating sequences to the first or last N interactions. The preprocessed data are saved to Parquet format. The Transformer4Rec also requires a JSON-like metadata file that includes columns names, minimum and maximum values, as well as data types.

### 6.5.2 Model training and evaluation

Transformer4Rec is based on HuggingFace (HF) Transformers library. HF Transformers is an open-source library [23] that provides standardized efficient

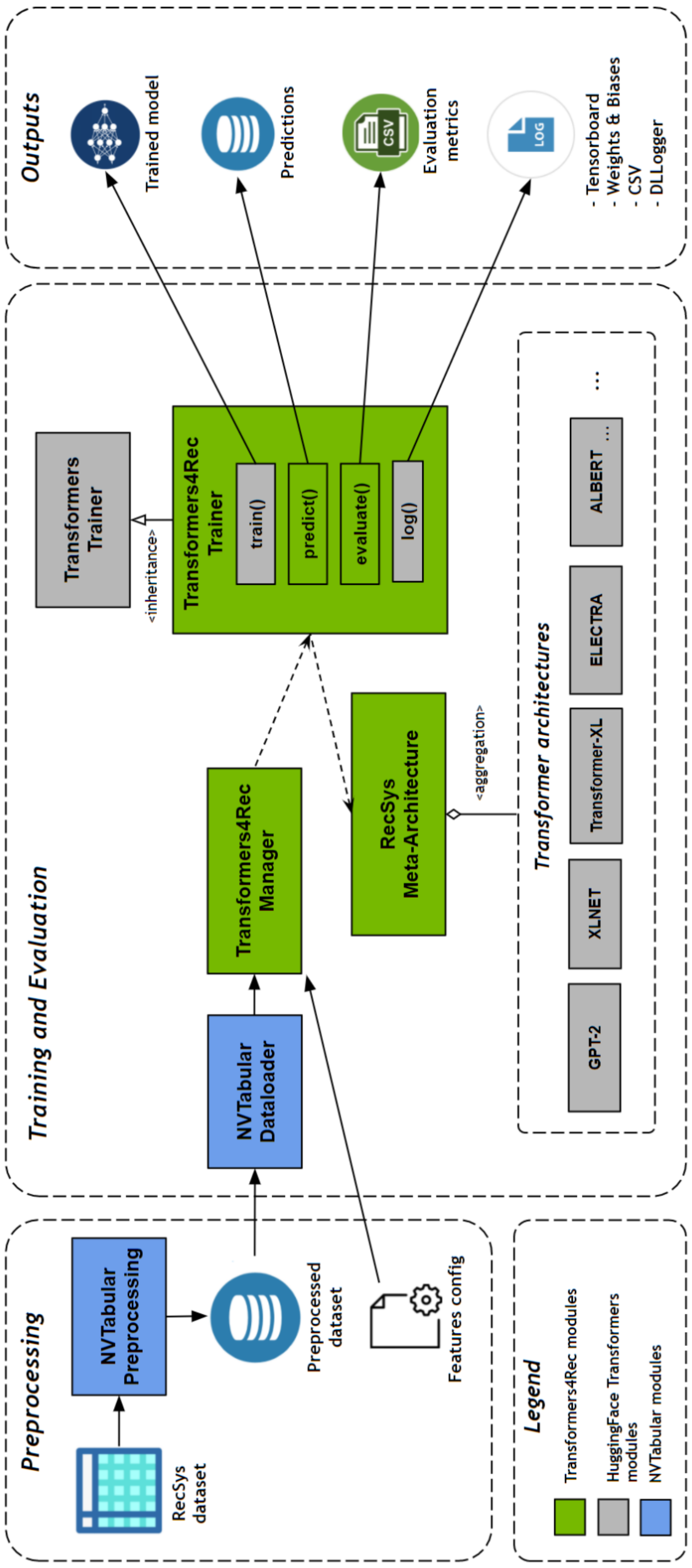


Figure 6.3: Transformers4Rec pipeline overview. Source: [21]

implementations of recent state-of-the-art Transformer architectures that have been already pretrained for different NLP tasks.

In the standardized HF Transformers API, a train and evaluation pipeline is managed by the Trainer class. The class among other methods provides *train()*, *predict()* and *evaluate()*. The Transformers4Rec library inherits from the class and overrides only the *predict()* and *evaluate()* methods to adjust them to the recommendation problem, keeping its original *train()* method, as it is identical for NLP and sequential recommendation.

In the training, Transformer4Rec uses loss function in form of cross-entropy, which is defined in the equation 6.4, where $I$ is a set of items, $N \in \{0, 1, ..., |I|\}$, $S \in \mathbb{R}^n$ a probability distribution with $\sum_{s_i \in S} s_i = 1 \wedge \forall s_i \in S : s_i \neq 0$ and $L$ is a sum of one-hot encoded item IDs that are recommended. The cross-entropy loss function is designed in a way that it most of all penalizes confident results that are not correct. For instance, when the item should be in the recommended set, but the model predicted the probability of 1%, the resulting loss value of the particular prediction will be 2 (in case of $log_{10}$). While for the probability of 30%, the loss function will be approximately 0.53.

$$D(S, L) = -\sum_{i \in N} L_i \cdot log(S_i) \tag{6.4}$$

The evaluation of session-based recommendations in Transformer4Rec is performed using traditional Top-N ranking metrics such as NDCG@N, Recall@N, Precision@N, and MAP@N. Greater detail about the metrics in the following chapter.

The prediction method outputs the probability distribution of items with dimensions according to the input schema loaded to the model. To be more precise, the dimensions are configured according to the given minimum and maximum item ID in the JSON-like schema.

### 6.5.3 Transformers4Rec meta-architecture

The Transformer4Rec meta-architecture consists of four modules. The illustration of the meta-architecture is shown in figure 6.4.

The first features processing module takes the input and eventually creates the interaction embedding. The process of producing embeddings consists of feature normalization and aggregation. In the Transfomer4Rec paper, authors do not describe normalization techniques, but generally the following techniques are used: min-max normalization(6.5), z-score normalization(6.6), where $x$ is a vector, $\mu$ is a mean value and $\sigma$ is a standard deviation of the vector. The aggregation is done by simply concatenating features. However, more complex methods are also available [21].

The Sequence Masking module masks the sequence of interaction embeddings according to the training strategy (e.g., Causal LM [24]) and feeds it to the Sequence Processing module.

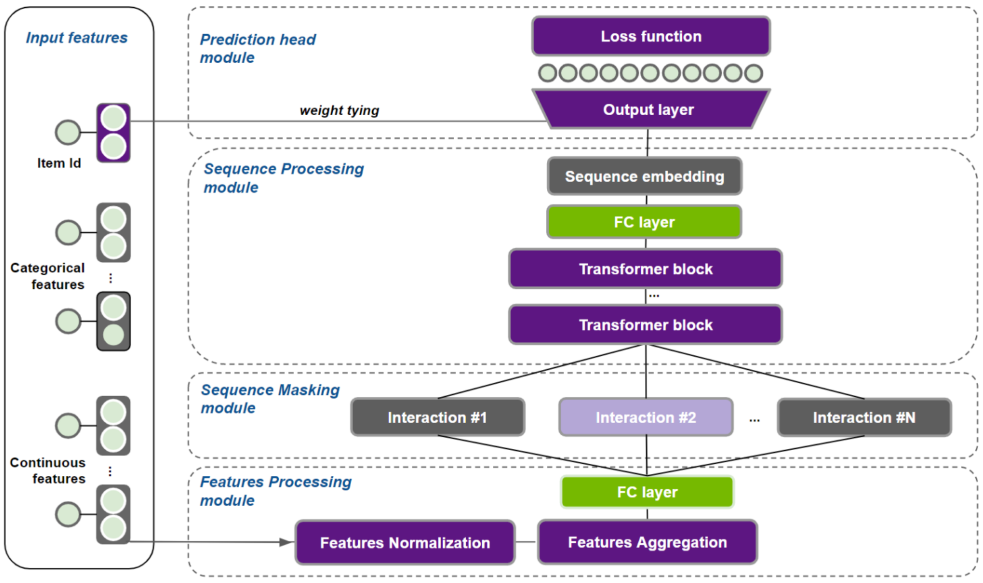


Figure 6.4: Transformer4Rec neural meta-architecture. Source: [21]

The Sequence Processing module includes stacked Transformer blocks. A number of blocks and architecture types (e.g. GPT-2, Transformer-XL, XL-Net, Electra) are configurable. It generates a vector for each sequence location, which is then projected to produce a sequence embedding.

The last prediction head module might be set up to do different tasks, including item prediction (for item suggestion) and sequence-level predictions (classification or regression).

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{6.5}$$

$$x' = \frac{x - \mu}{\sigma} \tag{6.6}$$

### 6.5.4 Transformer4Rec performance

Transformers had proven to outperform recurrent neural networks in natural language processing tasks. In a session-based recommendations task, the session lengths are shorter compared to NLP. For testing purposes, the authors had chosen four datasets: two from e-commerce [25, 26] and two from news portals [27, 28], with an average length of 5.49, 3.83 interactions for e-commerce, and 2.84, 2.69 for news. Many users in the news domain surf anonymously, with only their most recent interactions available. In the e-commerce platform, aside from the cold-start problem, users are also targeted for a certain item, and the current session delivers more useful information than past interactions from the user context.

The authors compared different approaches using NDCG@20 and HR@20 (more about the metrics in the following chapter). For comparison, authors had chosen algorithms based on Session-based k-Nearest Neighbor as baseline algorithms, such as V-SkNN [29], STAN [30], VSTAN [31]. The GRU4Rec [19] was taken as a baseline of RNN session-based recommendation. Lastly, different architectures of Transformer architectures were taken, such as BERT [32], ELECTRA [33], XLNet [34], and others.

Results show that session k-NN algorithms (V-SkNN, STAN, and VSTAN) are indeed strong baselines for a session-based recommendation, with higher HR@20 than some of the Transformer architectures. However, GRU4Rec is the best baseline for both e-commerce datasets and news datasets in terms of NDCG@20. Transformer architectures outperform the best baseline approaches by +14.15% NDCG@20 and +9.75% NDCG@20 e-commerce datasets than on the news datasets. The authors believe that it is due to the longer session length in the e-commerce datasets. No particular Transformer architecture performs best across all datasets. More about the results can be found in the Transfomer4Rec [21].

Chapter 7

# Evaluation of recommender systems

Regardless of the model design, it is necessary to assess how the chosen model performs. The motivation for the model assessment is simple. In general, the model assessment is used to answer the following questions. Do the recommendations ”work”? Do they increase sales? Which algorithm should be preferred for the application? Which parameter setting is better? This chapter describes methods to answer these questions.

## 7.1 Offline evaluation

Offline evaluations test the effectiveness of recommender system algorithms on a certain dataset. The goals of the offline evaluation are model selection and model assessment. During the model selection phase, the performance of different models is measured to pick the best one. After a model has been chosen in the model assessment phase, the error is then estimated on new unseen dataset.

If enough data is provided, the dataset undergoes splitting into three separate parts: train, test, and validation. Generally, 50% of the data is used for training, and 25% of the data is used for validation and testing. The trained set is used only for model training. The validation dataset is used to evaluate the model's performance and for the selection of hyperparameters. Moreover, the validation dataset is also used for model comparison. When a model has been selected, the test set is used to make a final estimation of the model performance.

When a dataset does not have enough data to divide it into three parts, one way of performing model selection and estimating the test error is k-fold cross-validation. When using the method, the dataset is divided into k parts (folds). Then $i$th split of a dataset is taken as a test, and other parts are used for training. The process repeats for $i = 1, 2, ..., k$. The final estimate of the test error is calculated as an average of the train errors. The number of folds is generally varying from five to ten. In some cases, it may even equal the sample size, which is called leave-one-out cross-validation.

Even though a model might be evaluated on a separate test set, the results of the offline evaluation should not be held as absolute truth, since the improvement of a model-based only on offline evaluation does not necessarily imply that it would perform better online. However, offline assessment should be used as a first step in deciding whether to examine a candidate model as a prospective replacement if it outperforms a present one based on multiple offline metrics. The following sections describe the metrics for offline evaluation of a recommender system.

### 7.1.1 Predictive accuracy metrics

Predictive accuracy metrics assess how close ratings estimated by a recommender system are to genuine users' ratings. These measures are used for non-binary ratings.

#### 7.1.1.1 Mean Absolute Error (MAE)

Mean absolute error is the average difference between the predicted rating by a recommender system and the actual rating given by the user. Since the predicted value may be larger or smaller than the value given by a user, an absolute value of the difference is taken. The measure is defined in equation 7.1, where $Y_i \in \mathbb{R}^N$ is a vector of users' ratings and $\hat{Y}_i \in \mathbb{R}^N$ are ratings generated by a recommendation system. Note that the function is not highly sensitive to outliers.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\left|Y_i - \hat{Y}_i\right| \tag{7.1}$$

#### 7.1.1.2 Mean Squared Error (MSE)

Mean squared error (MSE) is similar to MAE. The difference is, however, instead of taking absolute error to cancel the negative sign, we square it. The formula for calculation of MSE is defined in equation 7.2. Note that MSE is more sensible to outliers compared to MAE.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(Y_i - \hat{Y}_i\right)^2 \tag{7.2}$$

| | **Used** | **Not used** |
|---|---|---|
| **Recommended** | True Positive (TP) | False Positive (FP) |
| **Not recommended** | False Negative (FN) | True Negative (TN) |

Table 7.1: Classification of the possible result of a recommendation of an item to a user

#### 7.1.1.3 Root Mean Squared Error (RMSE)

MSE metric results may be hard to interpret because it scales up the errors. The issue is solved by taking the square root of MSE. The method is called root mean squared error (RMSE) and is defined in equation 7.3. RMSE disproportionately penalizes large errors, since the residuals are squared. This means that the metric is more affected by outliers than MAE. Nevertheless, RMSE has benefits over MAE. In the paper [35], the authors showed that given enough data, RSME allows for a reconstruction of the error set, whereas MAE can only accurately recreate 0.8 of the data set. Moreover, RSME does not use absolute values, making it more mathematically convenient when calculating gradient, distance, or other metrics.

$$MSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(Y_i - \hat{Y}_i\right)^2} \tag{7.3}$$

### 7.1.2 Classification accuracy metrics

Classification metrics are used when the recommender system does not predict the users' preferences of items, such as book or movie ratings, but instead tries to recommend the next items that users may interact with.

When a performance of a recommendation system built for the next item(s) prediction is been evaluated, the following four outcomes are possible: True Positive (TP), False Positive (FP), False Negative (FN), and True Negative (TN), as it is shown in table 7.1.

#### 7.1.2.1 General classification metrics

After computing the classification matrics, the following metrics are generally used: precision, recall, and f1-score. Precision (defined in equation 7.4) is the ratio between the True Positives and all the Positives. The precision measures the relevance of predictions. Recall (defined in equation 7.5) is the measure of a model correctly identifying True Positives. It gives an insight into a number of selected relevant items. F1-score (defined in equation 7.6) combines both precision and recall.

$$Precision = \frac{|TP|}{|TP| + |FP|} \tag{7.4}$$

$$Recall = \frac{|TP|}{|TP| + |FN|} \tag{7.5}$$

$$F1\text{-}score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{7.6}$$

#### 7.1.2.2 Classification metrics for a recommender system

All of the mentioned above metrics apply generally to a classification task. The recommendation task though a bit different in the way that the number of predicted and used items is generally limited to a certain number, e.g. ten or twenty-five. The number is mainly limited by the number of items that fit on the page. For that reason, instead of using regular *precision* and *recall*, we use *precision*@$k$ and *recall*@$k$, where @$k$ is $k$ the most relevant items.

#### 7.1.2.3 ROC and AUC for a recommender system

When the number of recommended items is not given, one way of calculating the optimal threshold of numbers of items is by using a receiver operating characteristic (ROC) curve. ROC curve measures the difference between True Positive Rate (TPR) and False Positive Rate (FPR) based on a threshold value, which in our case is the number of recommended items. Formulas for calculating TPR and FPR are defined in equations 7.7 and 7.8 respectively. An example of a ROC curve is shown in figure 7.1.

$$TPR = \frac{TP}{TP + FN} \tag{7.7}$$

$$FPR = \frac{FP}{FP + TN} \tag{7.8}$$

Recommendation systems are generally used in more than one place of a website. Therefore, the number of recommended items may vary. In that case, picking the best model may be complicated due to the different performance of the models based on the number of items. One model may better perform on a small number of recommendations but worse on a larger number. To compare these models, we can use the Area Under Curve (AUC). The area is calculated based on a ROC curve. The value of the AUC varies from 0 to 1, where 1 means the model can perfectly separate the dataset. In the case of a recommendation system, it will measure the performance of separating used and not used items for a different number of suggested items.

### 7.1.3 Ranking accuracy metrics

Classification metrics do not take into consideration the order of items. That may not be an issue when the number of recommendations is small, such as

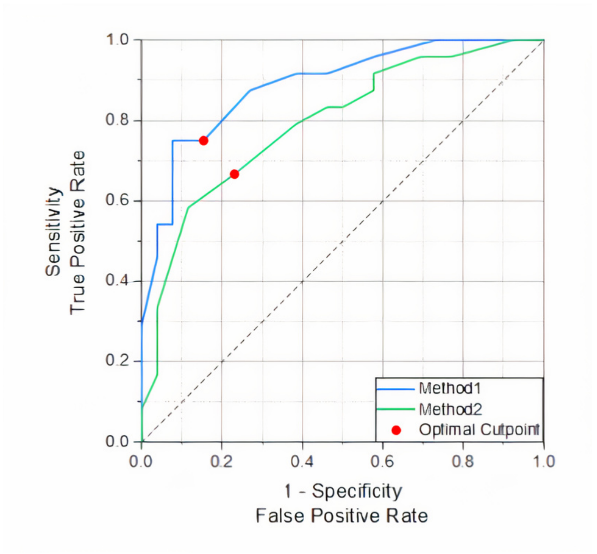


Figure 7.1: An example of ROC curves. Source: [36]

three or five. However, when a recommendation list contains more than ten items, a user may miss suggestions at the end of the list.

### 7.1.4 Relevance calculation for a recommendation task

To measure the model's ordering performance, we need to calculate the relevance of a predicted item. The relevance of an item may be defined as an importance of a predicted item to a user at a certain time. Measuring the exact relevance of an item is probably not possible since the user may not even know of the existence more relevant item. However, we can at least approximate the relevance based on historical data.

The ordering metrics originally come from learning to rank tasks, where an information retrieval system has a set of documents $D = \{d_1, d_2, ..., d_N\}$ and based on a query $q$ the system tries to order documents so that the most relevant ones are at the beginning of the list and the least relevant are at the end of the list. The ordering is based on the score of a relevance function, which gives each document a score based on a query. For instance, given a set documents {*"Attention is All you Need"*, *"Cat"*, *"Dog"*} and a query *"attention"* it will give the highest relevance score to *"Attention is All you Need"*, since it contains the word "Attention".

Since in a recommendation task, the query is not given, but rather an ID of a user, one way of calculating the relevance is based on a binary classification of relevant and irrelevant items. An item that is recommended and used will have a relevance score of 1, and an item that was recommended and was not used will have a score of 0.

#### 7.1.4.1 Normalized Discounted Cumulative Gain

The Normalized Discounted Cumulative Gain (NDCG) is a relation between a Discounted Cumulative Gain (DCG) of a prediction and Ideal Discounted Cumulative Gain (IDGC). Given the task of recommending top-k items, DCG can be calculated as it is shown in equation 7.9, where $rel_i$ means the relevance of $i$-th item. Then calculation of IDCG is defined in equation 7.10, where $I(k)$ represents the ideal list of items up to position $k$. Finally, the NDCG is calculated as it is shown in equation 7.11.

$$DCG_k = \sum_{i=1}^{k} \frac{rel_i}{log_2(i+1)} \tag{7.9}$$

$$IDCG_k = \sum_{i=1}^{|I(k)|} \frac{rel_i}{log_2(i+1)} \tag{7.10}$$

$$NDCG_k = \frac{DCG_k}{IDCG_k} \tag{7.11}$$

#### 7.1.4.2 Mean Reciprocal Rank

Mean Reciprocal Rank (MRR) focuses on the position of the first relevant item in the recommended list. Calculation of reciprocal rank (RR) is shown in the equation, where $i_u$ is the position of the first relevant recommendation in the recommended list for a user $u$. Then MRR is calculated as an average RR for all users, as it is shown in equation 7.13.

The metric, as shown in equations, focuses only on the ordering of the first item and does not take into consideration for the ordering of other items, which makes it less informative compared to NDCG.

$$RR = \frac{1}{i_u} \tag{7.12}$$

$$MRR = \frac{1}{|U|} \sum_{u \in U} \frac{1}{i_u} \tag{7.13}$$

#### 7.1.4.3 Average Precision

When precision (*Precision*) gives an insight into the overall classification performance of the model and does not take into consideration ranking, average precision (AP) does. The formula for calculating AP is shown in equation 7.14.

$$AP = \frac{1}{k} \sum_{i=1}^{k} \frac{rel_i}{i} \tag{7.14}$$

### 7.1.5 Other methods

The recommendation task is complex and cannot be accurately evaluated based only on previously mentioned metrics. The metrics do not assess aspects such as catalog coverage, novelty, diversity, temporal evaluation, and others. Yet, these metrics might be ambiguous and require domain knowledge to interpret them. For instance, one domain may imply high novelty and an another domain that have low novelty may not necessarily mean poor quality of a recommendation system.

#### 7.1.5.1 Catalog coverage

Catalog coverage is the ratio of uniquely recommended products to the total number of products. The formula for calculation is defined in equation 7.15, where $U$ is a set of all users and $R_u$ is a set of suggested items for user $u$.

$$Catalog\ Coverage = \frac{|\bigcup_{u \in U} R_u|}{|I|} \tag{7.15}$$

#### 7.1.5.2 Novelty

The novelty is determined by the uniqueness and innovativeness of the recommendations. Novelty has various definitions in the literature. Baeza-Yates and Ribiero-Neto [37] define it as it is shown in equation 7.16, where $R_u = K_u \cup S_u$ is a set of recommended items for a user $u$ and $K_u, S_u$ are set of known and unknown items for the user $u$. The measure also might be calculated as an average novelty of each recommended item $i$ as defined in equation 7.17 or 7.18, where $U_i$ is the set of users that was recommended item $i$, $U$ is a set of all users, and $I_R$ is a set of all recommended items.

$$Novelty = \frac{1}{|U|} \sum_{u \in U} \frac{|K_u|}{|R_u|} \tag{7.16}$$

$$Novelty = -\frac{1}{|I_R|} \sum_{i \in I_R} log_2 \frac{|U_i|}{|U|} \tag{7.17}$$

$$Novelty = \frac{1}{|I_R|} \sum_{i \in I_R} (1 - \frac{|U_i|}{|U|}) \tag{7.18}$$

#### 7.1.5.3 Diversity

Diversity measures how narrow or wide the range of recommended products is. The recommender system that suggests only the music of one artist might be considered as narrow. One way of calculating diversity is using the similarity function of two items. The formula for calculation is defined in equation 7.19, where $U$ is a set of users and $R_u$ is a recommendation set for a user $u$.

$$Diversity = \frac{1}{|U|} \sum_{u \in U} \sum_{i,j \in R_u, i \neq j} sim(i,j) \tag{7.19}$$

#### 7.1.5.4 Hit Ratio

Hit ratio is a common metric for measuring the prediction performance of a recommendation system. Given a set of recommendations $\hat{R}_u$ for a user $u \in U$ and a set of the used items $R_u$, the "hit" occurs when the sets have not empty intersect. Averaged hit measure across all users is called hit ratio, as defined in equations 7.20 and 7.21.

$$h(\hat{R}_u) = \begin{cases} 1 & \text{if } \left|\hat{R}_u \bigcap R_u\right| > 0 \\ 0 & \text{otherwise} \end{cases} \tag{7.20}$$

$$Hit\ ratio = \frac{1}{|U|} \sum_{u \in U} h(\hat{R}_u) \tag{7.21}$$

## 7.2 Online evaluation

Online evaluation is different from offline evaluation. The goal of online evaluation is to measure the observed satisfaction of the user in real-time rather than the supposed interaction with a recommended list. Defining the exact satisfaction measurement is probably not possible. However, online testing is at least trying to approximate it by using a variety of measures.

### 7.2.1 Click-through rate

Click-through rate (CTR) measures the number of clicks garnered by the recommendations. The formula for calculating CTR is shown in equation 7.22.

$$CTR = \frac{number\ of\ accepted\ recommendations}{number\ of\ recommendations} \tag{7.22}$$

The recommendation is accepted when the target action has occurred, such as click, comment, purchase, etc. The underlying assumption is that if more people click on the recommended items, the suggestions are more relevant to them. That may not be the case when the user sees only recommendations or when recommended items take up most of the space on the website. In that case, the user does not have a choice and clicks only on recommended items, resulting in high CTR.

### 7.2.2 Temporal diversity

Temporal diversity assesses the ability of a recommendation system to change over time. A model that always generates the same item suggestions may not be welcomed by a user. In many domains, users expect a recommendation system to produce new recommendations. To measure the temporal changing performance, the recommender system needs to generate multiple recommendation lists over different time periods. One way of calculating the metric is defined in equation 7.23, where $R_{t,u}$ is a list of $N$ recommended items to a user $u$ at time $t$ and $k$ is a number of time intervals.

$$\textit{Temporal diversity} = \sum_{u \in U} \frac{\left| \{ \bigcup_{t=0}^{k} R_{t,u} \} \right|}{k \cdot N} \tag{7.23}$$

### 7.2.3 Adoption and conversion rates

While the CTR can detect user attentiveness or interest, it cannot tell if users genuinely liked the recommended item they clicked. Therefore, different adoption metrics should be utilized to further assess the recommendations' efficiency.

Adoption mechanisms are domain-specific. It should be developed by a domain expert. The adoption mechanisms may include dwell time (time a user spends on an article or result after initially clicking it), scrolling time, user activity on an item, such as commenting, adding feedback. Generally, systems combine the mentioned mechanisms. For instance, in a news portal, these metrics may include dwelling time and scrolling. Relying only on dwelling time will lead to incorrect relevance measures, since a user may open a tab, and then switch to another tab and not come back.

### 7.2.4 Sales and Revenue

Another way of online evaluation may include the growth (or shrinking) of sales and revenue. However, the indicator is highly correlated with other factors that affect sales and revenue, and it is challenging to differentiate between them. Moreover, if the recommended items are rarely used by users, then improvements in the performance of the recommendation model will not affect sales or revenue.

### 7.2.5 Sales distribution

The recommender system may affect the distribution of items that have undergone a certain action, such as a click or purchase in both ways: positively and negatively. A study [38] revealed that the introduction of a recommender system for premium cigars led to a significant shift in consumers' purchasing behavior. The individualized recommendations, in particular, resulted in increased purchases in the long tail, and the sales spectrum was no longer dominated by a few best-sellers. However, as shown in study [39], recommendation systems, in particular collaborative filtering, may ultimately help to boost sales of already popular items.

### 7.2.6 User behaviour and engagement

Introducing a recommender system may affect user behavior and engagement, in particular it may increase user retention, which is often directly connected with business value [40].

In the news domain, for instance, study [41] observed longer sessions when a recommender was in place. In particular, authors reported that the visit lengths were 2.5 times higher when recommendations were shown on the page.

In the music streaming domain, in study [42], authors compared different recommendation strategies and found that a recommendation strategy that combined user behaviors and content data led to a 50% increase in activity level in terms of playlist additions.

### 7.2.7 A/B testing

A/B testing (also known as split testing or bucket testing) in machine learning is a method of comparing models against each other to determine which one performs better. During the testing, traffic is generally split equally to measure the performance of the individual models. The measuring is mostly conducted using CTR rate with adoption mechanisms.

To gather reasonable results, A/B testing should be conducted over a period of time on a sufficiently large number of users. The testing cannot be carried out only during one day since statistical noise may be too high.

Another issue with the testing is distributing the traffic evenly. The traffic cannot be split based on location or a certain period since it may lead to biased results. During the testing, the distribution should be random and all of the splits need to be tested at the same time.

Chapter 8

# Analysis and design

This chapter is devoted to the implementation and evaluation of different recommender systems. It consists of three parts. The first part starts with dataset description and preprocessing, then defines evaluation metrics for the dataset. The second part describes various implementations and their performance. The last part compares the implementations and selects the most suitable model for the Data Dictionary domain.

### 8.0.1 Dataset description

The dataset contains about two years of user behavioral data and the set of all items, including items without any interaction. The dataset came from a data dictionary application used by a client in the financial sector. Users of the applications are employees of the financial institution. The users can be categorized into three groups: data producers, data developers, and data consumers. Producers include IT engineers, architects, and developers, who design, implement and optimize systems. Data developers are a group of different analytics, such as data, BI, IT analytics, data scientists, etc. The last group of data consumers includes specialists or managers who consume analytical outputs and insights. The system is generally used by the first two groups and only occasionally by the group of data consumers.

The set of items includes the model, table, and view data. The user behavior dataset contains column interaction information, but the column data is not provided. However, based on user behavior data, it is possible to find the tables and models which the columns belong to.

The provided interaction set consists of two action types: clicked and searched items. It also contains a timestamp that indicates time when the action happened. Furthermore, based on parentness information, it is possible to generate more action types, such as indirect clicks. More about that in the preprocessing section.

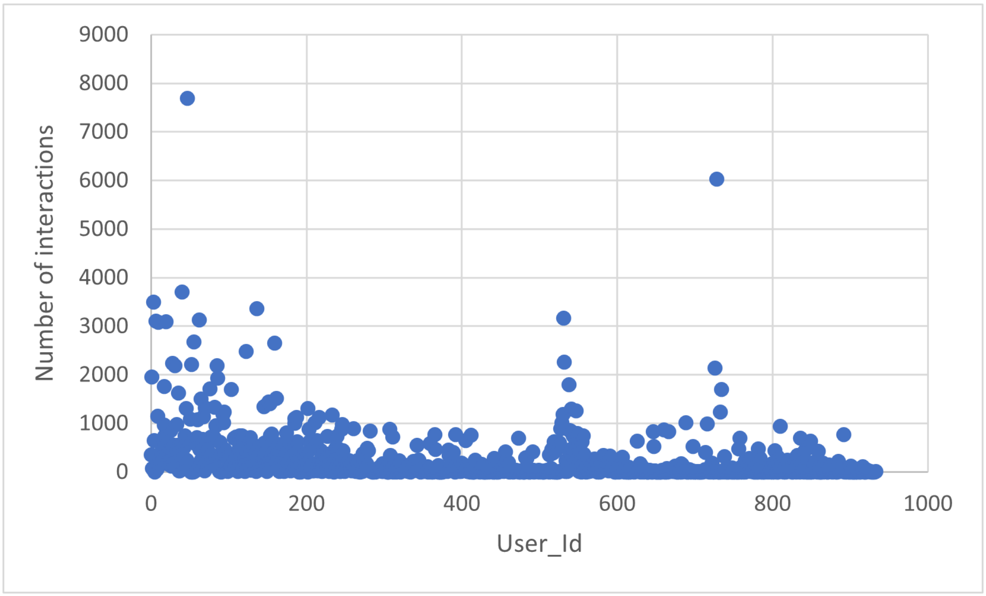


Figure 8.1: Number of interactions per user ID

#### 8.0.1.1 Dataset statistics

The number of unique users who used the application is 933, with an average number of interactions of 253.5 and median 42. Users have their unique ID assigned that had not changed through the period. Based on the mean number of interactions, the dataset contains 232 active users (users that had more than the average number of interactions). The number of interactions per user ID is shown in figure 8.1.

The dataset of items contains 172974 entities. The number of visited items is 25579, which is about 14.8% of the total number. More than 95% of the items have description information, which may be useful in finding similarities between objects. The average number of interactions per item is 9.86, and median is 3. The number of items that have more than the average number of interactions is 4304, which is more than 16.8% of the total number of visited items. Number of interaction per item ID is shown in figure 8.2.

The total number of interactions is 233427. More than 11% of that number are search interactions that were recorded when a user typed a keyword in a search engine, selected an item, and clicked on it. Others are interactions that were recorded when a user clicked on an item. An overview of the monthly usage of the application is shown in the figure 8.3.

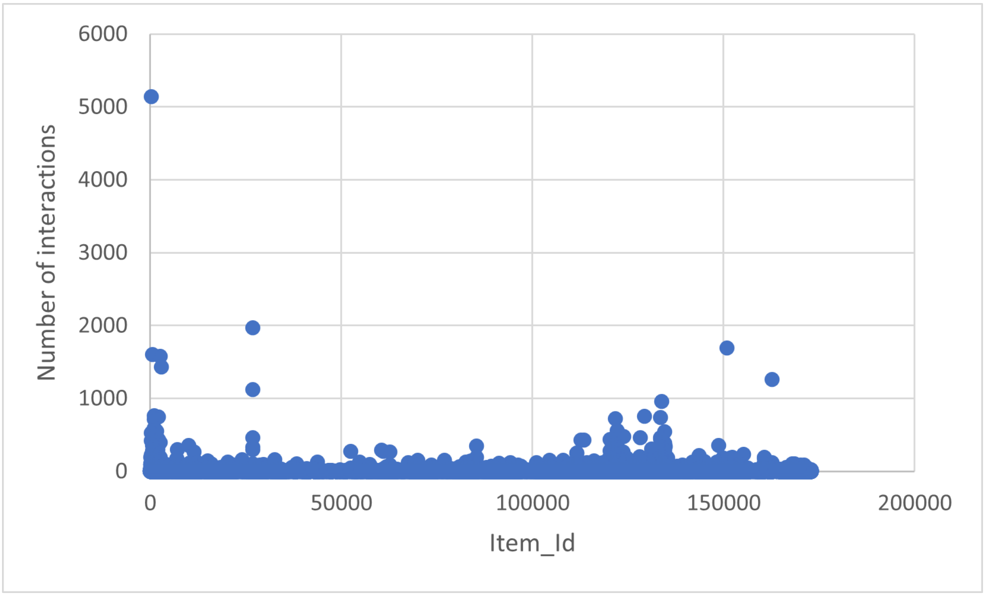


Figure 8.2: Number of interactions per item ID

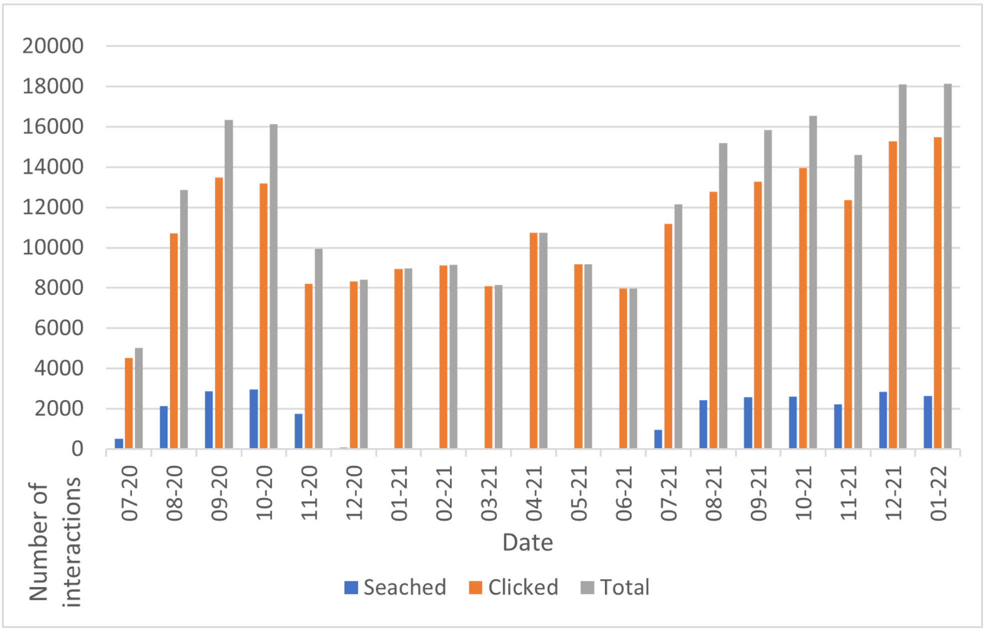


Figure 8.3: An overview of the monthly usage of the application

| **Attr_Name** | **Attr_Type** | **Is_Nullable** | **Example** |
|---|---|---|---|
| Action_Type_Id | Int | NO | 1 |
| Object_Type_Id | Int | NO | 2 |
| Model_Code | String | NO | CRM |
| Object_Code | String | YES | Customer |
| Sub_Object_Code | String | YES | Customer_Id |
| Accessed_Date | Datetime | NO | 2022-01-25 10:19:27 |
| User_Id | Int | NO | 42 |

Table 8.1: The columns of the interaction dataset

### 8.0.2 The metadata structure of given dataset

The interaction dataset consists of seven columns: *Action_Type_Id*, *Object_Type_Id*, *Model_Code*, *Object_Code*, *Sub_Object_Code*, and *Acessed_Date*. *Action_Type_Id*, *User_Id* is the type of the interaction, such as searched or clicked. *Object_Type_Id* is a value from a range of four, where values from one to four refer to the model, table, view, and column correspondingly. *Model_Code* is a code of the item's model. Note that the item may be a model itself, i.e. a user accessed the model item. *Object_Code* is a code of the table or view. *Sub_Object_Code* is a code of a column. It was filled when a user accessed a column. *Acessed_Date* is time when the interaction has occurred. Lastly, *User_Id* is an ID of a user who created the interaction. Other metadata information about the set is shown in table 8.1.

The item dataset consists of four columns: *Object_Type_Id*, *Model_Code*, *Object_Code*, *Comment*. The first three columns are the same as in the interaction dataset, but *Object_Type_Id* contains only values corresponding to the model, table, and view, meaning the column information in the dataset is missing. The only column that differs the dataset from the interaction dataset is *Comment*. The column shortly describes the meaning of the item. It is common to comment on all of the existing items in the system since the items may not always have a clear meaning. For instance, the abbreviation CRM may refer to Customer Relationship Management or Client Relationship Management. More information about the item dataset is shown in table 8.2.

### 8.0.3 Defining goals for a recommender system

As was mentioned in chapter 3, the developers of the data dictionary application are intended to integrate a recommendation system. The recommendation system will used to predict next visited item, which may be are a model, a table, or a view. Note that columns are not in the list. The number of recommended items may vary from three up to twenty-five, where twenty-five is the number of items that generally fit on a display.

| **Attribute_Name** | **Attribute_Type** | **Is_Nullable** | **Example** |
|---|---|---|---|
| Object_Type_Id | Int | NO | 2 |
| Model_Code | String | NO | CRM |
| Object_Code | String | YES | Customer |
| Comment | String | YES | The main table for storing customer information |

Table 8.2: The columns of the items dataset

Since the user interaction dataset includes up to nineteen months of historical activity for each user, the recommender system should capture long-term patterns in the user's taste. On the other hand, a user's interests may change over time, and the recommendation system also should be able to adapt to it.

### 8.0.4 Defining evaluation metrics

The evaluation strategies will include only offline evaluation since online evaluation demands conducting it over a long period, such as months. During the offline evaluation, the author of the thesis focuses on classification, ranking, and other metrics that are specific to the recommendation task.

The classification metrics will focus on recall rather than precision because, for the task, the number of not recommended but used items (false negatives) matters more than the number of recommended but not used (false positives). The ROC metric will be used to find an optimal number of recommended items and AUC for comparing the models.

The ranking performance of the models will be evaluated using NDCG. The ability of a recommender system to correctly rank the suggested results becomes more critical when the number of recommended items increases since a user may not even notice the relevant items at the end of a suggested list.

The metrics that are specific to the item recommendation task will include novelty and catalog to give an insight into the personalization performance and overall item coverage. In addition, the performance of a recommender system will be also measured by the hit ratio, because the measure is commonly used in recommender system evaluation.

### 8.0.5 Data preprocessing

Before designing and building the models, each model needs to be preprocessed. Different models will require special preprocessing. Still, the given dataset might be preprocessed generally to remove unwanted rows that were occasionally added to the dataset, to separate items and interaction datasets,

| **Attr_Name** | **Attr_Type** | **Is_Nullable** | **Example** |
|---|---|---|---|
| Item_Id | Int | NO | 17 |
| Object_Type_Id | Int | NO | 2 |
| Name | String | NO | CRM Customer |
| Comment | String | YES | The main table for storing customer information |

Table 8.3: The columns of the items dataset after the preprocessing

to remove tags from text attributes, etc. The following process will include item and interactions preprocessing.

#### 8.0.5.1 Item dataset preprocessing

Since the interaction set has some missing item information, the item set was enhanced by the missing data from the interaction dataset. Then the model code and the object code were joined to form the object name. Afterwards *Item_Id* was generated based on a row number. Finally, from the *Comment* column the HTML tags were removed. The metadata structure of the item dataset after the preprocessing is shown in table 8.3.

#### 8.0.5.2 Interaction dataset preprocessing

Since the interaction dataset contains a model, object, and subobject code, it is possible to generate additional transaction types, which may be useful in collaborative filtering or a content-based model. Based on parentness information, the author created two additional interaction types: indirect second-level and third-level access. The interactions with second-level access are generated based on interactions that contained a column, a table, or a view. For column interactions, additional interaction with its parent (a view or a table) is created. When interaction occurred with a view or a table, it will generate an auxiliary interaction with its model. The interactions with third-level access are generated only for column interactions. When a column is accessed, the auxiliary third-level access interaction with its model is created.

After the auxiliary interaction types were generated, *Item_Id* was created based on joined values of *Model_Code* and *Object_Code* and an item dataset. Lastly, the column *Sub_Object_Code* was dropped since it will not be used in recommender systems and *Interaction_Id* was added. The metadata structure after the preprocessing is shown in table 8.4.

| **Attr_Name** | **Attr_Type** | **Is_Nullable** | **Example** |
|---|---|---|---|
| Interaction_Id | Int | NO | 12 |
| User_Id | Int | NO | 42 |
| Item_Id | Int | NO | 17 |
| Action_Type_Id | Int | NO | 1 |
| Accessed_Date | Datetime | NO | 2022-01-25 10:19:27 |

Table 8.4: The columns of the interaction dataset after the preprocessing

#### 8.0.5.3 The dataset split

Once the preprocessing was finished, the train, validation, and test datasets were created based on the interaction dataset with searched and clicked items (artificially created interactions will only be used while training). For the split, the author had chosen Leave Only Last Item strategy. As the name suggests, it extracts the last part of user transactions to the test set and the second last part to the validation set. Since the number of predicted items will be up to twenty-five, the size of the parts will be equal to that number.

The used split strategy has a downside of potential feature leaking since the interactions are not strictly split by time. For instance, the model may learn the popularity of an item before it becomes popular. However, the strategy maximizes the number of interactions that can be used for training.

The created validation and test sets have 9625 interactions, and the training dataset has 450502 interactions. It may seem that about 2% of the total number of interactions were taken for test and validation datasets each. However, the validation and test datasets include about 41.2% of the total number of unique users each. In other words, the training dataset contains historical transactions of 933 users. Whereas the test and the validation dataset contain the last twenty-five transactions of 385 users.

### 8.0.6 Baseline models

In this section, the author describes the implementations and evaluation results of two baseline models. The baseline models serve as a benchmark against other more complex approaches that will be described later.

#### 8.0.6.1 Baseline model 1

The first baseline model is based on item usage frequency across all the users. The model always returns the same set of recommended items for all users. The algorithm is shown in listing 8.1. In the listing *k* is a number of predictions, *df_interactions* is a pandas [43] dataframe of all interactions, *Counter* [44] is a class that for each element counts its frequency. In addition, this class

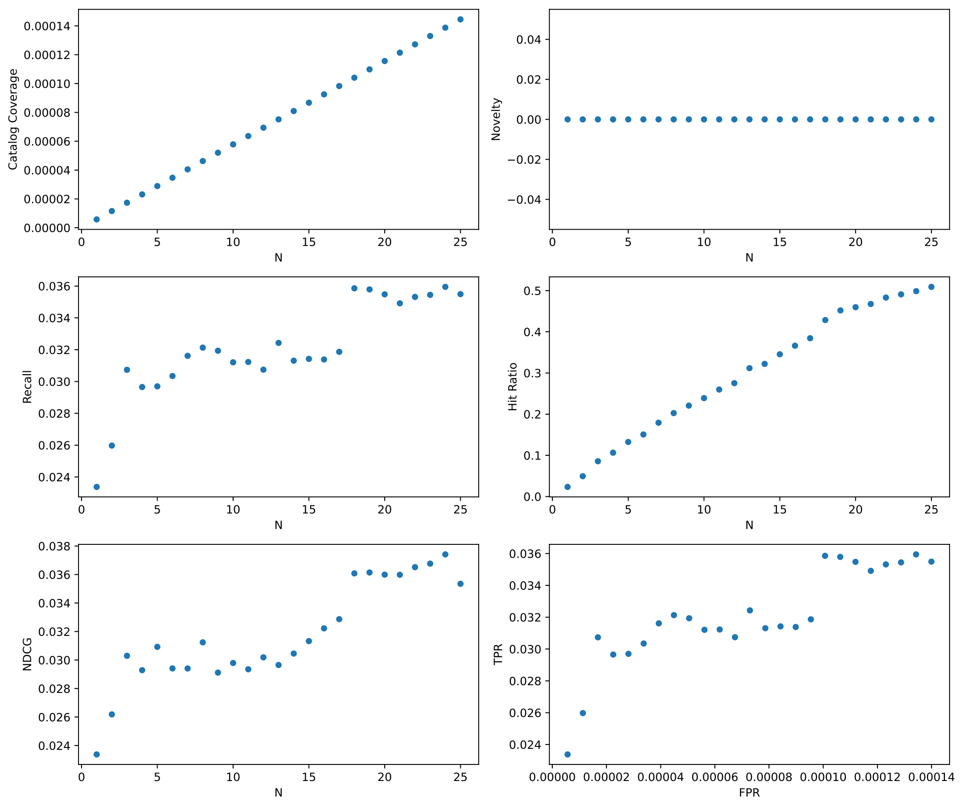


Figure 8.4: The evaluation results of the Baseline model 1

has method *most_common()* that returns items sorted by their frequency from the highest to the lowest.

Code Listing 8.1: The prediction function of the Baseline model 1

```
from collections import Counter

def get_predictions(df_interactions, k):
  most_common = Counter(df_interactions['Item_Id']).most_common()
  return most_common[:k]
```

The evaluation results tested on the validation set are shown in figure 8.4. The average recall is about 3.2%, and the NDCG score is nearly 3.1%. These results can be considered decent. However, novelty results and temporal changes over time in the method are close to zero.

### 8.0.6.2 Baseline model 2

The second baseline model is similar to the first one. The only difference is that it gives more personalized recommendations. The prediction function,

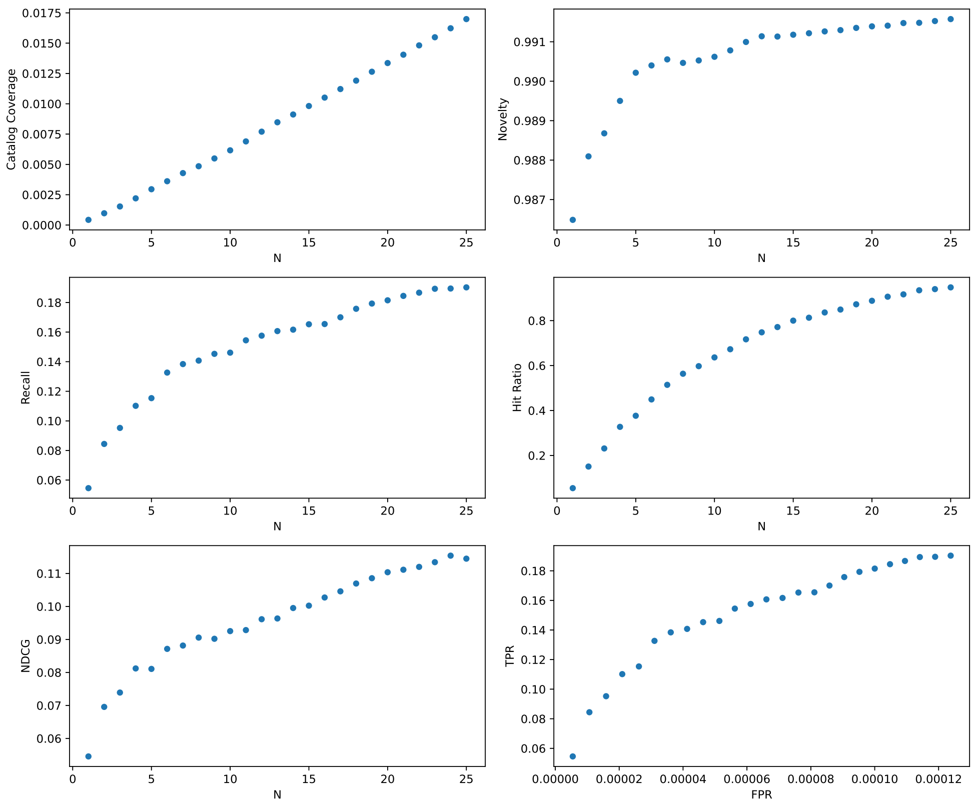


Figure 8.5: The evaluation results of the Baseline model 2

which is defined in listing 8.2, instead of returning *top-k* the most frequent items across all users, returns $k$ the most frequent user items.

Code Listing 8.2: The prediction function of the Baseline model 2

```python
from collections import Counter

def get_predictions(user_id, df_interactions, k):
  df_user = df_interactions[df_interactions['User_Id']==user_id]
  user_item_ids = df_user['Item_Id']
  most_common = Counter(user_item_ids).most_common()
  return most_common[:k]
```

The performance of the second model on the validation set is shown in figure 8.5. The model has a strong performance of an average of about 15%. The average value of NDCG across all $k$ is about 9.6%. The baseline model has a hit ratio of more than 94% at $k = 25$ on the validation set. Since the recommendations are generated for each user based on the most frequent user items, the model has a strong novelty score too. The only issue with the model is that it will not be able to keep up with changes in user preferences in real-time.

The author also measured the results of the same algorithm but without additionally generated user interactions (the process of the generation was described in the preprocessing section). The average recall went down by about 1% and other metrics were almost the same.

### 8.0.7 Content based model

Since the item dataset was provided, it is possible to build a content-based recommendation system. The clear benefit of the CB model is that generated results of the approach are not biased towards the most popular items and it is able to satisfy unique user interests. On the other hand, as it will be shown in the evaluation section, biased results do not necessarily signify poor quality of a recommender system.

#### 8.0.7.1 Building item profiles

Item profiles were generated using a trained model from the Sentence-BERT framework [45]. The framework contains state-of-art models for generating sentence, text, and image embeddings. In particular, the *paraphrase-multilingual-mpnet-base-v2* and *all-mpnet-base-v2* were used, both of them have one of the best performances. The first model was trained on sentences in more than 50+ languages, including English and Czech. The second model was trained solely on English sentences and has a bit stronger performance compared to the first one on English sentences.

The reason why multi-lingual was chosen is that the *Comment* column has descriptions in English and Czech. The multi-lingual model in theory should have better performance for the given dataset (disclaimer: it will not) because embeddings of similar sentences in different languages should have high similarity. For instance, given the sentence “That is a happy person” and the following sentences “To je šťastný člověk”, “That is a very happy person”, and “Today is a sunny day”, the output embeddings will have cosine similarities of 88.9%, 98.2%, and 45.1% correspondingly. Whereas the second model will output cosine similarities of 6%, 96.7%, and 31.5% correspondingly.

The output dimensions of the generated embeddings are 768 dimensional dense vector. The maximum possible number of words in the input is 384, longer sentences are truncated automatically. However, it is not an issue with the provided dataset since the maximum number for the words *Name* and *Comment* columns is 191.

#### 8.0.7.2 Building user profiles

The first step in building users’ profiles is rating generation. The process of rating generation is defined in listing 8.3. After ratings and item vectors embeddings are generated, user profiles are calculated. The process of the calculation is defined in listing 8.4, where *embeddings_by_id* is a dictionary of item embeddings grouped by *item_id*, *user_ratings* is a dictionary of *item_id* and *rating* pairs. In the listing the author used dot function from CuPy [46] library, which is GPU-accelerated analog of NumPy [47]. Thanks to that library, the user profile calculation took only 3 seconds on Tesla P100-PCIE-16GB GPU.

Code Listing 8.3: The process of rating generation

```
def calc_user_ratings(df_user_iteractions):
  ratings = {}
  for idx, row in df_user_interactions.itterrows():
    if row['Item_Id'] in ratings:
      ratings[row['Item_Id']] += 1/row['Access_Level_Type_Id']
    else:
      ratings[row['Item_Id']] = 1/row['Access_Level_Type_Id']

  # min-max normalization
  min_v = get_min_rating_value(ratings)
  max_v = get_max_rating_value(ratings)
  res = dict((k, ((v-min_v)/diff)) for k,v in ratings.items())
  return res
```

Code Listing 8.4: The user profiles generation

```
import cupy as cp
from cupy import array as arr

def calc_user_embedding(user_ratings, embeddings_by_id):
  ratings_vec = arr([rating for rating in user_ratings.values()])
  embeddings_vec = arr([embeddings_by_id[item_id] \
                        for item_id in user_ratings.keys()])
  res = cp.dot(ratings_vec, embeddings_vec)
  return res
```

#### 8.0.7.3 Prediction

The item prediction was implemented using cosine-similarity. To speed up the process, instead of calculating the similarity between each item and a user vector, CuPy dot product was used. After a similarity vector was calculated, the vector was sorted using CuPy *agrsort()* function. Thanks to that, the average time of getting *top-k* predictions was about 7.5 seconds for 385 users using the same GPU, as it was described earlier.

#### 8.0.7.4 Evaluation

The model shows better performance on a validation set compared to the first baseline model. However, it has a weaker performance compared to the second baseline. The evaluation results are shown in figure 8.6. The average recall and NDCG across all k from 1 to 25 are about 4.8% and 4.4% correspondingly.

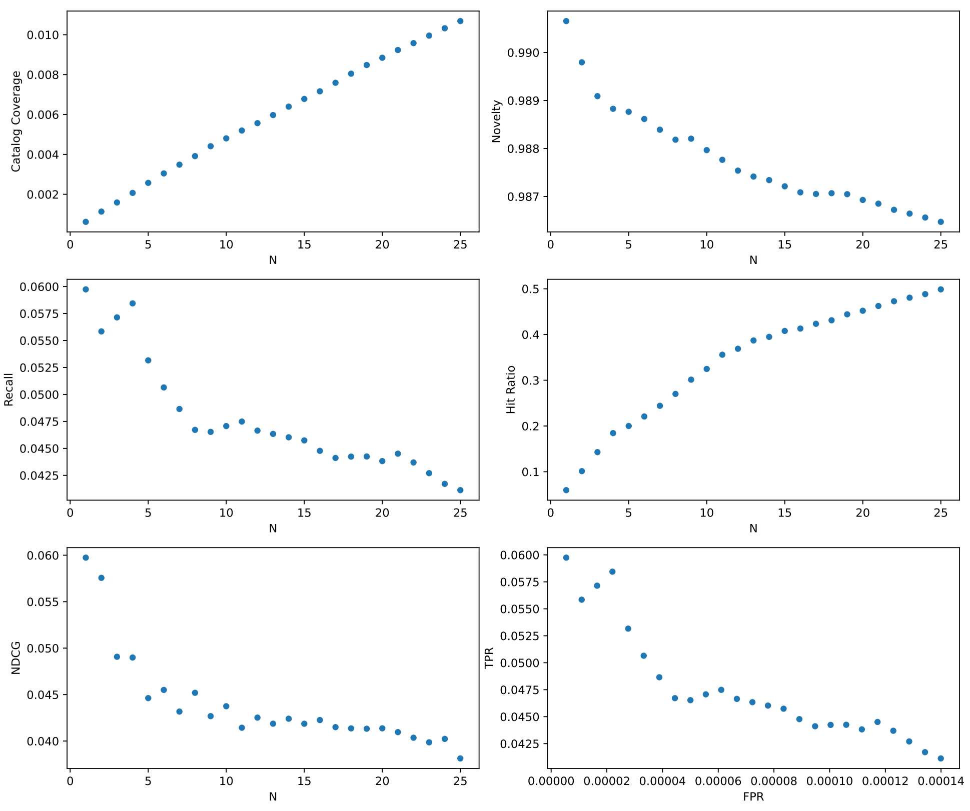


Figure 8.6: The evaluation results of the content-based model

As it was mentioned above, two approaches were tested for item embedding generation. Even though the *Comment* column has values in Czech and English, the multi-lingual approach had a slightly weaker performance. In particular, it had about 18% decrease in the NDCG and 9.5% decrease in recall.

Even though the model has weaker performance than the second baseline, it should better adapt to changes in user interests. Nonetheless, this is only a hypothesis, which may be proven or disconfirmed during online testing.

### 8.0.8 Collaborative filtering based model

Despite having a set of visited and searched items, it is still possible to build a collaborative filtering model based on matrix factorization. The process of building starts with implicit rating generation. Then the author used Tensorflow [48] for creating the user and item latent spaces. For training adam [20] optimizer was used with the standard learning rate of 0.001. RMSE was chosen as a loss function. Additionally, early stopping was added to prevent overfitting.

### 8.0.8.1 Ratings generations

In the CF approach three methods for rating generation were used. The first one is the same as it was described in the CB model. The second method altered the coefficients for different action types. The second method is shown in listing 8.5, where keys of $1, 2, 3, 4$ in *coeff_map* correspond to searched, clicked, indirectly clicked, twice indirectly clicked accordingly. In the last method, the author additionally added aging. The interaction aging was calculated as a difference between the latest date and the interaction date in months. Despite making the ratings more dynamic and time relevant, the second method showed slightly better results than the first and last one. The aging did not improve the results, because in the domain of the data dictionary the change in users' interests is probably not rapid. Hence, the model that returns more popular items lands up with better performance.

Code Listing 8.5: The process of rating generation

```
def calc_user_ratings(df_user_iteractions):
  ratings = {}
  coeff_map = {1:1,2:3,3:10,4:100}

  for idx, row in df_user_interactions.itterrows():
    if row['Item_Id'] in ratings:
      ratings[row['Item_Id']] += 1/coeff_map[row['
   Access_Level_Type_Id']]
    else:
      ratings[row['Item_Id']] = 1/coeff_map[row['
   Access_Level_Type_Id']]

  # min-max normalization
  min_v = get_min_rating_value(ratings)
  max_v = get_max_rating_value(ratings)
  res = dict((k, ((v-min_v)/diff)) for k,v in ratings.items())
  return res
```

### 8.0.8.2 Finding the optimal latent size

The matrix factorization has a hyperparameter $d \in \mathbb{N}$ that determines the size of the user $U_{emb} \subset \mathbb{R}^{n,d}$ and item $I_{emb} \subset \mathbb{R}^{m,d}$ embedding matrices, where $|U| = n$ and $|I| = m$. These matrices are multiplicated, that creates the approximation of a rating matrix $\hat{R} \subset \mathbb{R}^{n,m}$.

To find an optimal value of $d$, the model was trained with different values of $d$ in a range from 20 to 2000. After each training the model was evaluated for $N \in \{1, 2, ..., 25\}$. For each metric the author calculated an average of its values across different $N$. The results are shown in figure 8.7. The best scores of recall, NDCG, and the hit ratio were for $d = 250$. After the value, the total performance went down.

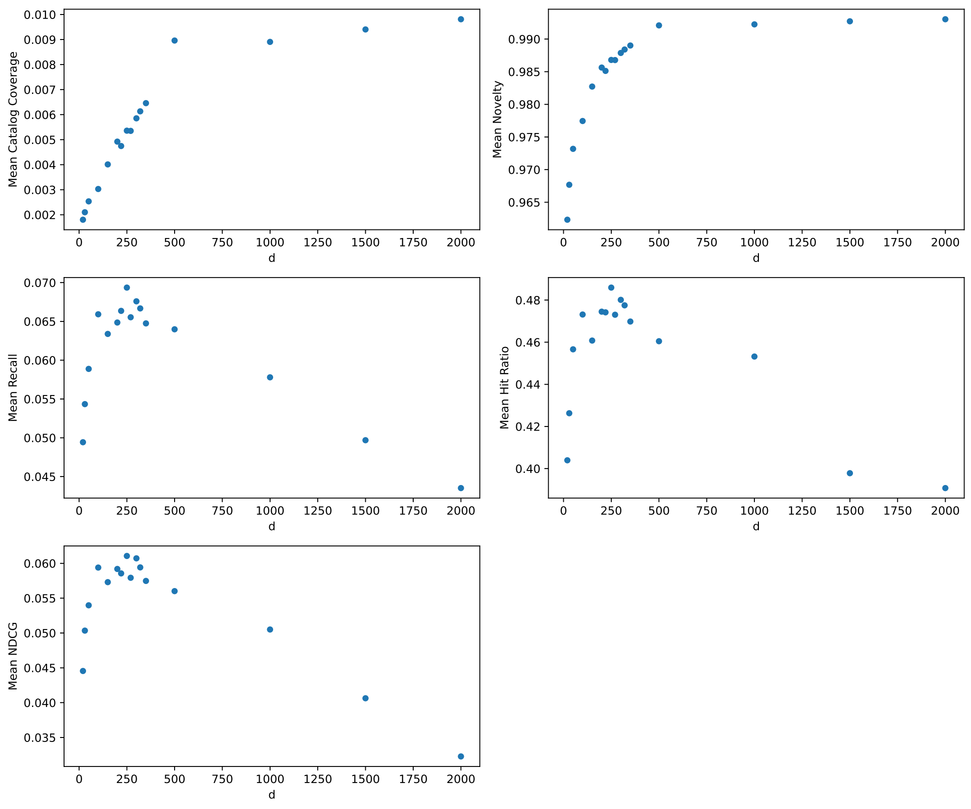


Figure 8.7: The performance of CF model for different $d$

#### 8.0.8.3 Evaluation

The evaluation of the CF model with $d = 250$ on a validation set is shown in figure 8.8. The overall evaluated model performance is better compared to the CB model. In particular, the average values of recall, NDCG, and hit ratio are increased by 140%, 169%, and 168% accordingly. However, the second baseline model is still ahead, especially in terms of recall.

### 8.0.9 Session based model

Another way of looking at the recommendation problem is the session-based recommendation approach that had became widespread in the last few years. In this section, the author of the thesis, in the context of a session-based recommender system for the data dictionary application, outlines different methods for session formation, building models using the Transformer4Rec library, and the evaluation of the models.

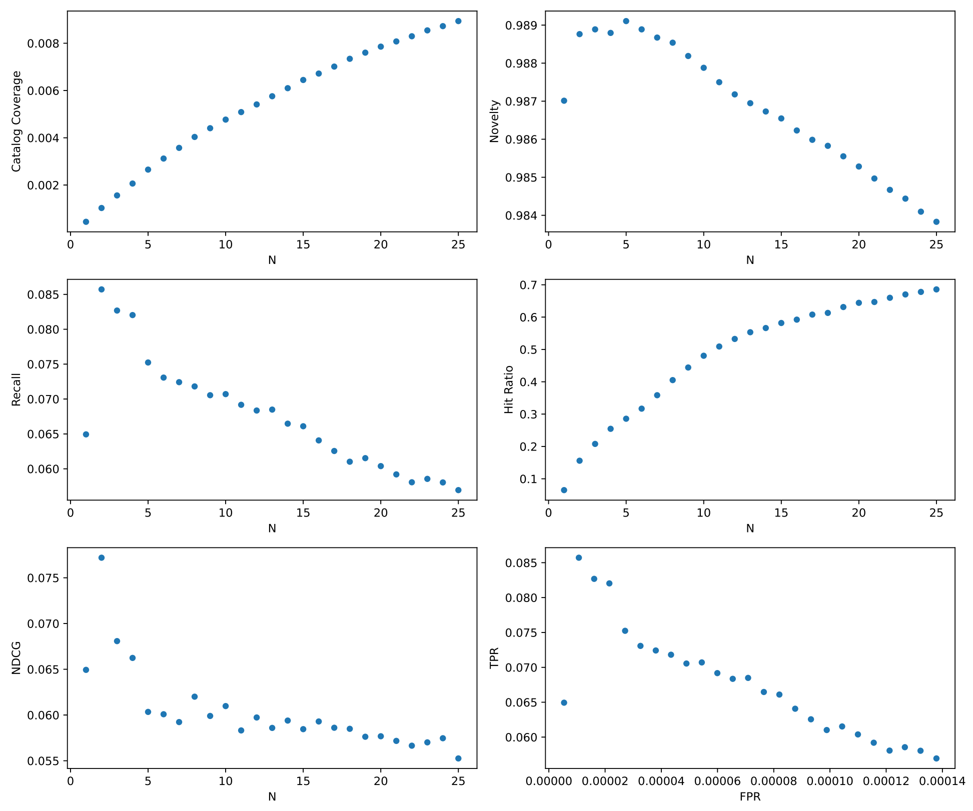


Figure 8.8: The evaluation results of the CF model

#### 8.0.9.1 Session formation

The first step in building the session-based recommender system is session creation. In e-commerce domains that allow anonymous interactions (purchases, clicks, etc.), the process of session formation is straightforward. When the user opens a page, e.g. in a private window, he gets assigned to a unique identification. Then all of the user interactions might be associated with the ID. After he/she closes the page, the system is no longer able to track the user.

Since in the data dictionary application all of the users are associated with a unique *User_Id* and the whole history of the user's interactions is stored, there is a variety of options for creating sessions. In the thesis, three different approaches were used.

The first approach was to split user interactions based on a user ID and a day of creation. In addition, if the number of interactions exceeds the maximum number, a new session was created. The Transformer4Rec library requires to define a maximum number of interactions. By default, sessions longer than that number are truncated. Instead of truncating, new sessions were generated.

Sessions that were formed based on a user ID and a day of creation are short, with a median length of 4 interactions. To increase the number of interactions in sessions, in the second approach, sessions were generated based on a user ID and a month of creation. In the approach, the median session length had increased to 17 interactions per session. These numbers had not changed even when the maximum session length had been altered to different values.

The last approach was to equally split the interactions set into an array of sessions based solely on a user ID with a variable limited maximum session length. The method is more flexible since it allows to configure the number of interactions in sessions. When the approach is used, the median session length is primarily dependent on the limit with the maximum median number of approximately 90 interactions per session.

Since the third approach allows for a richer configuration of the session length, the third approach was chosen to find an optimal maximum input session length.

#### 8.0.9.2 Session length optimization

The evaluation of different session lengths revealed that the model performs better on longer sessions. However, that does not mean that the model cannot make a prediction in short sessions. In fact, it was configured and trained to make predictions based on two and more interactions. The author tried different session lengths from 30 to 200. The larger number was not tested since the model performance started to weaken after the length of 120. The results are shown in figure 8.9. The best performance was achieved when the session length was equal to 120.

#### 8.0.9.3 Architecture overview

As it was mentioned above, the Transformer4Rec library allows using different transformer architectures, such as XLNet [34], GPT-2 [49], or RNN architectures like LSTM [50] or GRU [51]. In the thesis, the author focused on XLNet transformer architecture. The architecture has more than 10 hyperparameters. The main focus was on the following hyperparameters: dimensionality of the encoder layers and the pooler layer, number of hidden layers in the Transformer encoder, number of attention heads for each attention layer in the Transformer encoder, and the maximum session length. Other hyperparameters had not been changed and default values were used.

The optimization was conducted manually. Each parameter was optimized separately. After that, several candidates were picked based on knowledge of the meaning of the hyperparameters and evaluation results. These candidates were trained and evaluated. Finally, the best one was selected.

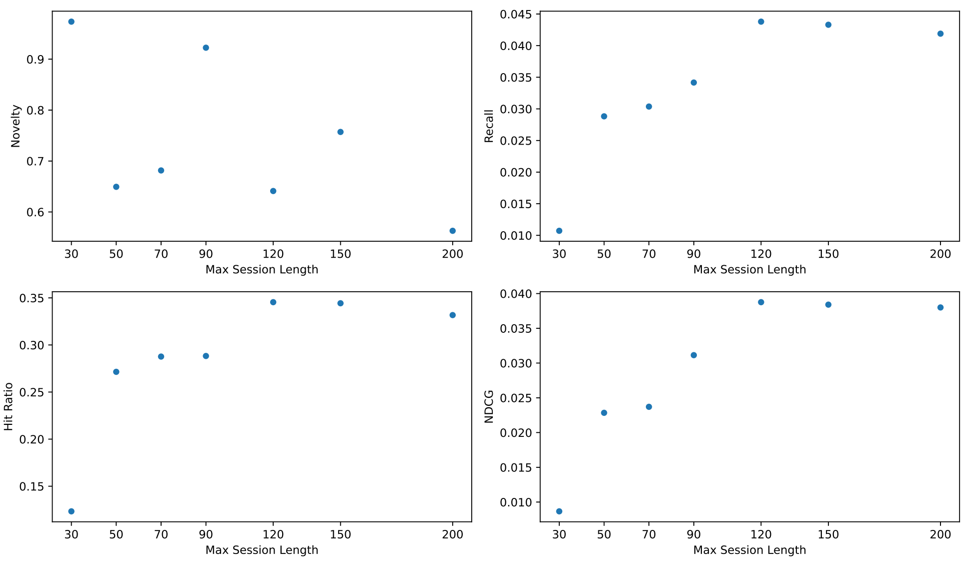


Figure 8.9: The performance of SB model for different session lengths

The optimization of the mentioned parameters revealed that more complex models require more training epochs. Otherwise, the models recommend the same items to all users. However, each model had a limit when the performance started to degrade after increasing the number of epochs. The optimal number of epochs for the simplest models was five. More complex models required approximately ten epochs. The most complex models had an optimal value of around fifteen epochs.

The best results were achieved with the model that had two hidden layers, two attention heads for each attention layer, and the dimensionality of the encoder layers and the pooler layer of sixty-four.

#### 8.0.9.4 Prediction process

Since one user may have multiple sessions in a dataset, the system always fetches the user's last session. After that, it feeds the session to the trained model and gets predictions for each item. The $k$ items with the highest probabilities are taken and shown to the users as recommended. The output list that is displayed to the user is sorted by the probability value from the highest to the lowest.

#### 8.0.9.5 Evaluation results

During the preprocessing that was described above, the searched and clicked behaviors were split into the train, validation, and test datasets. Sessions were formed using the training dataset. While evaluating, the predicted items that

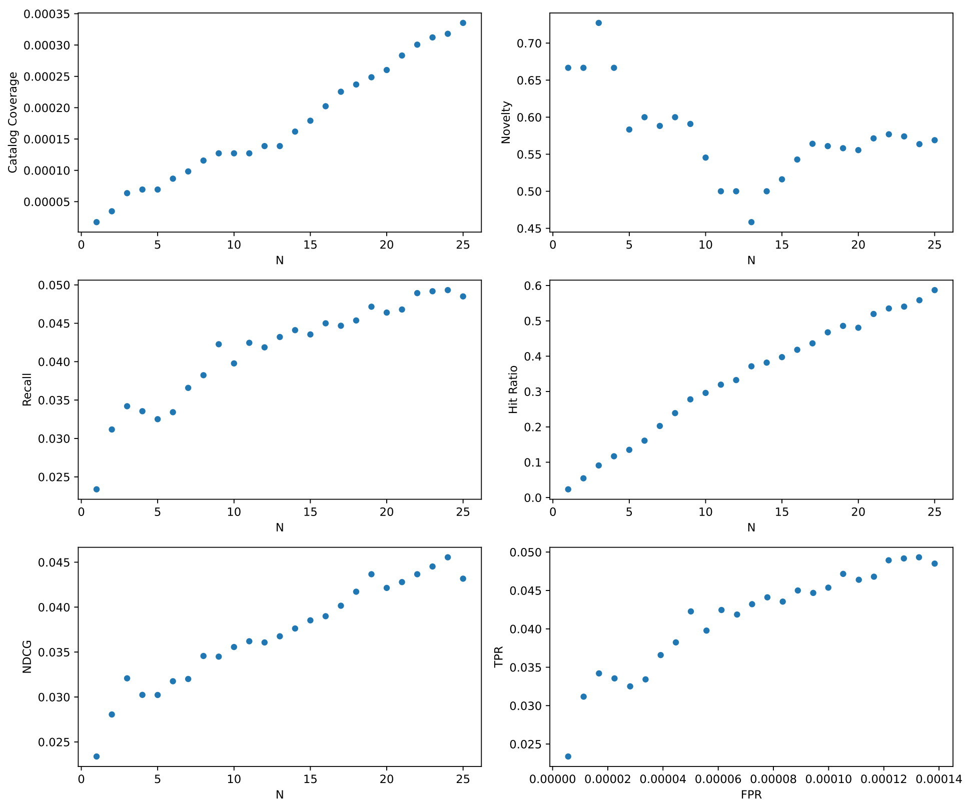


Figure 8.10: The evaluation results of the sesion-based model

were generated based on the last session in the training dataset were compared to the ones in the validation set.

The evaluation of the session-based model on the validation set is shown in figure 8.10. The session-based model has poorer performance compared to the CF model. It has about 39%, 30.6%, 41.81%, 38, 5% less performance of recall, hit-ratio, novelty, and NDCG accordingly compared to the CF model.

### 8.0.10 Model comparison

In this chapter, five models were implemented and evaluated: two baseline models, a content-based model, a collaborative filtering model based on matrix factorization, and a session-based model. The evaluation results of the models are shown in table 8.5. The results shown in the table were calculated in the following way: firstly values of different metrics for $k \in \{1, 2, ..., 25\}$ were calculated, where $k$ is the number of recommended items, and then for each metric an average of the values across different $k$ was taken, which then was filled to the table 8.5. Additionally, the AUC value was calculated for the different number of recommended items.

| | Baseline 1 | Baseline 2 | CB | CF | SB |
|---|---|---|---|---|---|
| **Cat. Cover.** | 0.00008 | 0.00847 | 0.01184 | 0.00542 | 0.00017 |
| **Novelty** | 0 | 0.99059 | 0.99397 | 0.98682 | 0.57385 |
| **Recall** | 0.03205 | 0.15099 | 0.04893 | 0.06753 | 0.04126 |
| **Hit Ratio** | 0.29776 | 0.66068 | 0.29049 | 0.48581 | 0.33714 |
| **NDCG** | 0.03182 | 0.09574 | 0.03544 | 0.06035 | 0.03695 |
| **AUC** | 4.3e-6 | 1.8e-5 | 6.2e-6 | 9.1e-6 | 5.5e-6 |

Table 8.5: The evaluation results of implemented models

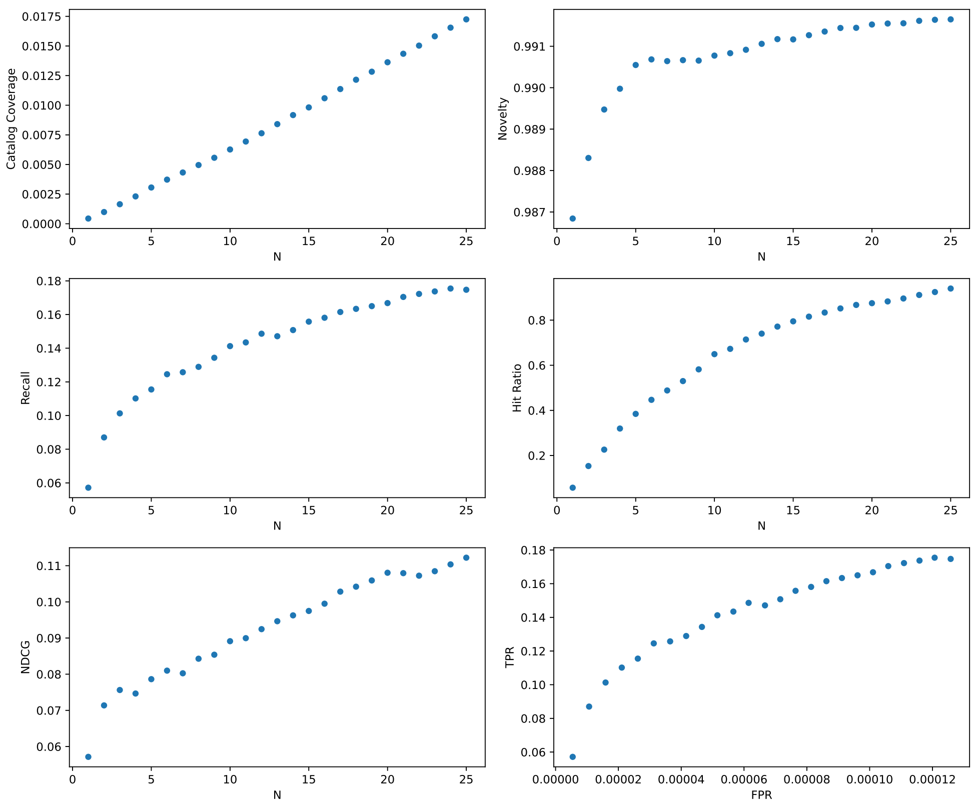


Figure 8.11: The evaluation results of the second baseline model on the test set

The highest scores of recall, NDCG, hit ratio, and AUC evaluated on the validation set has the second baseline model, which gives the recommendations for each user based on the user's *top-k* most interacted items. The model also has a high score in catalog coverage and novelty. Other models are far behind, especially in terms of recall. The model showed approximately similar performance on the test set, which is shown in figure 8.11.

Chapter 9

# Conclusion

The work aimed to develop a recommendation tool for the Data Dictionary application. To achieve that goal, firstly the author of the thesis got familiar with the Data Dictionary application, its use cases and defined the goals of a recommendation system for the application. Then author researched traditional recommendation approaches, such as content-based and collaborative filtering, and also modern session-based approaches. Afterwards, the author defined evaluation metrics of a recommendation system for the data dictionary application. The following metrics were chosen: *recall@k*, *NDCG@k*, *hit ratio @k*, *novelty@k*, and *catalog coverage @k*, where $k \in \{1, 2, ..., 25\}$. In particular, models were compared using an average value of the metrics since the application will use a different number of recommended items in the range of one up to twenty-five.

Originally the specification for the recommender system was that it should recommend items based on item similarities. To fulfill that requirement, the author of the thesis implemented the content-based (CB) model. In addition, the author implemented two baseline models: one of which (B1) suggests the most popular items and another (B2) returns items that the user interacted with the most, the collaborative filtering (CF) model based on matrix factorization, and the session-based (SB) model based on Transformers.

The offline evaluation of these models showed that the content-based model might not be the best choice for the dataset (and domain). Since the users of the data dictionary application do not expect the model to always give new suggestions, the baseline model that gives recommendations based on users' most visited and searched items might be the right choice. Despite the simplicity of the approach, the model showed the strongest performance compared to other implemented models in offline testing. In particular, the model had the highest recall, NCDG, and hit ratio and one of the highest values of catalog coverage and novelty. The collaborative filtering-based model had the second best value of the recall, NDCG, and hit-ratio. The CB and SB approaches performed a bit weaker on the validation set.

Every model has its own advantages and disadvantages. The B2 model in the offline testing showed the strongest ranking and classification performance. It is computationally inexpensive and straightforward in terms of implementation. However, the model may be the slowest in adjusting to changes in users' interests compared to other models. If there is no need to quickly adjust to changes in users' interests, the author's suggestion is to use the baseline model. Otherwise, the other models, such as the CF, CB, or SB, should be used.

Whichever model from the proposed ones is selected, the author of the thesis believes that the model will improve the application and make locating relevant data simpler.

# Bibliorgaphy

1. IYENGAR, Sheena S.; LEPPER, Mark R. When Choice is Demotivating: Can One Desire Too Much of a Good Thing? *Journal of personality and social psychology*. 2001, vol. 79, pp. 995–1006. Available from DOI: 10.1037/0022-3514.79.6.995.
2. IBM. *IBM Dictionary of Computing*. 1st ed. McGraw-Hill, Inc.Professional Book Group 11 West 19th Street New York, NYUnited States, [n.d.]. ISBN 978-0-07-031488-7.
3. *Lucidchart*. What is a database schema [online] [visited on 2022-03-10]. Available from: https://www.lucidchart.com/pages/database-diagram/database-schema.
4. DHARMENDRA SINGH RAJPUT, Ramjeevan Singh Thakur; BASHA, S. Muzamil. *Sentiment Analysis and Knowledge Discovery in Contemporary Business*. 1st ed. IGI Global, [n.d.]. ISBN 9781522549994.
5. *Elasticsearch*. Free and Open Search: The Creators of Elastic, ELK url Kibana — Elastic [online] [visited on 2022-03-10]. Available from: https://www.search.elastic.co/.
6. *TensorFlow Core*. word2vec — TensorFlow Core [online] [visited on 2022-03-10]. Available from: https://www.tensorflow.org/tutorials/text/word2vec.
7. RONG, Xin. *word2vec Parameter Learning Explained*. arXiv, 2014. Available from DOI: 10.48550/ARXIV.1411.2738.
8. RONG, Xin. *arXiv*. word2vec Parameter Learning Explained [online] [visited on 2022-03-22]. Available from: https://analyticsindiamag.com/guide-to-word2vec-using-skip-gram-model.
9. *Analyticsindiamag.com*. Guide To Word2vec Using Skip Gram Model [online] [visited on 2022-03-22]. Available from: https://analyticsindiamag.com/guide-to-word2vec-using-skip-gram-model.

10. MIKOLOV, Tomas; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. *Efficient Estimation of Word Representations in Vector Space*. arXiv, 2013. Available from DOI: `10.48550/ARXIV.1301.3781`.

11. *Wikipedia*. BERT (language model) [online] [visited on 2022-03-13]. Available from: `https://en.wikipedia.org/wiki/BERT_(language_model)`.

12. *MIT Press*. A Primer in BERTology: What We Know About How BERT Workf — Transactions of the Association for Computational Linguistics [online] [visited on 2022-03-13]. Available from: `https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00349/96482/A-Primer-in-BERTology-What-We-Know-About-How-BERT`.

13. KIM, Taeuk; CHOI, Jihun; EDMISTON, Daniel; LEE, Sang-goo. *ARE PRE-TRAINED LANGUAGE MODELS AWARE OF PHRASES? SIMPLE BUT STRONG BASELINES FOR GRAMMAR INDUCTION* [online] [visited on 2022-03-13]. Available from: `https://arxiv.org/pdf/2002.00737`.

14. ETTINGER, Allyson. *MIT*. What BERT Is Not: Lessons from a New Suite of Psycholinguistic Diagnostics for Language Models [online] [visited on 2022-03-13]. Available from: `https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00298/1923116/tacl_a_00298.pdf`.

15. VASWAN, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob. *Conference paper at 31st Conference on Neural Information Processing Systems (NIPS 2017)*. Attention Is All You Need [online] [visited on 2022-03-20]. Available from: `https://arxiv.org/pdf/1412.6980.pdf`.

16. TERVEEN, Loren; HILL, Will. *MIT*. Beyond Recommender Systems: Helping People Help Each Other [online] [visited on 2022-03-13]. Available from: `http://files.grouplens.org/papers/rec-sys-overview.pdf`.

17. WANG, Shoujin; HU, Liang; WANG, Yan; SHENG1, Quan Z.; ORGUN1, Mehmet; CAO, Longbing. *University of Shanghai for Science and Technology*. Modeling Multi-Purpose Sessions for Next-Item Recommendations via Mixture-Channel Purpose Routing Networks [online] [visited on 2022-03-19]. Available from: `https://www.ijcai.org/proceedings/2019/0523.pdf`.

18. LUDEWIG, Malte; JANNACH, Dietmar. *TU Dortmund and AAU Klagenfurt*. Evaluation of Session-based Recommendation Algorithms [online] [visited on 2022-03-19]. Available from: `https://arxiv.org/pdf/1803.09587.pdf`.

19. HIDASI, Balázs; BALTRUNAS, Linas; KARATZOGLOU, Alexandros; TIKK, Domonkos. *Conference paper at ICLR 2016*. Evaluation of Session-based Recommendation Algorithms [online] [visited on 2022-03-20]. Available from: `https://arxiv.org/pdf/1803.09587.pdf`.

20. KINGMA, Diederik P.; BA, Jimmy Lei. *Conference paper at ICLR 2015*. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION [online] [visited on 2022-03-20]. Available from: `https://arxiv.org/pdf/1412.6980.pdf`.

21. SOUZA PEREIRA MOREIRA, Gabriel de; RABHI, Sara; LEE, Jeong Min; AK, Ronay; OLDRIDGE, Even. *Conference paper at RecSys 2021*. Transformers4Rec: Bridging the Gap between NLP and Sequential / Session-Based Recommendation [online] [visited on 2022-03-20]. Available from: `https://research.facebook.com/file/1068762590546101/Transformers4Rec-Bridging-the-Gap-between-NLP-and-Sequential-Session-Based-Recommendation-1.pdf`.

22. *GitHub*. NVIDIA-Merlin/NVTabular [online] [visited on 2022-03-21]. Available from: `https://github.com/NVIDIA-Merlin/NVTabular`.

23. *HuggingFace*. HuggingFace Transformers [online] [visited on 2022-03-22]. Available from: `https://huggingface.co/docs/transformers/index`.

24. FEDER, Amir; OVED, Nadav; SHALIT, Uri; REICHART, Roi. CausaLM: Causal Model Explanation Through Counterfactual Language Models. *Computational Linguistics*. 2021, pp. 1–54. Available from DOI: `10.1162/coli_a_00404`.

25. *Kaggle*. eCommerce behavior data from multi category store [online] [visited on 2022-03-22]. Available from: `https://www.kaggle.com/datasets/mkechinov/ecommerce-behavior-data-from-multi-category-store`.

26. *Kaggle*. RecSys Challenge 2015 [online] [visited on 2022-03-22]. Available from: `https://www.kaggle.com/datasets/chadgostopp/recsys-challenge-2015`.

27. *SmartMedia*. The Adressa dataset for news recommendation [online] [visited on 2022-03-22]. Available from: `https://reclab.idi.ntnu.no/dataset`.

28. *Kaggle*. News Portal User Interactions by Globo.com [online] [visited on 2022-03-22]. Available from: `https://www.kaggle.com/datasets/gspmoreira/news-portal-user-interactions-by-globocom`.

29. LUDEWIG, Malte; JANNACH, Dietmar. Evaluation of session-based recommendation algorithms. *User Modeling and User-Adapted Interaction*. 2018, vol. 28, no. 4-5, pp. 331–390. Available from DOI: `10.1007/s11257-018-9209-6`.

30. GARG, Diksha; GUPTA, Priyanka; MALHOTRA, Pankaj; VIG, Lovekesh; SHROFF, Gautam. Sequence and Time Aware Neighborhood for Session-Based Recommendations: STAN. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Paris, France: Association for Computing Machinery, 2019, pp. 1069–1072. SIGIR'19. ISBN 9781450361729. Available from DOI: `10.1145/3331184.3331322`.

31. LUDEWIG, Malte; MAURO, Noemi; LATIFI, Sara; JANNACH, Dietmar. Empirical analysis of session-based recommendation algorithms. *User Modeling and User-Adapted Interaction*. 2020, vol. 31, no. 1, pp. 149–181. Available from DOI: `10.1007/s11257-020-09277-1`.

32. DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv, 2018. Available from DOI: `10.48550/ARXIV.1810.04805`.

33. CLARK, Kevin; LUONG, Minh-Thang; LE, Quoc V.; MANNING, Christopher D. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. arXiv, 2020. Available from DOI: `10.48550/ARXIV.2003.10555`.

34. YANG, Zhilin; DAI, Zihang; YANG, Yiming; CARBONELL, Jaime; SALAKHUTDINOV, Ruslan; LE, Quoc V. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. arXiv, 2019. Available from DOI: `10.48550/ARXIV.1906.08237`.

35. CHAI, T.; DRAXLER, R. R. Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature. In: 2014. Available from DOI: `10.5194/gmd-7-1247-2014`.

36. ORIGINLAB. *OriginLab Corporation*. ROC curve [online] [visited on 2022-03-27]. Available from: `https://www.originlab.com/doc/Tutorials/ROC-Curve`.

37. BAEZA-YATES, Ricardo; RIBEIRO-NETO, Berthier. *Modern Information Retrieval*. 1999. Available also from: `https://www.pearson.com/uk/educators/higher-education-educators/program/Baeza-Yates-Modern-Information-Retrieval/PGM407074.html`.

38. ZANKER, Markus; BRICMAN, Marcel; GORDEA, Sergiu; JANNACH, Dietmar; JESSENITSCHNIG, Markus. *Proceedings of 7th International Conference on Electronic Commerce and Web Technologies*. Persuasive online-selling in quality  taste domains [online] [visited on 2022-03-22]. Available from: `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.153.7856&rep=rep1&type=pdf`.

39. LEE, Dokyun; HOSANAGAR, Kartik. How Do Recommender Systems Affect Sales Diversity? A Cross-Category Investigation via Randomized Field Experiment. *Inf. Syst. Res.* 2019, vol. 30, pp. 239–259.

40. JANNACH, Dietmar; HEGELICH, Kolja. A Case Study on the Effectiveness of Recommendations in the Mobile Internet. In: *Proceedings of the Third ACM Conference on Recommender Systems.* New York, New York, USA: Association for Computing Machinery, 2009, pp. 205–208. RecSys '09. ISBN 9781605584355. Available from DOI: 10.1145/1639714.1639749.

41. GARCIN, Florent; FALTINGS, Boi; DONATSCH, Olivier; ALAZZAWI, Ayar; BRUTTIN, Christophe; HUBER, Amr. Offline and Online Evaluation of News Recommender Systems at Swissinfo.Ch. In: *Proceedings of the 8th ACM Conference on Recommender Systems.* Foster City, Silicon Valley, California, USA: Association for Computing Machinery, 2014, pp. 169–176. RecSys '14. ISBN 9781450326681. Available also from: http://florent.garcin.ch/pubs/garcin_recsys14a.pdf.

42. DOMINGUES, Marcos A.; GOUYON, Fabien; JORGE, Alípio Mário; LEAL, José Paulo; VINAGRE, João; LEMOS, Luís. Combining Usage and Content in an Online Music Recommendation System for Music in the Long-Tail. In: 2012, vol. 2. Available from DOI: 10.1007/s13735-012-0025-1.

43. TEAM, The pandas development. *pandas-dev/pandas: Pandas.* Zenodo, 2020. Latest. Available from DOI: 10.5281/zenodo.3509134.

44. TEAM, The Python development. *collections - Container datatypes - Python 3.10.4 documentation* [online]. [N.d.]. Latest [visited on 2022-03-27]. Available from: https://docs.python.org/3/library/collections.html.

45. REIMERS, Nils; GUREVYCH, Iryna. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing.* Association for Computational Linguistics, 2019. Available also from: http://arxiv.org/abs/1908.10084.

46. TEAM, The CuPy development. *CuPy: NumPy SciPy for GPU* [online]. [N.d.]. Latest [visited on 2022-03-27]. Available from: https://cupy.dev/.

47. TEAM, The NumPy development. *NumPy: The fundamental package for scientific computing with Python* [online]. [N.d.]. Latest [visited on 2022-03-27]. Available from: https://numpy.org/.

48. MARTÍN ABADI; ASHISH AGARWAL; PAUL BARHAM; EUGENE BREVDO; ZHIFENG CHEN; CRAIG CITRO; GREG S. CORRADO; ANDY DAVIS; JEFFREY DEAN; MATTHIEU DEVIN; SANJAY GHEMAWAT; IAN GOODFELLOW; ANDREW HARP; GEOFFREY IRVING; MICHAEL ISARD; JIA, Yangqing; RAFAL JOZEFOWICZ; LUKASZ KAISER; MANJUNATH KUDLUR; JOSH LEVENBERG; DANDELION MANÉ; RAJAT MONGA; SHERRY MOORE; DEREK MURRAY; CHRIS OLAH; MIKE SCHUSTER; JONATHON SHLENS; BENOIT STEINER; ILYA SUTSKEVER; KUNAL TALWAR; PAUL TUCKER; VINCENT VANHOUCKE; VIJAY VASUDEVAN; FERNANDA VIÉGAS; ORIOL VINYALS; PETE WARDEN; MARTIN WATTENBERG; MARTIN WICKE; YUAN YU; XIAOQIANG ZHENG. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* 2015. Available also from: `https://www.tensorflow.org/`. Software available from tensorflow.org.
49. RADFORD, Alec; WU, Jeff; CHILD, Rewon; LUAN, David; AMODEI, Dario; SUTSKEVER, Ilya. Language Models are Unsupervised Multitask Learners. In: 2019.

Appendix A

# Acronyms

| | |
|---|---|
| **AP** | Average Precision |
| **API** | Application Programming Interface |
| **AUC** | Area Under Curve |
| **BERT** | Bidirectional Encoder Representations from Transformers |
| **BI** | Business Intelligence |
| **CB** | Content-Based (recommendation approach) |
| **CBRS** | Content-Based Recommendation System |
| **CF** | Collaborative Filtering |
| **CTR** | Click-Through Rate |
| **DCG** | Discounted Cumulative Gain |
| **FN** | False Negatives |
| **FP** | False Positives |
| **FPR** | False Positive Rate |
| **GPU** | Graphics Processing Unit |
| **GRU** | Gated Recurrent Unit |
| **HR** | Hit Ratio |
| **HTML** | HyperText Markup Language |
| **IDCG** | Ideal Discounted Cumulative Gain |
| **IT** | Information Technology |

| | |
|---|---|
| **JSON** | JavaScript Object Notation |
| **LSTM** | Long Short-Term Memory |
| **MAE** | Mean Absolute Error |
| **MRR** | Mean Reciprocal Rank |
| **MSE** | Mean Squared Error |
| **NDCG** | Normalized Discounted Cumulative Gain |
| **NLP** | Natural Language Processing |
| **RMSE** | Root Mean Squared Error |
| **RNN** | Recurrent Neural Networks |
| **ROC** | Receiver Operating Characteristic |
| **RR** | Reciprocal Rank |
| **SB** | Session-Based (recommendation approach) |
| **SBRS** | Session-based recommendation algorithms |
| **SBRS** | K-Nearest Neighbours |
| **SVD** | Singular Value Decomposition |
| **TF-IDF** | Term Frequency–Inverse Document Frequency |
| **TN** | True Negatives |
| **TP** | True Positives |
| **TPR** | True Positive Rate |

Appendix B

# Contents of enclosed Micro SD

```
readme.txt ......................... the file with Micro SD contents description
src ................................ the directory of source codes
  wbdcm ............................ implementation sources
  thesis ........................... the directory of LaTeX source codes of the thesis
text ............................... the thesis text directory
  thesis.pdf ....................... the thesis text in PDF format
```