Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Bachelor's diploma thesis

in the field of study Computer Science

Recommendation System for Books

Paweł Rzepiński

student record book number 276891

Ryszard Szymański

student record book number 276914

thesis supervisor

Agnieszka Jastrzębska, Ph.D. Eng.

WARSAW 2019

....

supervisor’s signature author’s signature

Abstract

Recommendation System for Books

The amount of information we encounter in the Digital Era is enormous. The past problem of not having enough information transformed into the issue of finding it in a vast ocean of data. Recommendation systems are tools to navigate users to valuable items through the abundance of choices. People often rely on opinions of others when they find themselves unable to make an informed decision. Suggestions presented to the user aim to help them in the decision-making process of what to buy, watch or where to go.

The objective of this thesis was to develop a recommendation system for books. The work consisted of three main stages: describing and analyzing the dataset, experimenting with different recommendation algorithms by comparing and evaluating their outcomes, implementing selected techniques in a web application to showcase the two most common use-cases of recommender engines.

Recommending books using user-assigned tags proved to be the most accurate solution to suggesting similar items for a specific book. Applied methods were able to predict ratings (on the 1-5 scale) of various users with an average of 0.6 point precision. The Singular Value Decomposition model delivered the best results in regard to the absolute values and generalization potential. The workflow devised during the whole process supported reproducibility of the results.

In the developed recommendation system the quality of data played a more significant role than its quantity, especially in the content-based methods. In the collaborative-filtering approach the optimization of parameters was an essential step preceding the comparison of final results. Additionally, the need for various evaluation metrics to fully assess models' performance was observed.

Keywords: recommendation system, reproducibility, natural language processing, collaborative filtering, content-based similarity

Streszczenie

System rekomendacji książek

Ilość informacji, które napotykamy w erze cyfrowej, jest ogromna. Dotychchczasowy problem braku odpowiednich informacji został zastąpiony przez kwestię znalezienia ich w ogromnym zasobach danych. Systemy rekomendacji są narzędziami prowadzącymi użytkowników do wartościowych przedmiotów w obliczu licznych możliwości wyboru. Ludzie często polegają na opiniach innych osób, gdy nie są w stanie podjąć pewnej decyzji. Sugestie przedstawione użytkownikowi pomagają w procesie decyzyjnym dotyczącego tego co kupić, obejrzeć lub gdzie się udać.

Celem tej pracy było opracowanie systemu rekomendacji książek. Praca składała się z trzech głównych etapów: opisywania i analizowania zbioru danych, eksperymentowania z różnymi algorytmami rekomendacji poprzez porównywanie i ocenianie ich wyników, użycie wybranych technik w aplikacji internetowej w celu zaprezentowania dwóch najczęstszych przypadków użycia metod rekomendacji.

Rekomendacje książek bazujące na przyporządkowanych przez użytkowników tagach okazały się najdokładniejszym rozwiązaniem sugerującym podobne pozycje dla konkretnej książki. Zastosowane metody były w stanie przewidzieć oceny różnych użytkowników podawane w skali 1-5 ze średnią dokładnością wynoszącą 0,6. Model bazujący na rozkładzie według wartości osobliwej (Singular Value Decomposition) osiągnął najlepsze wyniki biorąc pod uwagę wszystkie miary. Opracowany system pracy wspierał odtwarzalność uzyskanych wyników.

W opracowanym systemie rekomendacji jakość danych okazała się znacznie ważniejsza niż jej ilość, szczególnie w metodach opartych na treści. W podejściu korzystającym z filtrowania społecznościowego (collaborative filtering) optymalizacja parametrów była niezbędna do późniejszego porównania końcowych wyników. Dodatkowo, wyraźnie widoczna była potrzeba stosowania różnych metryk do pełnej oceny wydajności modeli.

Słowa kluczowe: system rekomendacji, odtwarzalność, przetwarzanie języka naturalnego, filtrowanie społecznościowe, podobieństwo bazujące na zawartości

Warsaw, ...

Declaration

I hereby certify that I wrote my part of the Engineering thesis (according to the division of work described in Appendix 1 of the thesis) entitled „Recommendation System for Books” on my own, under the guidance of the thesis supervisor Agnieszka Jastrzębska, Ph.D. Eng.

....

Contents

1 Introduction

Recommendation systems are information filtering tools which navigate users to valuable items through the abundance of available data. People often rely on opinions of others when they find themselves unable to make informed decision. Suggestions presented to the user aim to help them in the decision-making process of what to buy, watch or where to go.

The Internet companies e.g. operating in e-commerce or providing media services rely on recommendation engines to increase revenue by developing a better understanding of the users' needs, maintaining their engagement and promoting new content. Moreover, the recommendation mechanisms can be responsible for the performance of their core offerings. Chief Product Officer in YouTube has revealed that more than 70 percent of viewing hours on the platform are driven by recommendations [30].

The relevance of the suggestions is mostly evaluated against a selected metric e.g. order value (Amazon) or watch time (Youtube, Netflix). Business domain of the recommendation system partially imposes selected techniques and properties of suggested items taken into regard. For example, the scale of the Youtube service requires a specific system architecture able to provide and update millions of recommendations under strict time constraints [6].

In case of Goodreads[1] – the social platform for books fans – the information filtering problem consists of choosing what to read next. Books are text-based items making it possible to analyze them using natural language processing. Moreover, since reading a book requires considerable time investment appropriate quality of the recommendations is of great importance.

Recommendation methods vary depending on different scenarios: cold-start – there is no prior data about the user and hot-start – the user's actions history can be taken into account. In the former case, it is possible to utilize information about properties of the books to suggest an item which is most similar to currently viewed by the user. In the latter case, the past interactions of the user and their choices are taken into account when providing suggestions specifically tailored to them. Moreover, the whole community can be used to enhance the recommendations for one person through the process of finding similar users and deriving information based on their experiences.

[1]`https://www.goodreads.com/`

Objective

The main objective of this thesis is to develop a recommendation system for books. The work will consist of three main stages: describing and analyzing the dataset, experimenting with different recommendation algorithms by comparing and evaluating their outcomes, implementing selected techniques in a web application to showcase the two most common use-cases of recommender engines. The workflow developed during the whole process will put emphasis on reproducibility of the results.

Recommendation methods implemented in the project will tackle both cold-start and hot-start situations mentioned in the previous paragraphs. Results of these techniques will be compared using different performance metrics, visualized and validated against ground-truth data for each of the approaches.

The web application will allow the user to explore the dataset and select specific users or books for which recommendations will be presented: "Similar books to X" – filled with items similar to the selected book and "You may also like X, Y, Z" – with recommendations based on books rated by the selected user.

2 Methods

In this chapter, we present methods used in both approaches to the recommendation problem – the content-based solutions leveraging item characteristics and collaborative-filtering techniques deriving information from actions of the whole community of users. We describe different models and their properties. For each approach evaluation metrics are defined. The inevitable reproducibility problem is introduced and discussed in the last section.

2.1 Content-based approach

Content-based methods determine recommendations based on the similarity of the items of interest. Those models take into account the items which the user liked in the past and search for content that is similar to them. In order to compare different items, a specific representation of a single item needs to be defined, which captures the distinctive features of the items. Then, by using a chosen similarity measure, matching items are identified. In consequence, the development of a content-based recommendation system is mainly composed of tasks regarding data cleaning and features engineering. The crucial part is to determine, based on expert knowledge, which features describe the item accurately.

2.1.1 Representation of the items of interest

Items of interest are described using a feature vector composed of specifically selected features. According to [28], most content-based recommendation systems make use of text features extracted from Web pages, emails, news articles or product descriptions. Natural language is a very complex phenomenon composed of infinite ambiguities and exceptions, which makes information retrieval from a text a challenging task that requires sophisticated techniques.

Text preprocessing

Raw text in its initial form has several flaws that make it not appropriate for feature extraction. The presence of stop words, which are very frequently appearing words such as *a* and *the* provides no useful information. Moreover, as words are conjugated, the same word can have several different forms, like for example *work* and *working*.

In order to address those issues, several operations can be used. Stop words can be filtered out by preparing a corpus composed of predefined stop words from a given language. Text normalization can be performed using stemming and lemmatization. Both of those operations have the goal of reducing inflectional forms to a common base form [29], but they achieve it in a different manner. Stemming is the process of reducing a given word to its stem. An example of this operation is the removal of the *ing* ending from the word *working* and as a result the word *work* is obtained. Lemmatization is a bit more complex as it tries to determine whether two words have the same canonical form. In case of irregular conjugation like for example *good, better, the best* lemmatization would map them to *good*. This is the case in which stemming fails to perform as it only looks at the word endings.

One of the most popularly used stemmers nowadays is the Porter Stemmer described in [23]. Basically it follows a defined sequence of rules, where a result of a single rule is passed to another rule repeatedly. Here example rules followed by the Porter Stemmer are presented:

$$IZATION \longrightarrow IZE \qquad \text{e.g. (normalization} \longrightarrow \text{normalize)} \tag{2.1}$$

$$FULLNESS \longrightarrow FUL \qquad \text{e.g. (usefulness} \longrightarrow \text{useful)} \tag{2.2}$$

$$IES \longrightarrow I \qquad \text{e.g. (duties} \longrightarrow \text{duti)} \tag{2.3}$$

Twenty years from the release of the Porter algorithm, the English Stemmer (often referred to as the Porter2 algorithm) was devised. It introduced improvements such as regarding *y* terminations being changed less often to *i*, *us* suffix not losing its *s* [22].

Wordnet is a large lexical database of English offering one of the earliest and most commonly used lemmatizers [8]. It performs lemmatization by using two types of processes. Based on a list of inflectional ending of a specific syntactic category, it attempts to detach inflectional endings from the individual words. Another list contains exceptions for each syntactic category in which a search for the inflected form is done [8].

Vector space model

One of the most frequently used representations for text documents is the vector space model. A single document d is represented by an m-dimensional vector in which each dimension corresponds to a given term. To each dimension, a weight is assigned which signifies how important is the given feature. One way to determine the vector representations of each document is by constructing a term-document matrix, each row corresponds to a single and each columns to a given document. Matrix cells are filled with the weights assigned to a particular word in a given document.

Table 2.1: Term-document matrix with word count weighting.

(a) Example documents for weighting comparison.

d_1 = John is happy.
d_2 = John is very happy.
d_3 = John is very very happy.

(b) Term-document matrix with word count weighting.

	d_1	d_2	d_3
happy	1	1	1
is	1	1	1
John	1	1	1
very	0	1	2

(c) Term-document matrix with TF-IDF weighting.

	d_1	d_2	d_3
happy	0	0	0
is	0	0	0
John	0	0	0
very	0	0.18	0.35

There exists several methods of assigning weights. One of the simplest approaches is using word counts as weights, which means that if the term t_i has appeared in document d_j n times, then the weight $w_{i,j}$ is equal to n. An example of a term-document matrix using the word count scheme is demonstrated in Table 2.1b.

Each column of the term-document matrix is a feature vector representing the document that corresponds to that column. The word count weighting scheme is a simple way of calculating term weights when representing documents. However, it turns out that it is not the best measure of association between words [14]. All terms are equally important which causes problems with words like *a*, *the*, *good* which appear in high amounts in all kinds of topics.

The TF-IDF approach addresses the issues that are present when using word count weights by making use of scaling features. Two coefficient are defined: TF which signifies how often a given term appears in a given document and IDF which is responsible for determining of how important the given term is in the context of the entire corpus. In academic literature there those coefficients are defined in various ways. In [29] $tf_{t,d}$ indicates the amount times term t has appeared in document d, whereas the idf_t coefficient is defined as:

$$idf_t = \log \frac{N}{df_t} \tag{2.4}$$

Where N is the total number of documents and df_t is the number of documents that contain the term t. Using the TF and IDF coefficients, the TF-IDF is defined as:

$$tfidf_{t,d} = tf_{t,d} \times idf_t \tag{2.5}$$

The TF-IDF weights of terms for the documents from Table 2.1a are presented in Table 2.1c.

N-grams

Analyzing natural language word by word is not an effective approach, as context has a major influence on the meaning of a given sentence. We might often encounter a situation in which a specific sequence of words created an entirely new meaning than all of elements analyzed separately. Moreover, a single term exists only as a sequence, as for example *San Fransisco*, the words *San* and *Francisco* rarely appear by themselves.

An n-gram is a sequence of n consecutive words. For n equal to 1 the sequence is referred to as an unigram, in case of 2 as a bigram and a trigram for 3. For example the trigrams for the sentence *Susan goes to school* are: *Susan goes to* and *goes to school.* N-grams improve context-awareness and allow to capture additional keywords and meanings, which is especially useful in the case of books, as often such elements like events or places of action are composed of more than one word. When using n-gram features, those sequences are treated as single terms and their weights are calculated the same way as if a single word was computed.

2.1.2 Item similarity

The vector representation of documents has been described in Section 2.1.1, in order to retrieve items that are alike a similarity measure has to be defined. One approach might consider the length of the vectors when measuring similarity. However, this approach suffers from the fact that two documents that are very similar might have a significant vector difference caused by their difference in length [29]. The cosine similarity addresses this issue by taking into account only the direction in which the vector is pointing. The cosine similarity between vectors $\vec{i}$ and $\vec{j}$ is defined in Equation (2.6)

$$sim(i, j) = \frac{i \cdot j}{|i||j|} \tag{2.6}$$

2.2 Collaborative filtering approach

Collaborative filtering techniques deliver a set of recommendations to a specific user by leveraging ratings of the remaining users known by the system. Given user u and item i we can predict the rating given by this user to a new item j by finding users with a similar history of rated items. Also, items i and j will be rated similarly by user u if they are given similar ratings by other users. This approach allows to discover similarities between items or users by ratings (given or obtained) and not by their inner characteristics contrary to content-based techniques.

Table 2.2: Example set of ratings for two users and three items.

	Item 1	Item 2	Item 3
User A	3	5	4
User B	1	2	?

2.2.1 SlopeOne

The SlopeOne approach is based on recommending items to users using the average difference in their preferences of items [17]. Thanks to its' simplicity large part of calculation can be precalculated, thus making predictions really fast. Moreover, results achieved by even simplest variation of the SlopeOne algorithm presented below can achieve results comparable to other approaches.

Given a set of relevant items $R_i(u)$ for user u and item i, i.e. items j which have been also rated by user u and have at least one common user with item i, the estimated rating for this user and the item can be calculated as:

$$\hat{r}_{ui} = \mu_u + \frac{1}{|R_i(u)|} \sum_{j \in R_i(u)} \text{dev}(i, j), \tag{2.7}$$

where μ_u is user's average rating and $\text{dev}(i, j)$ is mean difference between ratings of items i and j:

$$\text{dev}(i, j) = \frac{1}{|U_{ij}|} \sum_{u \in U_{ij}} r_{ui} - r_{uj} \tag{2.8}$$

where U_{ij} is a set of users who rated both item i and j.

Consider the situation described in Table 2.2. In order to predict a rating for user B and item 3 we have to calculate the average difference between relevant items for the remaining users i.e. user A:

$$\text{dev}(3, 1) = 4 - 3 = 1 \qquad \text{dev}(3, 2) = 4 - 5 = -1 \tag{2.9}$$

Then, the final estimate is given by taking into account other ratings of user B:

$$\hat{r}_{B3} = \frac{(1 + 1) + (2 + -1)}{2} = 1.5 \tag{2.10}$$

2.2.2 Neighbor-based methods

Neighbor-based models aggregate ratings of the most similar elements (users or items), i.e. nearest neighbors, when calculating an ratings estimation for a particular element. Such algorithms can utilize item or user similarities which are defined by a predefined metric.

In user-based approaches rating $\hat{r}_{ui}$ for user u and item i is calculated based on the ratings given to item i by users most similar to user u:

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in N_i^k(u)} \text{sim}(u,v) \cdot r_{vi}}{\sum\limits_{v \in N_i^k(u)} \text{sim}(u,v)} \tag{2.11}$$

In the item-based variant ratings of the most similar items are taken into account:

$$\hat{r}_{ui} = \frac{\sum\limits_{j \in N_u^k(i)} \text{sim}(i,j) \cdot r_{uj}}{\sum\limits_{j \in N_u^k(i)} \text{sim}(i,j)} \tag{2.12}$$

When deciding between those two approaches, multiple aspects have to be considered: (i) accuracy, (ii) efficiency, (iii) stability, (iv) justifiability, (v) serendipity as further described in [28]. The final decision has to be made based on the characteristics of the dataset, e.g. distribution of the ratings and average number of neighbors.

The collaborative filtering problem is a generalization of the data classification problem, thus the research and methods from the latter can be used for the former [1]. A classic example of such transition is the k-Nearest-Neighbor algorithm, which can be adapted to serve as a recommendation technique.

Similarity metric

When calculating similarity only common elements (users or items) are considered. One of the classic similarity metrics is the Cosine Vector Similarity which can be defined for the item-based variant as:

$$\text{CVS}(i,j) = \frac{\sum\limits_{u \in U_{ij}} r_{ui} \cdot r_{uj}}{\sqrt{\sum\limits_{u \in U_{ij}} r_{ui}^2} \cdot \sqrt{\sum\limits_{u \in U_{ij}} r_{uj}^2}} \tag{2.13}$$

The definition in Equation (2.13) is a slight modification of definition from Equation (2.6).

A more advanced similarity measure is the Pearson correlation coefficient, which expresses the tendency for users to rate items i and j similarly:

$$\mathrm{PCC}(i,j) = \frac{\sum\limits_{u \in U_{ij}} (r_{ui} - \mu_i) \cdot (r_{uj} - \mu_j)}{\sqrt{\sum\limits_{u \in U_{ij}} (r_{ui} - \mu_i)^2} \cdot \sqrt{\sum\limits_{u \in U_{ij}} (r_{uj} - \mu_j)^2}}, \tag{2.14}$$

where μ_i and μ_j denote mean ratings for items i and j.

Baselines

People tend to present different behaviours during the rating process when a rating scale is introduced. Some of these rating patterns are described in [20]. Taking into account such trends can greatly improve recommendation accuracy [11]. A normalization factor can be introduced as a data preparation step and calculated based on the ratings distributions or it could be incorporated in the similarity metric, e.g. as in the Pearson-baseline correlation coefficient (introduced in Equation (2.14)) when baselines are used instead of means.

2.2.3 Model-based methods

The main idea of model-based methods is to embed user and item characteristics into a set of latent vectors which can be viewed as categories (e.g. "horror books" or "horror books fan"). Such methods tackle the problem from a more global perspective as opposed to neighbor-based models which only look for similarity locally. The mentioned advantages could have been observed during the Netflix Prize contest where model-based algorithms achieved the best results [3].

Matrix factorization

Given a $m \times n$ matrix of ratings R, where rows correspond to users and columns to items, the matrix factorization algorithm calculates an approximation of the input matrix:

$$R \approx UV^T, \tag{2.15}$$

where U is a $m \times k$ matrix, V is a $n \times k$ matrix and k determines the level of approximation (rank). Let p_u be the row of U with index u and q_i the column of V with index i. Then, p_u is the latent factor of user u and q_i is the latent factor of item i. Each value of this vector can be interpreted as an affinity of the element toward one of the k concepts. The estimated rating for user u and item i can be defined as:

$$\hat{r}_{ui} = p_u \cdot q_i \tag{2.16}$$

The process of finding such factors is guided by the minimization of the cumulative sum of the function of error in predictions. There are multiple techniques for such search including Stochastic Gradient Descent (SGD) and Alternating Least Squares (ALS).

The difference between various techniques based on matrix factorization is mainly revealed in the constraints imposed on U and V matrices, e.g. Singular Value Decomposition (SVD) method introduces orthogonality condition for these matrices.

2.3 Evaluation

Assessing performance of method requires appropriate metric. The choice can made depending on form of the data – binary or numeric, its' characteristics – origin, quality, volume and most importantly the goal of the evaluation.

2.3.1 Ratings estimation accuracy

The comparison of ratings prediction models is often conducted using an accuracy metric. Typically, the whole dataset is divided into two sets: training set R_{train} used in model training and test set R_{test} used for evaluation. The simplest example of such metric is Mean Absolute Error (MAE) defined as:

$$\text{MAE} = \frac{1}{|R_{test}|} \sum_{\hat{r}_{ui} \in R_{test}} |r_{ui} - \hat{r}_{ui}| \tag{2.17}$$

The MAE straightforwardly reflects the accuracy of the prediction and can be easily interpreted.

The RMSE used during the Netflix Prize contest [3] has a higher penalty for large errors than MAE due to being calculated using the formula:

$$\text{RMSE} = \sqrt{\frac{1}{|R_{test}|} \sum_{\hat{r}_{ui} \in R_{test}} (r_{ui} - \hat{r}_{ui})^2}. \tag{2.18}$$

In both metrics presented a lower score implies better predictions, because they are functions of error between true and estimated rating value. The choice largely depends on data characteristics i.e. the percentage and the extremity of outliners.

2.3.2 Precision and recall

Treating recommendations as an information retrieval problem allows us to use evaluation metrics such as precision and recall, although some modifications are necessary. We have to define the relevancy of the recommended item. Depending on the data used as the test set relevancy can be expressed in terms of satisfying conditions such as (i) presence or absence of the item in the test set (ii) the estimated rating above a specified threshold [28]. Then, the precision and recall for a particular item or user can be defined as:

$$\text{Precision} = \frac{|\text{Recommended items which are relevant}|}{|\text{Recommended items}|} \tag{2.19}$$

$$\text{Recall} = \frac{|\text{Recommended items which are relevant}|}{|\text{Relevant items}|} \tag{2.20}$$

From the definitions of those metrics the trade-off between each other is visible – often the lower the precision the higher the recall and vice verse.

2.4 Reproducibility

The reproducibility problem is the lack of ability to repeat an analysis and obtain the same results using the same source code and input data. As described in [2] many findings suffer in modern scientific fields suffer from the reproducibility problem. This leads to unverifiable results which can turn into making incorrect discoveries and conclusions. It also inhibits collaborative research by not allowing collaborators to thoroughly check the analysis by themselves. Fortunately, attention has been brought to this problem and more effort is put into emphasising reproducibility. For example the Nature journal publicized an announcement regarding that topic [19]. The reproducibility of research is crucial as it proves the level of correctness of the obtained results and enables others to use the created methods and conclusions as stated in [27].

A high level of reproducibility can be achieved by sharing code, data and the environment that was used to run the code [32]. The original data is needed in order to be able to provide the exact same input. The source code is necessary in order to allow others to rerun the whole analysis. However, it is important to ensure that the source code will always give the same result, which may not always be the case when using random number generators. It is essential that a specific seed is set when randomness appears in such operations as dividing the data into training and testing datasets or model training. Moreover, the environment in which the code was run has to be closely replicated as different version of packages and system requirements may alter the behaviour or even break the analysis process.

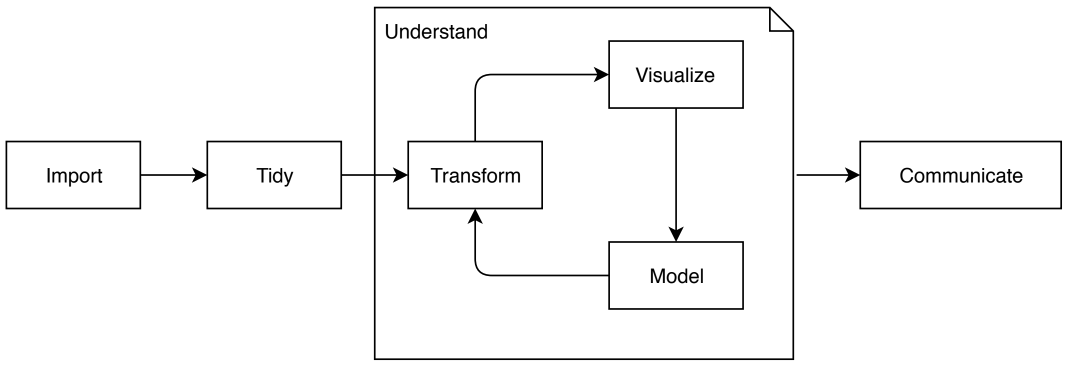


Figure 2.1: Diagram representing the process of data exploration. Based on [35].

Reproducibility issues can also be encountered during the development of the analysis. In [35] the data exploration process is described as the repetition of a loop composed of data transformation, visualization and modelling. It is illustrated by the graph presented in Figure 2.1.

A single change in a part of that process can invalidate future results and actions built on previously obtained outcomes. Taking into consideration the fact that model training and features extraction are highly time-consuming tasks only the necessary steps should be repeated. Keeping track of such dependencies manually is very prone to human error and inhibits development and reproducibility.

3 Data analysis and preparation

This chapter gives a thorough description of the data used in the recommendation system. The analysis emphasizes problems present in the available information and provides meaningful observations of the data characteristics which can be leveraged in the next chapters.

3.1 Data overview

The data used in this thesis comes from *goodbooks-10k* dataset which is available on the GitHub platform under the Creative Commons Licence [36]. It was collected from the Goodreads platform – a social cataloging website featuring a database of books along with annotations and reviews. The dataset consists of six million ratings for the ten thousands most popular books. The data is composed of both structured and semi-structured data in the form of csv and xml files. They contain books' metadata including but not limited to: book titles, descriptions, author names and assigned tags. Moreover, the data about users' interactions is also present in the form of book ratings based on a scale from 1 to 5. This set features more than 50 thousand unique users.

3.2 Book metadata

The books metadata is provided in xml files – one per book and includes, among others, sections mentioned below.

Multiple identification numbers

Duplicate book entries are present due to the existence of different editions of the same book such as translations and republications. Additionally, a variety of identification numbers are used in the data: (i) numbers specific to the dataset in the range 1 to 10000, (ii) book edition identification numbers from Goodreads service, (iii) identifiers of specific books understood as the abstract work also given by the Goodreads platform.

Descriptions

Descriptions providing a brief introduction to the plot of the book are available. As illustrated by the Figure 3.1 most books tend to have descriptions shorter than 2000 characters. On average the length text is 904 characters. The dataset contains books written in various languages and as a result there exists non-English descriptions. Most of the entries are in English, except for 199 of them. Moreover, 315 books in the dataset had no description.

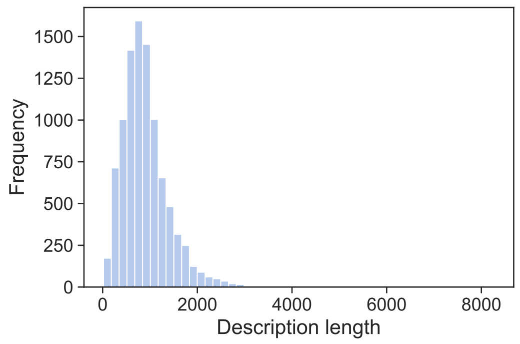


Figure 3.1: Distribution of description lengths.

Tags

Users can assign custom tags to the books on the Goodreads platform – often used as 'shelves', 'lists' or just as genre indicators. The dataset consists of 34252 unique tags. Among them one can identify preselected 'Goodreads genres' which are 1228 tags listed on one of the platform's subsites[1]. Since users are allowed to create custom tags they are defined in different languages and sometimes have no meaningful names.

One of the most valuable tags is the one named 'to read'. This tag gets assigned to a book when the user clicks on the 'want to read' button on the Goodreads website. The data about such actions is expressed in one of the csv files in the form of user and book pairs.

Similar books

The xml files include an element called 'similar_books' which lists books considered as similar to the current one as selected by Goodreads. Additionally, there is an element called 'series_work', which contains the id of the series to which the book belongs too, however no book data regarding that specific series is present. In order to check the correctness of the *similar_books* data, a manual validation was performed. Several popular book series such as *Harry Potter* or *Winnie the Pooh*

[1]https://www.goodreads.com/genres/list

were selected. The analysis showed that books belonging to the same series rarely appeared in the 'similar_books' tag, however the remaining positions did have a similar setting or belonged to a similar genre.

3.3 Ratings

The data about ratings given by users consists of user id, book id and rating value tuples. There are no duplicates nor books not included in the xml files.

The distribution of ratings has a profound impact on decisions made when selecting collaborative-filtering method. Recommendations for users with a low number of ratings can be highly inaccurate. In the dataset the mean rating count for all users is approximately 112 and every user has at least 19 ratings – the entire distribution can be seen in Figure 3.2. There are no clear disproportions there as the chart resembles normal distribution.

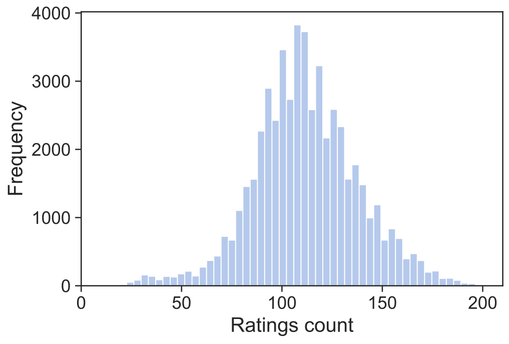


Figure 3.2: Distribution of users ratings counts.

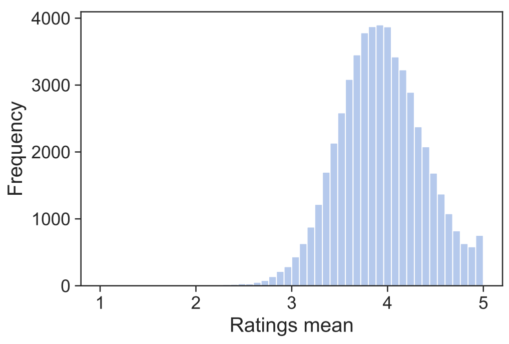


Figure 3.3: Distribution of users' mean rating.

Likewise, the mean ratings count for books in the dataset is approximately 598 and every book has at least 8 ratings. Rating counts for books are not evenly distributed due to the popularity effect – the most popular book in the dataset has 22,806 ratings when the least rated one 8. Such phenomena can be attributed to *Power Law* relationship[2] and is often encountered in online ratings services e.g. movie databases [26]. Figure 3.5 shows that the 100 most popular books account for 1 million ratings and the top 1000 for more than 3 millions. At the same time, the majority of the books was rated between 10 and 1000 times as illustrated in Figure 3.4.

As stated in Section 2.2.2 users mean rating are also important in the decision process. Users in the dataset, on average, have a mean rating of 3.93. The distribution presented in Figure 3.3 clearly shows a tendency for users to give ratings from the upper part of the scale more frequently. Moreover, extremes are also present – there are 266 users who only used the rating of 5.

[2]Also refereed to using *long tail* phrase to illustrate the values distribution.

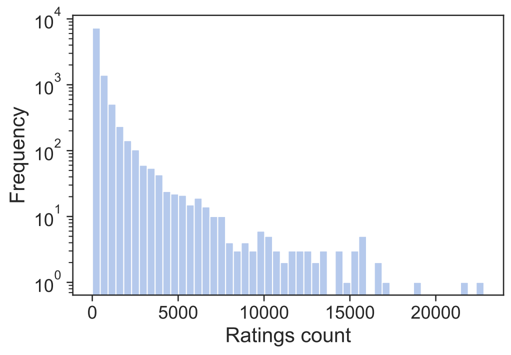


Figure 3.4: Distribution of books ratings counts.

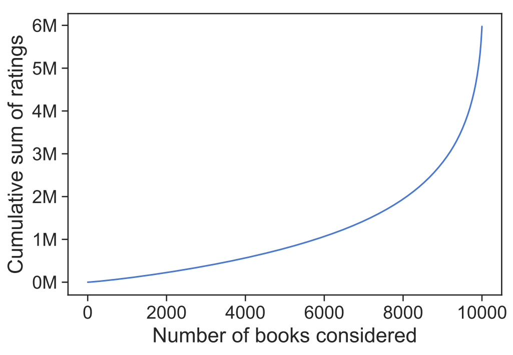


Figure 3.5: Cumulative sum of ratings count for books. Counts were presorted according to book popularity. Vertical axis numbers in millions.

3.4 Data preparation

Based on the data analysis presented before and the requirements of the selected methods several data processing steps were conducted.

Books metadata merging and cleaning

Books metadata was divided between different files. In order to ease the subsequent work, the data had to be merged into one source. First of all, data from xml files was extracted and appended to the main book csv file. Descriptions written in other languages than English were removed from the dataset. Then, all identification numbers were merged into one aggregating data for various editions of the book. Later, tags-related data was appended.

Due to the existence of tags with names defined in languages than English or without meaningful names as mentioned in Section 3.2, only Goodreads genres were considered. As a result the amount of tags was reduced from 34252 to 1228. However, this did not impact the validity of the data as each book still had, on average, 21 unique tags assigned after this process as depicted in Figure 3.6.

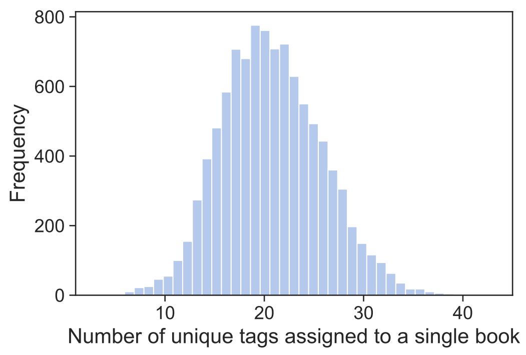


Figure 3.6: Amount of unique tags assigned to a single book.

Ratings dataset partitioning

Ratings dataset was split into training and test set using a 90:10 ratio using reshuffled data. Constant random seed was used to ensure the reproducibility of this operation. Implementations of the collaborative-filtering methods used in this thesis assume that all users and items are known to the system before evaluation. Thus, the produced training set was verified to satisfy that condition. Moreover, the dataset containing information about books marked as 'to read' by the users was filtered to exclude books present in the ratings training set. Situations when the user added a book to his 'to read' list and then rated negatively affects evaluation.

4 Implementation

Having defined the methods and the data used in the system we can proceed to the implementation process. First of all, a specifc workflow is introduced to tackle the problem of reproducibility. Then, we describe the problems encountered and decisions made in content-based and collaborative-filtering approaches. The obtained results are provided along with visualizations. At last, we present the idea behind the web application and use cases.

4.1 Workflow

The development of the recommendation system is a combination of programming work and data analysis. Traditional work methodologies used in software development are not sufficient in this situation. In order to facilitate the development and to address issues described in Section 2.4 a custom workflow has been devised.

The project pipeline can be divided into the following phases: data cleaning, feature extraction, model training, calculating predictions and evaluation. The implementation of every single step starts with an exploratory analysis, during which the problem is analyzed and a decision about how to proceed with the implementation is made. In the case of data cleaning that means visualizing the data and assessing what problems need to be addressed e.g. missing data. This analysis has to be well documented as it provides insights on the thinking process. Next, the specific operation is implemented and added to the pipeline. Before running the finished phase, the functionality needs to be tested by writing appropriate unit tests. This is especially important when implementing manual and case-specific operations i.e. data cleaning or feature extraction. After running the functionality, an analysis of the obtained results is performed. Unit tests are only code specific, thus they are unable to examine whether the results are reasonable from the business point of view and are not to be used to draw conclusions. Moreover, an evaluative analysis might detect issues that were not noticed during the exploratory analysis. It is crucial to document this part as it contains conclusions regarding obtained results and provides transparency of the analysis.

In order to support the devised workflow, a specific set of tools has been selected. Data analysis is performed using Jupyter Notebooks – documents produced by the Jupyter Notebook app [15] which contain both source code and rich text elements. As they allow to run Python code, visualization and outcomes filtering can be interconnected with notes and commentary in a single document. Those notebooks can be also exported into .pdf reports to provide immutable snapshots of the conducted work. The code implementation follows several guidelines. All logic is contained within Python package later to be run from the main pipeline. Packaging allows the distribution of the project and enables others to make use of the created models. When incorporating a new functionality into the pipeline the immutability of the data has to be preserved and, as consequence, intermediate results should be stored in separate files. The pipeline is implemented and orchestrated using GNU Make [31] – a tool widely used and preinstalled on many Unix systems. It automatically detects which parts of the entire process have to be rerun, omits unnecessary computations and guarantees that all operations are executed in the correct order. The Makefile (make configuration file) is also used as the main script that automates the following processes: environment recreation, environment variables declaration, reports pdf conversion and documentation generation.

To assure the high quality of the software development, another group of tools has been chosen. Sphinx [5] is a Python documentation generation project which is able to create an HTML website based on the comments located in the source code. It is also possible to supplement the outcome with the description of modules and functions that are present in the package. The recreation of the environment is achieved by the use of the Python virtual environment management module named venv [25]. It allows the user to create an isolated Python environment which contains all dependencies necessary to run the developed project. Requirements are stored in a .txt file containing package names and their exact versions used during development. Additionally, code specific tools: yapf [9] for code formatting and mypy [18] for optional static typing are used in order to guarantee consistent conventions throughout the whole project. The integrity of the pipeline was checked by the continuous integration platform named Travis CI [34]. Running the whole process from the data download to the scores evaluation on the subset of data allowed for continued verification of pipeline stability.

4.2 Content-based approach

All implemented content-based approaches follow a specific list of steps:

1. Selection of relevant attributes that will be included in feature vectors. In the project descriptions and tags were chosen and different combinations of those features were considered.

2. Preparation of selected attributes by addressing issues found in the data.

3. Extraction of features using the vector space model as described in Section 2.1.1.

4. Selection of nearest neighbors based on the cosine similarity. The reasons for selecting the cosine similarity are explained in Section 2.1.2

The steps of data preparation and feature extraction that were realized in a different manner are described in depth in Sections 4.2.1 to 4.2.3. The models were implemented using the scikit-learn [21] – an open source machine learning library for Python.

4.2.1 Description based model

Book descriptions have been selected for feature extraction as they provide useful insight into the plot and are supposed to provide summarize the items of interest to the users.

Two approaches regarding proper nouns were considered as they behave as a double edged sword. Keywords e.g. *Harry Potter* or *Hunger Games* allow to easily identify books belonging to the same series. However in case of *Harry* if main characters of two books have the same name, they can get classified as similar due to the presence of that specific keyword even though the plots can be greatly different. As a result both types of descriptions – containing proper nouns and not containing proper nouns are used separately in the latter steps. Proper nouns detection was achieved by part of speech tagging – the process of classifying words into their parts of speech [4]. The POS tagger recommended by NLTK [24] – the Averaged Perceptron tagger was used in order to find proper nouns [10].

Punctuation and stop-words were removed from the descriptions as they act as noise in textual data. In order to normalize the text, the remaining words were stemmed and lemmatized using the English Stemmer and the Wordnet Lemmatizer respectively.

Examples of preprocessing results for both approaches regarding proper nouns are presented in Figures 4.1 and 4.2. Strange words endings can be observed in the preprocessed descriptions e.g. tini instead of tiny. However, this is not an issue as all words in the descriptions will be processed in the same way and will correspond to the exact same feature.

After preprocessing the length of the descriptions has been reduced by almost half as on average each cleaned description had 420 characters in case of the approach keeping proper nouns and 348

Harry Potters life is miserable. His parents are dead and hes stuck with his heartless relatives, who force him to live in a tiny closet under the stairs.	harri life his parent dead stuck heartless forc live tini closet

Figure 4.1: Example preprocessing results using the approach that keeps proper nouns

Harry Potters life is miserable. His parents are dead and hes stuck with his heartless relatives, who force him to live in a tiny closet under the stairs.	life his parent dead stuck heartless forc live tini closet

Figure 4.2: Example preprocessing results using the approach that removes proper nouns

characters when removing proper nouns. Extreme cases in which the preprocessing resulted in an empty description were excluded from further parts of the implementation.

Book descriptions are represented using the vector space model described in Section 2.1.1. Two different approaches to evaluating term weights have been used: word count and TF-IDF. The TF-IDF weights were not calculated using the equation presented in Equation (2.5), the IDF in scikit-learn is defined as in Equation (4.1).

$$idf_t = \log \frac{N}{df_t} + 1 \tag{4.1}$$

Terms with a zero IDF score will be entirely ignored, hence the reason for adding 1 to the IDF equation [7].

During feature extraction different n-gram ranges were considered. Eventually, n-grams no longer than trigrams were considered as book descriptions were not very long as stated in Section 3.2.

4.2.2 Tag based model

Book tags characterize books in a different manner. They are assigned by a variety of users and the amount of how many times a specific tag was assigned is quantified. As a result it is possible to measure e.g. 'how romantic' a book is.

Tags make a relevant feature as they are not overspecific, but at the same time, they provide useful insight e.g. *wizard*, *knitting*. Additionally, the low dimensionality allows to omit heavy computations.

Table 4.1: Feature vector calculation based on document's content.

(a) Example documents with data about assigned tags.

Document	Tag name	Count
d_1	romance	3
d_1	fantasy	1
d_2	science-fiction	4

(b) Feature vectors of the documents from Table 4.1a.

Document	Tag feature vector
d_1	$[0.75, 0.25, 0]$
d_2	$[0, 0, 1]$

Each book is represented as a vector composed of weights corresponding to a specific tag. Weights are calculated using the formula presented in Equation (4.2)

$$w_i = \frac{\text{number of times the tag } t_i \text{ was assigned to a given book}}{\text{number of all tags assigned to a given book}} \tag{4.2}$$

In order the explain the reason why the tag count is normalized, let us consider the following example: book A has been tagged a 100 times in total and book B a 1000 times, the *comic-book* tag was assigned to book A a 100 times and to book B 300 times. If raw tag counts were used, book B would be considered to be more *comic-book* then book A even though in this situation book A is a pure *comic-book* book B will be evaluated as the *more comic-book like*. An example calculation of the tag feature vector is presented in Table 4.1a.

Tag feature calculation example

Given a set of tags composed of *romance*, *fantasy* and *science-fiction*. Example tag feature vectors will be calculated for the data presented in Table 4.1a.

Assuming that the consecutive dimensions correspond to the tags in the following order (i) romance (ii) fantasy (iii) science-fiction the feature vectors will correspond to the values presented in Table 4.1a.

4.2.3 Multiple features model

Descriptions and tags allow to represent books from different perspectives. Both of those approaches offer different advantages. The idea was to combine those features in order to reduce flaws coming from those methods when they are used separately. In the case of the description based approach books with a very similar settings e.g. books about lawyers might be considered similar even though one book might be a romance and the other might be a thriller. This is where the tag features play a significant role as they are supposed to detect that difference and prevent those situations. From the other perspective, tags benefit from descriptions as text features pro-

Table 4.2: Precision and recall evaluation results for the content-based models.

Model	Precision	Recall
Tag based	0.033	0.241
Tags + TF-IDF	0.018	0.124
Tags + TF-IDF + No Proper Nouns	0.018	0.124
TF-IDF	0.007	0.047
TF-IDF + No Proper Nouns	0.006	0.037
TF + Tags	0.005	0.031
Count	0.004	0.029
Count + Tags + No Proper Nouns	0.003	0.020
Count + No Proper Nouns	0.003	0.019

vide additional information about the plot of the book and two romance novels with a different setting will not be considered as similar. The feature vector in this model is the concatenation of the feature vectors that were calculated according to the rules described in Sections 4.2.1 and 4.2.2.

4.2.4 Results

Content-based methods were evaluated based on the similar books data described in Section 3.2 which is treated as ground truth. The goal of those models was to find books that are most alike and to recommend books that are very similar to those that the user enjoyed in the past. In order to measure how efficient the implemented content-based models are in achieving that strategy the similar books data and the top 20 most similar books evaluated by the models were compared using the precision and recall performance metrics mentioned in Section 2.3.2. The obtained results are presented in Table 4.2. The tag based model turned out to be the most effective with a precision score almost twice as high as the second best approach – the combination of tags and TF-IDF features. The removal of proper nouns usually resulted in worse performance.

The probable reason why the tags model performed the best is the fact that tag features are much more specific than descriptions. The presence of such keywords e.g. 'wizards' allows to identify books with a similar setting. Tags are standardized which makes books easier to compare based on a specific set of features. Additionally, tags are assigned by multiple users, which leads to tags having a wider perspective than descriptions.

The TF-IDF model performed better than the word count model, which was expected based on the issues described in Section 2.1.1. The addition of tag features improved the performance of TF-IDF models whereas it did not improve the word count model. This might be caused by the improper scaling of features as tags are normalized and word counts are not, which leads to tag features not being into account when calculating similarity.

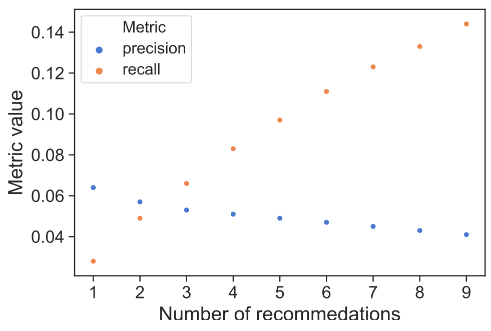


Figure 4.3: The precision and recall scores for the tag based model plotted over the number of recommended items.

The precision and recall scores for the best performing model and their relation to the number of recommended books is depicted in Figure 4.3. The optimal values of precision and recall were achieved when recommending from 2 to 4 books.

4.3 Collaborative filtering approach

The implementations of the collaborative filtering methods were largely based on the Surprise [12] library for Python. The framework provided a consistent interface for different algorithms and provided basic methods of rating prediction evaluation.

4.3.1 Methods selection

In the collaborative-filtering approach to the recommendation problem, the following methods were selected: SlopeOne, kNN, SVD.

SlopeOne algorithm served as the baseline for comparison between methods due to its simplicity and reasonable efficiency. In neighbor-based models family item-based variant of kNN was chosen. First of all, the numbers of users is much greater than the number of items which profoundly impacts computational complexity. Secondly, the more accurate predictions are achieved by smaller, but more trustworthy neighbors than the larger low-confidence group [28]. In the matrix factorization group of methods SVD was used. The more advanced variant named SVD++, which also takes into account implicit feedback, was rejected due to training time increase by factor of 30. Further description of this model can be found in [16].

The right-skewed distribution of ratings described in Section 3.3 was the reason for choosing the biased version of algorithms for both kNN and SVD models.

Table 4.3: Comparison of the RMSE values for different similarity measures used in kNN model evaluated on test sets.

Similarity measure	Mean RMSE
Pearson (baseline)	0.8018
Pearson	0.8395
Mean Squared Difference	0.8425
Cosine Vector Similarity	0.8456

4.3.2 Hyperparameters optimization

Machine learning algorithms are often parameterized. The process of optimizing such input values, called hyperparameters optimization, can lead to better results by leveraging data characteristics.

For kNN and SVD models we used multiple iterations of the Grid Search procedure to perform a search over all possible combinations of predefined sets of values for each parameters. This process was conducted using the training dataset only to avoid impacting the final validation stage. The data was divided into 5 equal parts (called folds) and for each combination of the parameters values following steps were repeated:

1. For each of the 5 folds:
    a) Train the model using other 4 folds.
    b) Evaluate the metric on remaining data.
2. Report the average value from the previous steps as a result.

The metric chosen for optimization was RMSE defined in Section 2.3.1. Nevertheless, training (also called fitting) and testing times were also taken into consideration.

kNN model

Neighbor-based model, including kNN, are using specific metric to calculate similarity metric between users or items. In our case we tested Cosine, Mean Squared Difference, Pearson or Pearson with baselines. As the ratings from the dataset were distributed around higher part of the scale(as described in Section 2.2.2) we assumed that metric taking into account baselines should give best results. Experiments showed that this was the case as presented in Table 4.3.

Taking into account baselines requires an estimation optimization method e.g. SGD or ALS. Both these methods achieved comparable results with default parameters, but the latter was trained faster. Therefore, we chose ALS.

Other important parameters for neighbor-based models are the lower and upper bounds for the number of neighbors used during calculation. Every book in the dataset had been rated at least

Table 4.4: Comparison of the RMSE values, training and testing phases duration for different number of factors evaluated on test sets using SVD model.

Number of factors	Mean RSME	Mean fit time [s]	Mean test time [s]
50	0.8352	139	29
100	0.8368	224	26
200	0.8396	388	27
500	0.8465	1246	28

8 times, so there were no issues with satisfying minimal requirement. When taking into account the maximal number of 30 or 40 neighbors similar results were obtained. As the higher number of considered items had negative impact on the testing time, the lower value was selected. In both cases the kNN algorithm was able to find, on average, maximal allowed number of neighbors.

As the last step of optimization procedure for the kNN model, baseline estimate learning parameters were selected. The default values for numbers of epochs and regularization parameters were already adjusted for 1-5 rating scale which was confirmed by experimental results.

The abovementioned process led to the improvement of the mean RMSE across folds on the training set from 0.8018 to 0.8014.

SVD model

In case of the SVD model the major parameter was the number of factors i.e. the length of the latent vector mentioned in Section 2.2.3. The obtained results were inconsistent with expectations, because a higher number of factors did not increase the accuracy. Therefore, the default value of 100 was selected for further steps.

The rest of the parameters was directly connected to the SGD technique. The right-skewness of the ratings distribution (described in Section 3.3) gave reasons to experiment with the initial mean of factors vectors distribution. The default value of 0 was replaced with 0.1 which achieved better results. In case of the regularization term, the learning rate and number of the epochs, default values were already selected for the rating scale of 1 to 5. Thus, only the number of epochs was raised from 20 to 25.

The parameters optimization process improved average RMSE across folds on training set from 0.8369 to 0.8206.

Table 4.5: The accuracy metrics results for SlopeOne, kNN, SVD models on ratings test set.

Model	MAE	RMSE
kNN	0.6008	0.7935
SVD	0.6196	0.8097
SlopeOne	0.6585	0.8471

4.3.3 Results

The final evaluation was performed using the ratings test set and to-read test set mentioned in Sections 3.2 and 3.3. The former includes both unary and scalar feedback – the fact of rating particular item by the user and the value of such action, when the latter contains only binary form – the items marked as *to read* by the user. Thus, the ratings test set can be evaluated by a wider range of metrics.

Ratings prediction accuracy

Ratings prediction accuracy was evaluated using the MAE and RMSE metrics described in Section 2.3.1. The results are presented in Table 4.5. As expected the SlopeOne model achieved the highest error rate. We anticipated the SVD model to achieve the best performance, although it scored slightly worse than the best one – kNN. This ordering differs from the results obtained e.g. during Netflix prize competition [3].

Without the parameters optimization process described in Section 4.3.2 the RMSE calculated for the ratings test set for kNN and SVD models was 0.7945 and 0.8248 respectively.

Top N recommendations effectiveness

One of the most common usages of recommender systems is presenting the user a list of recommendations. The length of such a list depends highly on the business context – the data available and the presentation interface. We have decided to evaluate our models using top 20 predictions viewed as the most interesting to the user i.e. with the highest rating estimate. Both test sets contain unary feedback which can be used in evaluation methods from Section 2.3.2 – precision and recall. There is one additional prerequisite – the definition of relevancy. Based on the mean rating of users in the dataset we assumed that any item with predicted rating higher than 4.0 is treated as relevant to the user. The obtained results are listed in Table 4.6. The SVD model was the best performer with results approximately twice higher than the kNN model. Moreover, for each model the scores achieved on the ratings and to-read test set differed by the factor of two.

The number of the recommendations presented to the user impacted the values of the precision and recall metrics as shown in Figure 4.4. The metrics dependency, stated in Section 2.3.2, is clearly

Table 4.6: The comparison of the precision and recall metrics results for SlopeOne, kNN, SVD models on to-read and ratings test sets. Top 20 recommendations were taken into consideration.

Model	Precision		Recall	
	to-read test set	ratings test set	to-read test set	ratings test set
SVD	0.0067	0.0113	0.0076	0.0199
kNN	0.0041	0.0055	0.0048	0.0096
SlopeOne	0.0021	0.0019	0.0022	0.0035

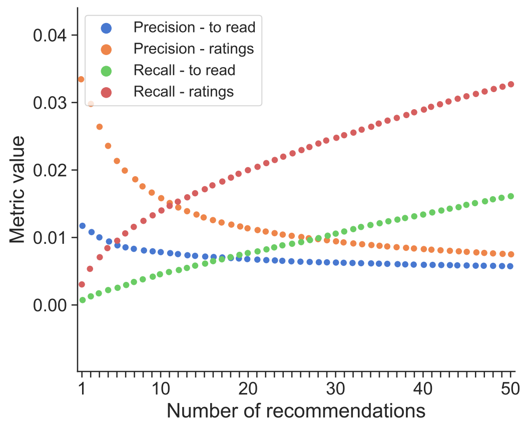


Figure 4.4: The precision and recall values for the SVD model plotted over different number of recommended items. Two datasets were used – ratings test set and to-read test set.

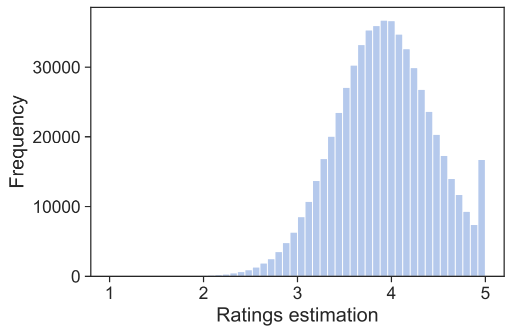
illustrated. Most balanced values were achieved for length of the recommendations list in the range of 10 to 20.

4.3.4 Comparison analysis for SlopeOne and kNN models

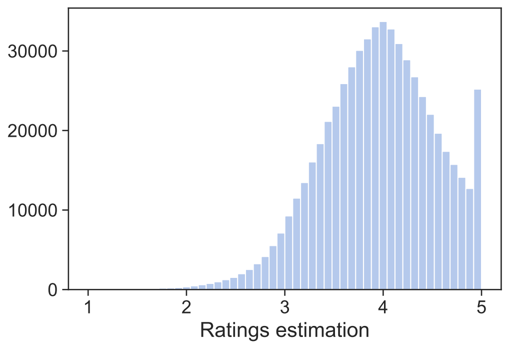
The SlopeOne and kNN models are based on the same principle of aggregating neighbour ratings. Thus, the difference between their results raise a need for the comparison analysis using the ratings test set.

Figure 4.5 illustrates the distributions for the SlopeOne and kNN models. We can see that the former model far more often predicted ratings with value close to general ratings mean of 3.93.

The analysis of the cases with the highest error clearly showed that the huge errors occurred in situations of predicting high (4-5) rating when user gave low ratings (1-2), but rarely the opposite. For the kNN model and estimates with the top 1000 highest errors such situation occurred in 99.2 % of the cases. At the same time, kNN prediction mistakes were not caused by the insufficient



(a) Distribution of ratings estimates calculated by the SlopeOne model using ratings test set.



(b) Distribution of ratings estimates calculated by the kNN model using ratings test set.

Figure 4.5: Distributions of ratings estimates calculated by the SlopeOne and kNN models using ratings test set.

number of neighbors – the overall mean average number of neighbors was 29.51 and declined to 29.01 amongst estimates with the top 1000 highest errors.

4.4 Web application

The results of research in recommendation system are hard to share and present comprehensively due to the volume of data, computation costs of the data processing and business context dependence. As a mean of tackling this problem a dashboard in the form of a web application written in the Dash framework [13] has been created. Dashboards are a clean, intuitive and visually rich medium of communication which present data in an interactive manner.

The idea was to create a tool that would allow users to explore the dataset and to provide a hands on experience of how the implemented recommendation models work. A high level of transparency is achieved as users can test models on the entire dataset themselves and form an opinion not only based on preselected examples. The obtained results can be viewed in the web application without repeating the whole data processing pipeline and serve as an interactive presentation of system's main features. Additionally, the web application facilitated development as it allowed for an initial evaluation of the implemented models based on intuition. Moreover, existence of the end component of the system on the early stage of development enforced early integration of all modules.

The web application allows to select a specific user from the dataset and see their ratings. Only ratings present in the training set are displayed. Then, it is possible to select one of the collaborative-filtering models to see top 20 recommendations for chosen user. Moreover, thhe application allows the selection of a specific book and content-based model to display a list of similar books.

5 Discussion and conclusions

In this paper, the description of the developed recommendation system for a novel dataset was presented. To provide books suggestions multiple methods based on two different approaches were used. The incremental development of the data processing scripts and recommendations models required multiple cycles of analysis, decision making and implementation stages. The execution of the whole process provides a real-world example of a workflow template for data science task. The selected set of tools and programming patterns provided a solution to the stages dependency resolution and reproducibility problem. Final scores were evaluated using various metrics to capture different aspects of the data as well as the strengths and weaknesses of the recommendations models. The created web application provides an interactive way to explore the dataset and visualize the characteristics of the trained models.

Based on the results presented in Sections 4.2.4 and 4.3.3 several conclusions regarding evaluation metrics and the general problem of recommendations can be drawn.

5.1 Discussion

Typical performance metrics of the information retrieval task like precision and recall are relative and, in our case, can be merely used to compare models, but not in terms of absolute values. The absence of a recommendation in the relevant set does not imply low quality of the recommendation. The same reasoning can also be applied to similar books data where issues like popular books being considered similar more often than others have a significant effect on the results. In ratings prediction methods and providing top recommendations list, the precision and recall metrics are also nuanced. Consider situation where there are two books, later read by user, namely book A with the rating of 5.0 and book B rated with 2.0 value, but written about current, social matter of great importance to the user. The system suggests the book A due to its' high estimate value. However, one can argue that the book B has far higher interaction probability and thus should have been recommended. The case of optimization for customer high ratings versus frequency of their interaction with the site is a matter of the business requirement.

Data quantity is important, however, it should be noted that quality and generalization potential can be more significant. This phenomenon is observed in the content-based methods when comparing tag features to description features. The goal of both tags and descriptions is to summarize books and capture their characteristics. Even though descriptions contain more data as compared to tags, they were not able to achieve better results. Tags proved to be of higher quality due to their standardization and being assigned by multiple users. At the same time descriptions are unstructured and written by a much smaller group of contributors.

In the Netflix prize competition various teams were able to achieve an RMSE score in the range of 0.85-0.90 [33]. Although we used a different dataset, the score of 0.7935 for kNN model can be considered high. Nevertheless, using RMSE as the only metric can not be sufficient to fully asses models' performance. When comparing scores using both metrics i.e. rating estimation and top recommendation list better cumulative score of the SVD model can be observed and thus its higher generalization potential. Moreover, results for the precision and recall metrics were highly dependent on test data characteristics – generally models scored much higher values on ratings test set than on to-read test set. On the other side, the chosen metric also profoundly changes the model training process. The parameters values optimization had a non-negligible impact on obtained results, especially considering SVD algorithm. Nonetheless, there is no guarantee that the change in metric value would be reflected in customer satisfaction. One of the ways to verify the impact could be A/B testing in the online scenario.

The recommendation problem is a very complex subject requiring expert knowledge from a variety of fields. The presence of subjectivity in the evaluation of recommendations constitutes a major issue and differ recommendation problem from regression or classification task. Moreover, recommendation methods are not easily transferable to different business domains – similar technique applied to various cases might result in entirely different outcomes. For example, in the context of news site, the freshness of the data is a major factor, because recommending outdated news is not desirable. Therefore, methods based only of text processing are expected to fail in such scenario. In conclusion, it is crucial to precede recommendation system development with identification of the business goals and determination of a performance metric which can reflect to what extent those targets are achieved.

5.2 Future work

During the development of the collaborative-filtering models, multiple problems related to the data volume arose. The need for parallel computation and distributed computing in terms of selected methods and workflow was duly noted. The SVD++ model was rejected due to computational costs even though preliminary tests showed more than 0.01 improvement in terms of the RMSE. On one side, future work could be based on leveraging more powerful computing environment. On the other side, the implicit feedback methods could also be used to take into account different type of information e.g. the fact that user did not rate some set of books. The hybrid approach to recommendation problem could combine both content-based models and collaborative-filtering ones in order to provide better results e.g. in case of the cold-start problem.

The online context of recommendation systems is not considered in this thesis. The problem of incorporating new users and items when the system is deployed remains unsolved. The further research could also be targeted towards considering the characteristics and challenges connected to online scenario – procedures of retraining models, different set of evaluation metrics and live performance and stability monitoring.

Acronyms

ALS	Alternating Least Squares
IDF	Inverse Document Frequency
kNN	k-Nearest Neighbors
MAE	Mean Absolute Error
NLTK	Natural Language Toolkit
POS	Part of Speech
RMSE	Root Mean Squared Error
SGD	Stochastic Gradient Descent
SlopeOne	Slope One
SVD	Singular Value Decomposition
TF	Term Frequency
TF-IDF	Term Frequency - Inverse Document Frequency

Bibliography

1. C. C. Aggarwal et al. *Recommender systems*. Springer, 2016. DOI: 10.1007/978-3-319-29659-3.
2. B. Baumer, M. Cetinkaya-Rundel, A. Bray, L. Loi, and N. J. Horton. "R Markdown: Integrating a reproducible analysis tool into introductory statistics". *arXiv preprint arXiv:1402.1894*, 2014.
3. J. Bennett, S. Lanning et al. "The Netflix Prize". In: *Proceedings of KDD cup and workshop*. Vol. 2007. New York, NY, USA. 2007, p. 35.
4. S. Bird, E. Klein, and E. Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. "O'Reilly Media, Inc.", 2009.
5. G. Brandl and the Sphinx team. *Sphinx – documentation generation tool*. URL: http://www.sphinx-doc.org (visited on 12/01/2019).
6. P. Covington, J. Adams, and E. Sargin. "Deep neural networks for Youtube recommendations". In: *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM. 2016, pp. 191–198.
7. scikit-learn developers. *Scikit-learn documentation – TfidfTransformer function*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html (visited on 12/01/2019).
8. C. Fellbaum, ed. *WordNet: An Electronic Lexical Database*. The MIT Press, 1998. DOI: 10.7551/mitpress/7287.001.0001.
9. Google LLC. *yapf - Python code formatter*. URL: https://github.com/google/yapf (visited on 12/01/2019).
10. M. Honnibal. *A Good Part-of-Speech Tagger in about 200 Lines of Python*. 2013. URL: https://explosion.ai/blog/part-of-speech-pos-tagger-in-python (visited on 12/01/2019).
11. A. E. Howe and R. D. Forbes. "Re-considering neighborhood-based collaborative filtering parameters in the context of new data". In: *Proceeding of the 17th ACM conference on Information and knowledge mining - CIKM '08*. ACM Press, 2008. DOI: 10.1145/1458082.1458345.

12. N. Hug. *Surprise, a Python library for recommender systems*. URL: `http://surpriselib.com` (visited on 12/01/2019).

13. P. T. Inc. *Dash - Python framework for building analytical web applications*. 2015. URL: `https://plot.ly/products/dash/` (visited on 12/01/2019).

14. D. Jurafsky. "Speech and language processing: An introduction to natural language processing". *Computational linguistics, and speech recognition*, 2000.

15. T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. "Jupyter Notebooks – a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides and B. Schmidt. IOS Press. 2016, pp. 87 –90.

16. Y. Koren. "Factorization meets the neighborhood". In: *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*. ACM Press, 2008. DOI: `10.1145/1401890.1401944`.

17. D. Lemire and A. Maclachlan. "Slope One Predictors for Online Rating-Based Collaborative Filtering". In: *Proceedings of the 2005 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, 2005. DOI: `10.1137/1.9781611972757.43`.

18. *mypy – optional static type checker for Python*. URL: `http://mypy-lang.org/` (visited on 12/01/2019).

19. Nature. *Nature journal addressing the reproducibility issue*. 2013. URL: `https://www.nature.com/news/announcement-reducing-our-irreproducibility-1.12852` (visited on 12/01/2019).

20. K. Park, B. Kim, T. Park, and B. Zhang. "Uncovering response biases in recommendation". In: *Multidisciplinary Workshop on Advances in Preference Handling, Papers from the 2014 AAAI Workshop, MPREF@AAAI*. 2014.

21. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* 12, 2011, pp. 2825–2830.

22. M. Porter. *The English (Porter2) stemming algorithm*. URL: `http://snowballstem.org/algorithms/english/stemmer.html` (visited on 12/01/2019).

23. M. Porter. "An algorithm for suffix stripping". *Program* 14:3, 1980, pp. 130–137. DOI: `10.1108/eb046814`.

24. N. Project. *NLTK documentation – pos_tag*. 2019. URL: `https://www.nltk.org/_modules/nltk/tag.html#pos_tag` (visited on 12/01/2019).

25. Python Software Foundation. *venv - -virtual environment managment module*. URL: `https://docs.python.org/3/tutorial/venv.html` (visited on 12/01/2019).

26. M. Ramos, A. M. Calvão, and C. Anteneodo. "Statistical Patterns in Movie Rating Behavior". *PLOS ONE* 10:8, 2015. Ed. by W.-X. Zhou, e0136083. DOI: `10.1371/journal.pone.0136083`.

27. *Reproducibility in Science A Guide to enhancing reproducibility in scientific results and writing*. URL: `https://ropensci.github.io/reproducibility-guide/` (visited on 12/01/2019).

28. F. Ricci, L. Rokach, and B. Shapira. "Recommender Systems: Introduction and Challenges". In: *Recommender Systems Handbook*. Springer US, 2015, pp. 1–34. DOI: `10.1007/978-1-4899-7637-6`.

29. H. Schütze, C. D. Manning, and P. Raghavan. *Introduction to information retrieval*. Vol. 39. Cambridge University Press, 2008.

30. J. E. Solsman. *YouTube's AI is the puppet master over most of what you watch*. 2018. URL: `https://www.cnet.com/news/youtube-ces-2018-neal-mohan/` (visited on 12/01/2019).

31. R. M. Stallman, R. McGrath, and P Smith. *GNU Make: A Program for Directed Compilation*. Free Software Foundation, 1991.

32. R. Tatman, J. VanderPlas, and S. Dane. "A practical taxonomy of reproducibility for machine learning research", 2018.

33. A. Töscher, M. Jahrer, and R. M. Bell. "The BigChaos solution to the Netflix Grand Prize". *Netflix prize documentation*, 2009, pp. 1–52.

34. Travis CI, GmbH. *Travis CI – continuous integration service*. URL: `https://travis-ci.com` (visited on 12/01/2019).

35. H. Wickham and G. Grolemund. *R for data science: import, tidy, transform, visualize, and model data*. O'Reilly Media, Inc., 2016.

36. Z. Zając.*goodbooks-10k – books dataset*. 2018. URL: `https://github.com/zygmuntz/goodbooks-10k` (visited on 12/01/2019).

List of Figures

List of Tables

Appendices

1 Work division

The work division between team members is listed in Tables 1 and 2. All programming tasks were peer-reviewed using GitHub platform. The sections in thesis paper were also subject to review.

Table 1: The work division in the project.

Main contributor	Project
Paweł Rzepiński	Data cleaning and preparation
	Collaborative filtering models: interfaces, implementation, evaluation, tests
	Hyperparameters optimization
	DevOps: Travis CI integration, documentation generation, notebooks export
Ryszard Szymański	Development pipeline study and template
	Natural language processing and feature extraction
	Content-based models: interfaces, implementation, evaluation, tests
	Web application

Table 2: The work division in thesis paper.

Main contributor	Thesis
Paweł Rzepiński	Introduction
	Collaborative filtering approach: methods, implementation
	Web application
Ryszard Szymański	Content-based approach: methods, implementation
	Reproducibility
	Workflow
Both	Data
	Conclusions

2 Project's manual

2.1 Disc content

The directory structure of the CD is described in Figure 1.

```
/
├── doc
│   ├── abstract_EN.pdf
│   ├── abstract_PL.pdf
│   ├── thesis.pdf
│   └── title_page.pdf
└── src
    ├── data
    │   ├── external - Data from third party sources.
    │   ├── interim - Intermediate data that has been transformed.
    │   ├── processed - The final, canonical data sets for modeling.
    │   └── raw - The original, immutable data dump.
    ├── booksuggest - Source code of the recommendation module.
    ├── docs - Codebase documentation.
    ├── Makefile - Makefile with commands like ‘make data‘, ‘make models‘, ‘make scores‘.
    ├── models - Trained and serialized models, model predictions.
    ├── notebooks - Jupyter notebooks.  Naming convention is a number (for ordering), the
    │               creator’s initials, and a short ‘-‘ delimited description, e.g.  ‘1-
    │               rzepinskip-initial-data-exploration‘.
    ├── Readme.md - The top-level README for developers using this project.
    ├── reports - Generated analysis as PDF files.
    ├── requirements.txt - The requirements file for the web application.
    ├── requirements-dev.txt - The requirements file for reproducing the analysis environment.
    └── setup.py - Project’s main module.  Can be installed with pip command.
```

Figure 1: CD directory structure.

2.2 Setup manual

System and package requirements to run the project:

- UNIX based system
- GNU Make
- Python 3.7
- pip

All commands mentioned below should be run from the project's root folder. Use `make help` to display help information about available commands.

Viewing results

1. Create a virtual environment:
   ```
   make create_environment
   ```
2. Activate the virtual environment:
   ```
   source rs-venv/bin/activate
   ```
3. Install packages required by the web application:
   ```
   make app_requirements
   ```
4. Run the app:
   ```
   make app
   ```
5. Enter the web page address displayed in the console. Web application should be accessible at `http://127.0.0.1:8050/`.

Reproducing the analysis

1. Create a virtual environment:

   ```
   make create_environment
   ```

2. Activate the virtual environment:

   ```
   source rs-venv/bin/activate
   ```

3. Install packages required for development:

   ```
   make requirements
   ```

4. Download the raw data:

   ```
   make data
   ```

5. Train models:

   ```
   make models
   ```

6. Evaluate models:

   ```
   make scores
   ```

- When using the whole dataset the make models command takes about 20 minutes, make scores lasts more than 12h.
- To check pipeline on the small subset of data use `TEST_RUN=1` parameter when running make commands. Then, the whole process should take about 5 minutes.
  Example: `make scores TEST_RUN=1`
- To utilize make's parallelization use `-j <n_jobs>` parameter where `<n_jobs>` specifies the number of parallel jobs run. Most often, `n_jobs` should be equal to the number of cores in the processor, although there are also some RAM requirements when using whole dataset.
  Example: `make scores -j 2`