DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Introduction to Recommender Systems and their Applications

Katjana Kosic

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Introduction to Recommender Systems and their Applications

Einführung in Empfehlungsdienste und ihre Anwendungen

Author:	Katjana Kosic
Supervisor:	Prof. Dr. Christian Mendl
Advisor:	Dr. Felix Dietrich
Submission Date:	15.03.2022

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.03.2022 Katjana Kosic

Acknowledgments

First and foremost, I would like to express my gratitude towards my thesis advisor Dr. Felix Dietrich for supporting and guiding me throughout the process of writing the thesis and always being eager to help.

I would like to thank my family and friends for being great supporters. Their motivation and belief in me helped me during all ups and downs

Abstract

The usage of recommender systems and their impact on everyday life has gained a lot of importance in recent years. The primary objective is to guide users to the discovery of new products and services by providing suggestions based on already known user interests and ratings. This is an essential feature in the digital world, since users tend to easily get overwhelmed by choice if given a large number of items to choose from. There is a big variety of areas in which recommender systems are used, like product recommendations for online shopping stores or artist-based song recommenders for music streaming platforms. As the datasets for the respective recommendation tool can differ in size and structure, it is crucial to find the best suiting recommendation technique for rating predictions out of a big collection of different approaches and methods. A complication that can occur in recommender systems is the cold-start problem, which refers to an issue in which it is challenging for the system to infer interactions between users and items due to insufficient information.

This thesis introduces recommender systems in general and how they can be approached with different techniques as well as possible solutions for the cold start problem. The presented dataset describes a case of the cold-start problem, which needs to be resolved in recommender systems. For this purpose, this thesis describes an analysis of existing recommendation methods and discusses their suitability for solving the cold-start problem for the arXiv dataset. The results achieved in this thesis can provide essential insights into how to deal with the cold-start problem in datasets that do not contain any user records.

Contents

1 Introduction

The voluminous material available on the Web and in digital libraries, along with its dynamic and varied nature, has resulted in an ever-increasing difficulty in locating users' preferences and needs in the most efficient manner possible. Users require personalized assistance in searching through huge amounts of available data based on their interests and preferences. As a technique of tailoring material for users, several information sources utilize recommender systems (Lops, de Gemmis, & Semeraro, 2011). The influence and importance of recommender systems on daily digital life has increased in recent years and can be found in many online content delivery services that provide recommendations.
As users tend to easily get overwhelmed by choice, recommender systems aim to help users with their decision by narrowing the decisions into fewer recommendations and, thus, help users discover new products that might match their interests. This is done by predicting each user's rating for a certain item (Ramlatchan et al., 2018). Due to the inevitability of recommendation service applications in various fields like e-Commerce, product recommendation, online advertisement and so on, a big variety of algorithms and methodologies for data filtering and recommendation giving has been discovered in order to suggest the optimal items to the user (Asanov, 2011; Rocca, 2019)

Recommendation systems are built based on one of the three main approaches: collaborative filtering, content-based methods and hybrid approaches. In collaborative filtering the rating predictions are made based on past user interactions, which are stored in an user-item matrix. The preferences, that users have for certain items, can either be expressed as implicit or explicit feedback.
In comparison to collaborative filtering, content-based approaches additionally take both user and item information into account, which describes both user characteristics and item content.
Content-based and collaborative filtering based recommenders can be combined in order to build a hybrid recommender system, which leads to more precise recommendations (Ramlatchan et al., 2018). The choice of the most suitable approach is task-dependent.

In order to get a general overview of recommender systems and how they can be implemented, this thesis is structured as follows. The following section introduces general challenges, that recommender systems face, as well as various dataset types and known recommendation systems applications. Section 3 focuses on the structure of recommender systems. It describes three possible methodologies to implement recommender systems and adresses the cold-start problem in detail and how to solve it. In addition to that, the section describes the usefulness of knoelwdge graphs and similarity measures recarding recommender systems. This is followed by a case study, which is conducted to propose a solution for the cold-start problem for a given dataset. The results will be summarized and discussed in section 4.

2 State of the Art

The following section gives insights into the most common shortcomings and challenges that recommender systems face, as well as well-known recommender systems that are used in practice.

2.1 Challenges for Recommender Systems

In an increasingly competitive online market, recommender systems represent a fundamental tool for increasing user satisfaction and bring advantages for business growth (Blumenfeld, 2021). Nevertheless, recommender systems still have challenges and issues that can reduce the degree of accuracy of rating predictions. In the following, the most common issues will be presented.
Sparsity issue: Usually, the majority of all system users do not rate most of the items that are available due to the lack of interest or users simply not wanting to rate anything. This causes the ratings matrix, also called the user-item matrix, to become sparse, which makes it difficult for collaborativ filtering based recommender systems to predict user ratings and identify similar user groups (B. Kumar & Sharma, 2016; Yuan et al., 2016), causing an unsatisfactory recommendation (Yi, Zhong, Chen, & Jie, 2020). The most extreme case of sparsity in such systems is called cold-start, which will be discussed in more detail in section 3.5.
Scalability problem: Another crucial shortcoming that recommender systems face is the problem of scalability. The amount of dynamically changing input data for recommender systems that is generated by user-item interactions is increasing as new users and items are added to the system, which can lead to less accurate prediciton results. This leads to scalability of algorithms with real-world datasets being a major issue for recommender systems (Tyagi, 2021).
Privacy issue: In order to produce accurate recommendations it is often required for a recommender system to obtain presonal user information and preferences in order to receive the most optimal and individualized outcome. Nevertheless, this causes concerns about data privacy and security and leads users to being hesitant about feeding the system with personal data (Tyagi, 2021).
Cold-start problem: The cold-start problem describes a situation in which it is difficult to recommend a newly added item to the users due to the lack of user interaction with

this item. Similarly when a new user joins the system it is difficult to recommend anything to the user since there are no rating records present (Yuan et al., 2016). This shortcoming will be discussed in more detail in section 3.3.

2.2 Dataset Types

Due to the wide range of application fields of recommender systems there exist different kinds of dataset types to work with. As recommender systems aim to predict and estimate user preferences or ratings, it is crucial that the dataset contains information about users' preferences and, if required, user information data like age, sex, location, etc.
User preference can be expressed by two different categories. Explicit feedback is given directly by the user in form of e.g. a rating ranging from 1 to 5, a direct feedback, reviews or likes that express whether the user liked the item or not. Implicit feedback is collected when a user interacts with items and describes indirect user preferences like clicks, views, queries, etc. For instance, if a user bought a product, it is not clear whether the user liked the product or not.
An example for a well known dataset for recommender systems is the MovieLens 25M dataset, which describes 5-star ratings of more than two million user on over 60.000 movies and is therefore considered as explicit feedback.
In social networks, user preferences are considered implicit and are percieved by feedback in forms of clicks, views or purchases. Despite this, explicit feedback can also be found in social networks in forms of likes or comments (Shetty, 2019).
Although recommender systems generally heavily rely on user information and preferences to make accurate predictions, there are still datasets that have no user information at all. Such datasets usually consist only of item information, which makes it difficult for the system to recommend anything.

2.3 Known applications of Recommender Systems

B. Kumar and Sharma (2016) stated that recommender systems have been designed and developed to recommend a variety of items that can be broadly classified into several domains of interest, including movies, books, web pages, clothing, etc. Recommender systems incorporated in commercial applications recommend products to customers and encourage them to buy things that are in their interest field, ranging from technichal products to clothing and many other. The product recommendations are based on the customer's demographic information or previous shopping behavior and search histories. Recommender systems are also applied in movie or music recommendation

platforms.
As recommender systems seem to be unavoidable nowadays, many major companies like Netflix, Google, Amazon or Instagram have taken advantage of these systems. For instance, Netflix is an American company that provides users with a streaming platform for movies and series on a fee basis. It is based on recommending suggestions for new movies and series to users that they might not have discovered yet. This helps users choose titles, that match their personal preference from Netflix' large movie and series catalog. Their recommendation algorithm takes user information, time and location data, search history and many other data points into consideration. In 2006 Netflix announced an open competition called the "Netflix Prize", which challenged the contestants to build the most accurate collaborative filtering algorithm for predicting the users ratings for movies based on a data set of 100,480,507 ratings that 480,189 users provided for certain movies (Koren, Bell, & Volinsky, 2009).
Amazon is the largest online retailer and webservice provider for e-commerce worldwide. It is well known for its personalized recommendations, which helps customers discover new products from their item catalogue they might not have found otherwise. Amazon's recommendation algorithm is also based on collaborative filtering and recommends new items based on the customers current search history, purchases and past ratings, but not including the prediction of ratings for recommended items (Schafer, Frankowski, Herlocker, & Sen, 2007; Smith & Linden, 2017).
Google is an internet search engine that helps users obtain information about anyone or anything in the world. For each search term, Google will offer several websites that may contain the requested information. Besides, Google performs product recommendations by taking user data like what kind of user views which products, users' search histories, user purchases and product attributes like price, categories and brand into account.
Recommender systems are also used in social media platforms like Instagram. Instagram is a platform, which allows users to post pictures, stories and videos, which can be liked, commented and viewed by other users, also called followers. It has a feature called the *explore page*, which is visited daily by the Instagram community in order to discover new pictures, videos and stories that match their interests. Based on the posts that the user and the users they follow like, see and comment on, new posts and stories that might be of interest to the user are suggested on the explore page.

3 Introduction to Recommender Systems and their Applications

The intent of this chapter is to present the general structure of recommender systems as well as possible implementation methods. In doing so, it also establishes the connection to knowledge graphs and similarity measurements. The cold-start problem, which is in the foreground of this work's problem statement, is approached on the basis of a data set while presenting a possible solution.

3.1 Recommender Systems

The aim of recommender systems is to predict user ratings based on their past behavior and interactions with items, in order to recommend new items that fit their actual interests. Another way of recommending new items to users is by recommending additions or extensions, that might complement an item the user recently interacted with. Ricci, Rokach, and Shapira (2010) describe the term *item* as a general term used to describe what the system recommends to the user, where each recommender system is focused on a specific type of item like movies, songs, products in an online retailer, news articles, etc.
In general, recommender systems can be implemented in three different ways, namely collaborative filtering, content-based methods and hybrid approaches. The following sections aim to describe these approaches in more detail.

3.1.1 Collaborative Filtering

The objective of collaborative filtering algorithms is to generate a group of users by making the predictions solely based on past interactions between users and items in order to recommend personalized items. This is done by detecting similar users or items based on users past behavior, assuming they tend to agree in the future as well (Luo, 2018; Ramlatchan et al., 2018). The interactions, which are sufficient enough to group similar users and items, are stored in a so called user-item interactions matrix (Rocca, 2019). The entries in the matrix can be interpreted differently, depending on the context. Taking movie streaming providers as an example, an entry in the matrix would

be a movie rating of type integer given by the user. For e-commerce recommenders, the entries would represent booleans, whether the user clicked a recommended item or not (Rocca, 2019).
Collaborative filtering can be divided into two sub-categories: memory-based and model-based approaches.

Memory-based collaborative filtering works with either all recorded user-item interactions, that are stored in the interactions matrix, or simply as a sample of the interactions. The main steps of a memory-based algorithm include the similarity calculation between users and items, taking the user-item interactions matrix in account, and the prediction of user rating for a recommended item, which can either be a single specific recommendation or a list of the top N items (M. Sharma & Mann, 2013)).

Memory-based approaches can be classified into user-based and item-based collaborative filtering.
On the one hand, user-based systems aim to identify users with similar interactions profiles to the queried user and compute their similarity by comparing their past ratings on the same items. Hence, the algorithm can compute the predicted rating for a particular item based on a user's k-nearest-neighbors (Rocca, 2019; M. Sharma & Mann, 2013). Therefore, each user can be represented by its vector of interactions with various items. Two users can be considered close neighbors by a specific similarity measure, if they provide similar interactions or ratings on the same items. The most popular items among the k-nearest-neighbors, that the reference user has not yet interacted with, will then be recommended to the user (Rocca, 2019).
Situations where a user, who has only one interaction in common with the reference user, has a perfect match and is regarded closer than a user who has a lot more common interactions but agrees on only 98 percent of them need to be avoided (Rocca, 2019).
On the other hand, item-based collaborative filtering computes the similarity of two items by comparing the rating they received made by the same user. Two items are considered similar if the majority of users that interacted with both items interacted with them likewise. Similar to user-based collaborative filtering, the similarities between the best rated item and the rest of the items can be calculated. By keeping the k-nearest-neighbors of the reference item, that are new to the user, the user can receive multiple item recommendations. According to M. Sharma and Mann (2013), the predicted interaction of an item with a user is obtained as a weighted average of the interactions of the user on items, weighted by the similarity between those items.

In comparison to memory-based, model-based collaborative filtering algorithms assume that the representation of users and items are built based on a model (Rocca, 2019). By using machine learning algorithms, the goal is to predict user ratings for still unrated items. The recommendations are made by estimating statistical model parameters for user ratings rather than using all available data to make predictions. These ratings are then used to learn a model of user preferences in order to make accurate rating predictions. Although this approach avoids the sprasity problem, the model requires a lot of time to learn the predictions. Based on the user's ratings on previously rated items, the probabilistic technique is used to calculate the likelihood that the user would give a specific rating to a new item (M. Sharma & Mann, 2013). Collaborative filtering based methods are the most common recommendation algorithms. An example for a popular collaborative filtering system is MovieLens, a filtering system for movies in which the user can rate a set of movies with 1 to 5 stars. MovieLens can then recommend a movie to a particular user, which was liked by his community and predict how this user might rate the recommended movie.

Collaborative Filtering can be realized by the two major concepts Nearest Neighborhood Algorithms and Latent Factor Models. The computation of associations between items or users is the focus of neighborhood approaches. For instance, the item-oriented technique assesses a user's preference for an item based on the same user's assessments of "neighboring" items. Alternatively to neighborhood approaches, latent factor models aim to explain ratings by describing both items and users based on implicit factors inferred from the ratings patterns (Koren et al., 2009). User-based algorithms generate rating predictions for a particular item by analyzing the ratings the item got from the users similar to the queried user, which are called the user's neighborhood.

There are plenty of frameworks that perform collaborative filtering. Apache Spark (Apache Software Foundation, 2014) is an open-source framework for cluster computing, data engineering and science and machine learning. Spark ML supports model-based collaborative filtering by using the matrix factorization algorithm Alternating Least Squares (ALS) to learn latent factors for predicting missing user-item interactions.

3.1.2 Content-based Methods

Content-based recommender systems aim to suggest products that are similar to those that a user has positively interacted with in the past. Compared to collaborative filtering, content-based methods require an additional amount of information about the content and features of the items and user interests besides the user-item interactions matrix. In content-based systems users usually have a profile, which includes user

information like age, location and interests in order to describe their characteristics and preferences, or the systems obtains user information by letting the user fill out a survey that exposes their interests and personal information. By comparing the properties of the user profile with the attributes of an item, new interesting items can be recommended to the user (Lops et al., 2011). Content-based systems keep track of the content or features of each item being recommended, which is used to suggest items that are comparable to those that the user previously favored based on how similar particular items are to one another or how similar they are to the user's preferences (de Campos, Fernández-Luna, Huete, & Rueda-Morales, 2010).
A user profile is a structured representation of the user's interests that is used to suggest new products that might be relevant or interesting to them. Such profiles are usually created automatically based on user feedback and define types of items the user positively interacted with (M. Sharma & Mann, 2013). An item profile describes its most important features, e.g. a song can be described by the title, the artist, the genre and the length.
The main recommendation step is to match the attributes of a potentially interesting item against the stored attributes of the user. The more accurately the profile reflects the interests of the user the more accurate the result of the recommender will be (Lops et al., 2011).

Lops et al. (2011) presented in their chapter about content-based recommender systems, that the recommendation process can be performed in three main steps.
Content Analyzer: Unstructured data needs pre-processing to extract information that will be necessary for the recommendation. The content analyzer translates the content into a suitable format that will be passed on to the next processing step.
Profile Learner: This step obtains information about user preferences and creates a profile, after generalizing the collected data using machine learning techniques.
Filtering Component: In the last step, suggestions are made by matching the user's profile representation against the item representations.

Because keywords are commonly used to characterize content in text-based systems, the importance of word k_i in a document d_j is decided by some weighting measure $w_{i,j}$. The term frequency - inverse document frequency (TF-IDF) metric is a well-known measures for establishing keyword weights in information retrieval. The idea behind the weight is that the higher the weight of a term, the more frequently it appears in that document than in the other document, and so is more significant to its content (M. Sharma & Mann, 2013).

3.1.3 Hybrid Approaches

Content-based and collaborative filtering approaches can be combined to a hybrid recommendation approach, which is often associated with a much better recommendation precision since it benefits from their complementary advantages. There are several ways to combine the two above presented approaches. One method is to implement collaborative filtering and content-based systems seperately and afterwards combine their predictions. Another way is to incorporate features of content based systems into a collaborative filtering approach or the other way around or constructing a model that combines both collaborative and content-based features (M. Sharma & Mann, 2013).
Hybrid recommender approaches aim to avoid shortcomings of individual recommender approaches, like the new item problem of collaborative-filtering, and to gain better performance.
There are different hybridization methods such as weighted hybrid recommenders or hybrid systems using switching mechanisms. In weighted hybrid recommenders, internal models (e.g. a collaborative filtering model and a content-based model) can be defined that take dataset as the input. The weighted recommender system then integrates the outputs from each of the models into static weightings, which will remain constant across the trainig and test set (Chiang, 2021).
What distinguishes swtiching-based recommenders from weighted recommenders is that depending on the situation and criteria only one of the intern recommender models is selected, meaning that the hybrid system switches between the recommendation techniques. In Figure 3.1, the additional layer for switching between recommenders is integrated in the hybrid recommender system.

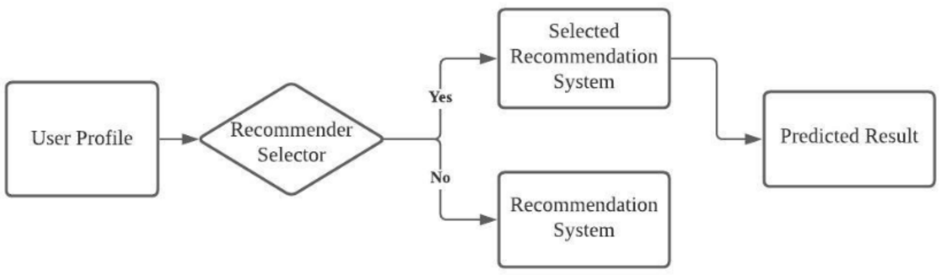


Figure 3.1: Example of a switching-based hybrid recommender system (Burke, 2002; Chiang, 2021).

3.2 Turning Datasets into Knowledge Graphs

There is no commonly accepted definition for knowledge graphs since it depends on the problem statement. For the purpose of this work, knowledge graphs are defined as a graph-structured knowledge base, which describes the relationship between entities. The two main steps of building a knowledge graph include knowledge extraction (during which subject-predicate-object (SPO) triples are extracted using natural language processing and entities are recognized and linked) and graph construction (Kishore, 2019).
Depending on the task, the edges of the graph can represent different kinds of relationships between the vertices, like similarites between text documents. There are numerous frameworks that can realize a graph structure, e.g. Neo4j (Neo4j, 2010) or spaCy (Honnibal, 2015). Neo4j is an open-source graph database that stores data in graph structures. The data can be stored as nodes, edges, that can be labeled, and attributes. The queries are executed in Cypher, a declarative query language. Spacy is an open-source library for Natural Language Processing, that can extract data from unstructured data and represent it as a graph sturcture.

Finding items or users that are similar in a recommender system relies on similarity computing. It is determined by distance metrics. Data points that are closest to each other are considered the most similar, while farthest foints are considered least similar. For instance, two movies are comparable in terms of similar gernre, cast or plot.
Similarity measures can be divided into string-based (term-based), corpus-based and knowledge based (similarity, relatedness) measures (Gomaa & Fahmy, 2013). For the purpose of this thesis, the focus will be on string-based similarity measures. This section aims to describe the most common similarity measures for the items q and d, which represent the queried paper and respectively other research papers, in more detail. q_t and d_t are vectors over the set of terms $T = \{t_1, ..., t_m\}$.
The following similarity measures were all presented by Benard Magara, Ojo, and Zuva (2018):

Cosine Similarity: This measure takes two vectors and computes the cosine of the angle between them in order to calculate the similarity. The smaller the angle between the vectors, the more similar they are. If the dot product of the queried vectors equals 0, it can be interpreted as a strong dissimilarity between them:

$$SIM_{cos}(\vec{q_t}, \vec{d_t}) = \frac{\vec{q_t} \cdot \vec{d_t}}{|\vec{q_t}| \times |\vec{d_t}|} \tag{3.1}$$

Euclidean Distance, also called L2-norm, is commonly used in machine learing practices for regularization and normalizing. It measures the degree of similarity between two data points:

$$D_{eucl}(\vec{q}_t, \vec{d}_t) = \sqrt{\sum_{t=1}^{m} |\vec{q}_t - \vec{d}_t|^2} \tag{3.2}$$

Jaccard Coefficient: According to Benard Magara et al. (2018) this measure calculates similarity as the intersection of vectors divided by the union of the vectors. The result varies in a range from 0 to 1, 1 meaning that the queried vectors are equivalent:

$$SIM_{jacc}(\vec{q}_t, \vec{d}_t) = \frac{\vec{q}_t \cdot \vec{d}_t}{|\vec{q}_t|^2 + \vec{d}_t^{\,2} - \vec{q}_t \cdot \vec{d}_t} \tag{3.3}$$

Pearson Correlation Coefficient: In this case, Pearson Correlation is utilized to measure the correlation between two text documents, ranging from -1 to +1, +1 meaning that there is a strong correlation. Here, TF_q and TF_d refer to the terms that are present in q and d:

$$SIM_{pear}(\vec{q}_t, \vec{d}_t) = \frac{m \sum_{t=1}^{m} \vec{q}_t \times \vec{d}_t - TF_q \times TF_d}{\sqrt{(m \sum_{t=1}^{m} \vec{q}_t^{\,2} - TF_q^2) \cdot (m \sum_{t=1}^{m} \vec{d}_t^{\,2} - TF_d^2)}} \tag{3.4}$$

3.3 Evaluation Metrics

When evaluating the performance of a recommendation algorithm, it is more interesting to observe the performance on new data rather than on old data. Evaluation metrics can be classified into probabilistic, qualitative, ranking and user satisfaction metrics. While qualitiative evaluation aims to reduce the number of errors during recommendation, ranking metrics describe how well the recommended items are ranked. User satisfaction metrics evaluate the level of user satisfaction during empirical experiments (B. Kumar & Sharma, 2016). This section focuses on probabilistic error metrics that evaluate the reliability of predictions generated by recommendation algorithms (Cremonesi, Turrin, Lentini, & Matteucci, 2008).

Cremonesi et al. (2008) proposed two general metrics to evaluate recommender systems called Error Metrics and Classification Accuracy Metrics.

Error Metrics measure the accuracy level between the predicted rating p_i and the actual user rating a_i.

- *Mean Squared Error (MSE):* This error metric assesses the average squared difference between the predicted rating and the actual ratings. The lower the MSE, the more accurate the rating prediction is

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2. \tag{3.5}$$

- *Root Mean Squared Error (RMSE):* The difference of this metric to the MSE is that the additional root in the equation causes the MSE to have the same dimension as the predicted value:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2}. \tag{3.6}$$

- *Mean Absolute Error (MAE):* The MAE is more robust to data with outliers and measures the gradient of accuracy between the predicted and the outcome:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|p_i - a_i|. \tag{3.7}$$

When requiring a ranked result of items, e.g. the top-N-recommendations, the above mentioned metrices might not be the best choice, since they are not meaningful enough to evaluate classification tasks that are needed in certain recommendation systems, which is why classification metrics are much more suiting (Cremonesi et al., 2008).

Classification Accuracy Metrics define how effective the predictions for the user are in terms of distinction between user relevant and irrelevant items. These metrics are used if it is more interesting to evaluate whether the user had a positive interaction or not, rather than how accurate the prediction is to the exact prediction (Cremonesi et al., 2008). Each recommendation can be classified as either a *true positive (TP)*, a *true negative (TN)*, a *false positive (FP)* or a *false negative (FN)*.

According to Bondarenko (2019), **Precision and Recall** refer to fractions, where precision is the fraction of relevant instances among the retrieved instances, while recall is the fraction of relevant instances that have been retrieved over the total amount of relevant instances

$$Precision = \frac{TP}{TP + FP} \tag{3.8}$$

$$Recall = \frac{TP}{TP + FN}. \tag{3.9}$$

Concerning top-N-recommendations recommenders, the data is divided into a training set and a test set. The algorithm is initially applied to the training data and produces a list of suggested items. A *hit set* refers to the fraction of the training set, which only contains the suggested top-N items that are also contained in the test set (Baesens & vanden Broucke, 2016).
Cremonesi et al. (2008) stated in their work, that the number of items rated by each user is much smaller than the items available in the entire dataset and that the number of relevant items in the test set may be much smaller than that one in the whole dataset, meaning that precision and recall are mainly used to compare different recommendation algorithms on the same dataset.

The **F-Measure** combines precision and recall to bypass the problem of the recall increasing while the precision is decreasing for increasing N

$$F1 = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}. \tag{3.10}$$

3.4 Methods for creating Recommender Systems

There are plenty of methodologies that recommender systems can be built upon, including methodologies using machine learning algorithms or statistical solutions. The following sections aim to describe three methods in more detail: section 3.4.1 discusses the Matrix Completion problem and algorithms for Matrix Factorization, section 3.4.2 contains a description of Deep Neural Network based recommendation algorithms and section 3.4.3 focuses on Bayesian statistics.

3.4.1 Matrix Factorization and Matrix Completion

The Matrix Completion Problem refers to a task that aims to fill in missing rating records in the user-item interactions matrix. Each entry (i, j) in the user-item interactions matrix represents the rating of item j given by user i. The entries are empty, if the user has not yet interacted with the particular item. The goal is to predict the missing ratings and fill the empty entries in the matrix, assuming it is low-rank, in order to avoid the sparsity issue of recommender systems (Ramlatchan et al., 2018).
Since collaborative filtering approaches usually struggle with the sparsity and scalability issues, *Matrix Factorization* describes a more advanced method, developed to decomposes the original sparse matrix into low-dimensional matrices with latent factors and reduced sparsity and solve the matrix completion problem. Matrix Factorization is the most successful realization of latent factor models and is one of the most common ways of implementing model-based collaborative filtering based algorithms (Pan & Xia,

2015; Ramlatchan et al., 2018). Thereby the user-item interaction matrix is decomposed into the product of two smaller dimensionality rectangular matrices resulting in a representation of users and items in a lower dimensional latent space (Koren et al., 2009).
In linear algebra, matrix factorization describes a mathematical operation, which decomposes a matrix into a product of matrices. In case of recommender systems, the user-item interactions matrix is to be subdivided into two matrices, one containing the users as rows and the latent factors as columns and the other containing the items as columns and the latent factors as rows as seen in Figure 3.2 (Liao, 2018). Latent factors refer to non-observable variables, whose presence can be identified through their impact on observable variables.

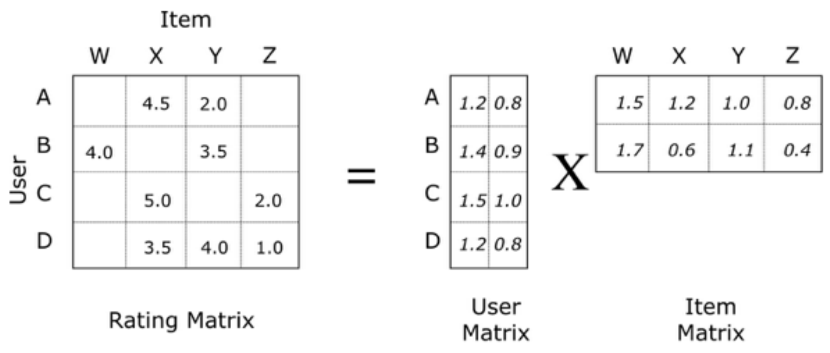


Figure 3.2: Example of a decomposed rating matrix (Liao, 2018).

The goal is to discover these latent factors for both users and items to create a low-rank matrix approximation of the actual interactions matrix. N. K. Kumar and Shneider (2016) describe the low rank matrix approximation as approximating a matrix by one whose rank is less than that of the original matrix. The purpose of such approximations is to obtain a more compact representation of the matrix while minimizing the loss of information. This avoids the two main problems, sparsity and scalability, of collaborative filtering approaches.
In this context it means, the higher the correspondance between user and item latent factors, the more likely a recommendation will be issued, leading to a more accurate estimate of the interactions matrix (Koren et al., 2009).
Due to the fact that matrix factorization aims to approximate the actual ratings matrix R with the user matrix U and item matrix V, the following cost function needs to be minimized

$$J = ||R - U \times V^T||_2 + \lambda(||U||_2 + ||V||_2). \tag{3.11}$$

As already mentioned in section 3.3, the first term describes the *mean squared error* between the actual ratings matrix R and the approximated matrix, whereas the second term is the *weighted-λ-regularization*, which is required for a better generalization of the results, meanintg to prevent overfitting (Karat, 2015).

Alternating Least Squares (ALS) is one of the most popular algorithms for matrix factorization recommenders. As mentioned in section 3.1.1, Apache Spark ML implemented the ALS algorithm for matrix factorization based collaborative filtering for implicit feedback. It is a two-step procedure of iterative optimization, in which in each iteration, it first fixes the item matrix V and then solves for the user matrix U, then fixes U and then solves for V. Alternating between these two steps causes the cost function to decrease until it reaches the level of convergence and stays unchanged. It is advantageous in systems that use parallelization, due to the independence of each iteration computation of user, respectively item factors from other iterations (Karat, 2015; Koren et al., 2009).

3.4.2 Deep Neural Networks

Deep Neural Networks have gained great importance and success in recent years, especially for speech recognition, computer vision and natural language processing.The influence of deep learning is equally widespread with new research confirming its usefulness when used for information retrieval and recommender systems. The subject of deep learning in recommender systems is evidently thriving (Zhang, Yao, Sun, & Tay, 2017).
Deep Neural Networks are artificial neural networks consisting mainly of layers, neurons, weights and activation functions. The data flows in forward direction from the input layers and through the hidden layers until it reaches the last output layer.
Training and inference are the two main phases in which deep learning recommendations are produced. In the training phase, the model is trained to predict user-item interactions probabilities by passed instances of prior interactions, which is followed by inferring the likelihood of new interactions (McDonald, 2021).

Zhang et al. (2017) stated in their paper that deep learning based recommender systems can be divided into two major categories: recommendations with neural building blocks and recommendations with deep hybrid models.
When performing recommendations with neural building blocks, models are grouped into eight subcategories, in accordance with the eight deep learning models that are commonly used: Multilayer Perceprton (MLP), Autoencoder (AE), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Restricted Boltzmann Machine (RBM), Neural Autoregressive Distribution Estimation (NADE), Adversarial

Networks (AN), Attentional Models (AM), and Deep Reinforcement Learning (DRL) based recommender systems. The applicability of a recommendation model is determined by the deep learning technique used. For instance, RNNs are used for text classification prediction problems, while CNNs are used for classification prediction problems on image data.
Recommendations with deep hybrid models means utilizing multiple deep learning techniques in order to produce accurate recommendations. Due to the flexibility of deep neural networks it is possible to combine numerous complementing neural building blocks, which results in a more powerful and accurate hybrid model.

Neural Collaborative Filtering (NCF) is a neural network based framework that can generalize matrix factorization and learns user-item interactions through a *Multi-Layer Perceprton (MLP)*. Regardless that matrix factorization is a useful choice for collaborative filtering, the simple computation of the latent features, the inner product of the user and item matrix, lowers its' performance. This can be improved by incorporating user-item bias terms into the interactions function (He et al., 2017; A. Sharma, 2019).
NCF aims to recommend a ranked list of items to the user. The user either interacted with the item, not directly meaning it was a positive interaction, or not, not meaning the user disliked the item.
According to He et al. (2017), the input layer takes the two feature vectors that describe the user and the item as the input. The embedding layer projects the sparse representation of each vector to a dense vector. In order to convert the obtained user/item embedding, that can be viewed as the latent vector for the user/item, to prediction scores, the embeddings are fed into a multi-layer neural architecture. Each of these layers can be tweaked to uncover certain latent structures of user–item interactions. Figure 3.3 shows the architecture of the framework.

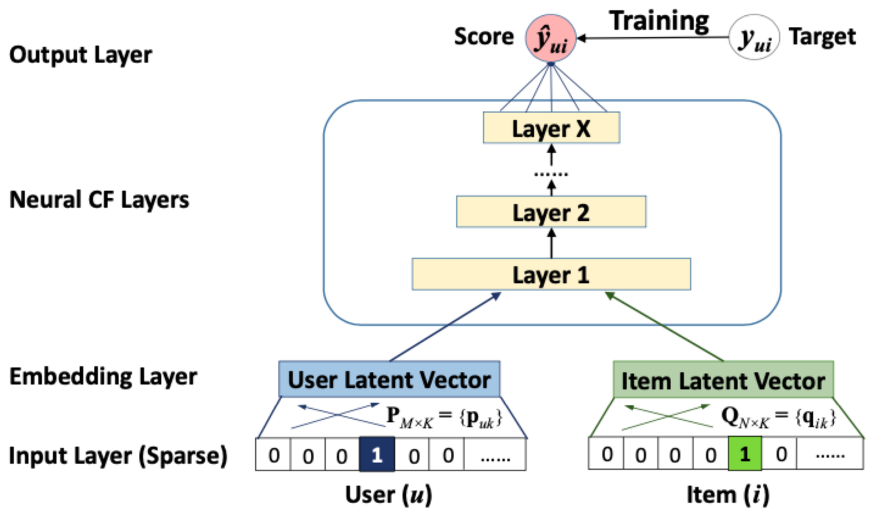


Figure 3.3: Architecture of the Neural Collaborative Filtering framework (He et al., 2017).

3.4.3 Bayesian Statistics

Approaches derived from *Bayesian Statistics*, which are based on the *Baye's Theorem* for probability computing, have also become popular for recommendation prediction.
Bayesian Personalized Ranking (BPR) is the most common Bayesian approach for recommender systems. Personalized ranking in general refers to the ranking of items in a ranked item list based on the user's implicit behavior over past items. The core approach is to predict a personalized score for an item that reflects the user's preference for it and afterwards the items are sorted and ranked based on their score (Rendle, Freudenthaler, Gantner, & Schmidt-Thieme, 2009). However, in BPR, rather than predicting an exact rating for a particular item, it uses item pairs and optimizes for a correct ranking of the pairs.

Rendle et al. (2009) proposed BPR in two main steps in their work, consisting of a BPR optimization criterion (BPR-Opt), following the LearnBPR algorithm. The BPR-Opt is derived using the likelihood function $p(i >_u j|\Theta)$ and the prior probability $p(\Theta)$, Θ being the parameter vector of the model class that determines the ranking. The probability function describes the individual probability that user u prefers the item i over item j, while $>_u$ meeting the properties *totality*, *antisymmetry* and *transitivity* of a total order

$$p(i >_u j|\Theta) = \sigma(\hat{r}_{uij}(\Theta)). \tag{3.12}$$

Here, $\sigma(x)$ refers to the logistic sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-}x} \tag{3.13}$$

and $\sigma(\hat{r}_{uij}(\Theta)$ captures the relationship between u, i and j, which is then delegated to the underlying model class (e.g. matrix factorization or k-nearest-neighbors) (Rendle et al., 2009). The ranking task is completed with the following optimization computation

$$BPR - Opt = \ln p(\Theta| >_u) = \sum_{(u,i,j) \in D_s} \ln \sigma(\hat{r}_{uij}) - \lambda_\Theta ||\Theta||_2. \tag{3.14}$$

$p(\Theta)$ is a density of a normal distribution, λ_Θ is a model specific regularization factor and $D_S := \{(u,i,j)|i \in I_u^+ \wedge j \in I \backslash I_u^+\}$ is the training data.

The second step in the proposed BPR procedure is the learning of a BPR algorithm, that is a stochastic gradient-descent algorithm aiming to optimize the performance of the model (Chen, 2020).

Beside BPR it is also possible to create a recommender system using *Bayesian Networks*, a method used in model-based collaborative filtering techniques. A Bayesian network is an acyclic directed graph containing nodes that depict random variables, which are connected through arrows as edges. Each node N_i has a conditional probability distribution that is dependent on the parent node. When realizing this approach in terms of recommendations, most of the systems are based on the hybrid approach. Since Bayesian network models focus on the relationships between users, items and features, the following random variables represent the nodes of the graph (de Campos et al., 2010):

- **User nodes** U_i represent the probability of a user rating an item with value s from the ratings domain R. If s equals 0, it means that there cannot be any useful information about the user's preference inferred.
- **Item nodes** I_j can accept values from the domain $\{i_{j,0}, i_{j,1}\}$ that denote the relevance of the item.
- For each feature that describes an item, there is a **Feature node** F_k, which can similarly to the item nodes denote, whether a feature applies to the given item or not.

The content-based component of this model introduces new nodes that represent the active user's preference on the item, based on the user relevant features of the item. The collaborative component combines the preferences that were inferred from similar user nodes into a single node. Both nodes are then combined to the final hybrid component, which obtains the final prediction (de Campos et al., 2010). Figure 3.4 depicts the structure of the Bayesian network.

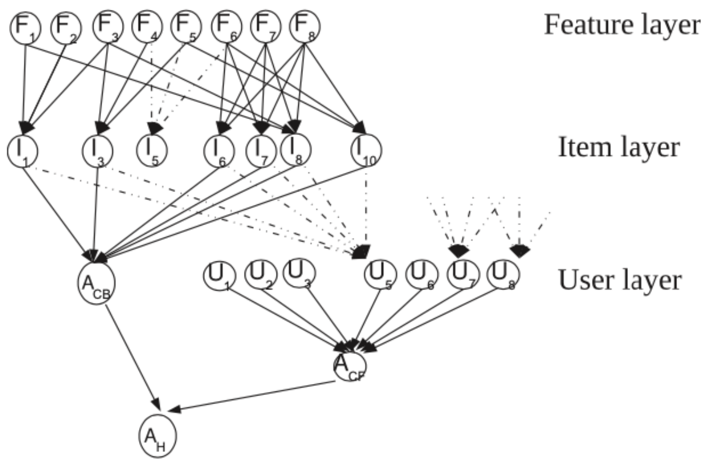

Feature layer
Item layer
User layer
F_1 F_2 F_3 F_4 F_5 F_6 F_7 F_8
I_1 I_3 I_5 I_6 I_7 I_8 I_{10}
A_{CB}
U_1 U_2 U_3 U_5 U_6 U_7 U_8
A_{CF}
A_H

Figure 3.4: Structure of a Bayesian Network (de Campos, Fernández-Luna, Huete, & Rueda-Morales, 2010).

3.5 Cold Start Problem and Solution Methodologies

So far, it is clear that recommender systems are an enrichment for the digital world, but they still exhibit shortcomings, as discussed in section 2.1. This section will specifically focus on the cold-start problem that collaborative filtering based recommender systems face and whether the above mentioned methodologies offer a solution to the cold-start problem.

The cold-start problem is a shortcoming of collaborative filtering based recommender systems and occurs when there is a lack of item information or preference information about a certain user and thus, no interaction with the system given. In the *complete cold-start problem* there are no rating records available at all whereas in the *incomplete cold-start problem* only a few rating records of users and items are available. There are three different cases that can be distinguished in a cold-start scenario (Bobadilla, Ortega, Hernando, & Bernal, 2012; Volkovs, Yu, & Poutanen, 2017):

- User cold-start: When a new user is introduced to the system, he/she usually does not have any interaction yet, which makes it difficult to recommend anything to him/her.
- Item cold-start: By adding a new item to the system, it might provide content information, however, it does not provide any interactions with the system yet.
- New community: This case refers to the initial start-up of a recommender system, where a catalogue of items already exists but no users are present. Hence, the lack of interaction leads to this cold-start case.

DropoutNet is a neural network based latent model that aims to solve the cold start problem by focusing on dropout during training. In deep learning, dropout refers to a regularization method, which includes dropping out neurons in a neural net. The idea behind DropoutNet is that by randomly dropping item ratings during training, the network can be made resistant against the item cold-start. The essential feature is that instead of dropping the nodes as usually when using dropout during training, the item features are randomly dropped, leading to a less dependant neural network on specific ratings (Oshiba, 2018; Volkovs et al., 2017).

Wei, He, Chen, Zhou, and Tang (2016) proposed a solution approach for the cold start problem that adresses the complete and incomplete cold start problem by introducing two recommendation models. The framework that successfully improves the recommendation performance in the item cold start cases combines a collaborative filtering model and a deep neural network. The collaborative filtering model predicts unknown ratings whereas the deep neural network extracts content features of items.

Wang and Chen (2015) introduced a distributed hybrid framework, that adresses the user cold start problem, that is based on user classification. First, each user is classified into a type, taking user information into account. Depending on the user type and the rating records, tasks are forwarded to the recommendation modules, which will create a recommendation output seperately. Lastly, the obtained recommendation results are combined into one final result. The cold start problem can be avoided due to the fact that the framework dynamically assigns suitable recommendation algorithms depending on the respective user type.
In addition to the above examples of existing cold start solution approaches, there is a variety of deep neural network models that provide a successful method for solving the cold start problem.

Even though matrix factorization is utilized to solve large-scale matrix problems, itself it does not offer a solution for handling new users, that do not exhibit preferences or rating records (Yi et al., 2020), and hence may not be the best recommendation approach to solve the cold start problem. This is due to the fact that matrix factorization is a method for implementing collaborative filtering, which requires at least some rating records to infer similar users and items. When introducing new users or new items to the system, this approach is not suitable (Oshiba, 2018).
Yeung, Yang, and Ndzi (2012) presented a Bayesian network based algorithm to eliminate the cold start problem in their framework. They stated that when a new user enters the system, it will use group profile data to create a global Bayesian network that can solve the data sparsity problem.

3.6 Study Object: ArXiv Dataset

The study object for this thesis will be the arXiv dataset (Cornell University, n.d.), that was downloaded from Kaggle, an online community of data scientists, which allows users to find, publish and use datasets for machine learning purposes. ArXiv is a digital open-access research sharing platform, containing a collection of scholary articles, research papers and several other academic e-prints. It offers registered researchers to submit, retrieve, search and discover articles in their area of interest. The contents of these e-prints include research from the fields of computer science, mathematics, physics, statistics, financial mathematics and biology.
After registering to ArXiv, the user may select one of the above mentioned research fields and can afterwards search for any keywords or titles, that fall under the selected field category. ArXiv will then list all e-prints that that contain the search term.
The arXiv dataset provides metadata as a collection of over 1.7 million JSON entries, each item consisting of necessary paper metadata as seen in Table 3.1 (Cornell University, n.d.).

What stands out is that the dataset does not contain any user related data like ratings, preferences or personal user information. This represents a problem for any recommender system, since it is not possible to recommend papers to users, if there is no user preference information available.

Table 3.1: arXiv dataset metadata

id	ArXiv id of the respective paper
submitter	the name of the submitter of the paper
authors	the authors of the paper
title	the title of the paper
comments	additional information about the paper
journal-ref	information about the journal the paper was published in
doi	Digital Object Identifier
abstract	the abstract of the paper
categories	the category the paper falls under
version	the version history of the paper
update_date	the date the paper was updated last
authors_parsed	the parsed author names

3.7 Application of recommendation methods to the study object

In the following it will be discussed whether the in section 3.5 mentioned cold-start recommendation approaches are suitable to solve the cold-start problem for the arXiv dataset.
Since every in section 3.4 presented recommendation method requires some kind of user record data or at least personal user information, they are not applicable to the arXiv dataset. As the cold-start problem is a shortcoming of collaborative filtering based recommenders, the arXiv dataset needs to be approached in a different way.
In such cases, a possible attempt is to use natural language processing techniques and knowledge graphs in order to produce accurate recommendations. For this problem statement it is crucial to look at the structure of the dataset. The dataset exhibits a lot of item content features like the author of a paper, the abstract or the title of the paper, which would be most interesting for a person reading the according paper. If a reader is interested in a certain research paper, they might be interested in either other works that the author published, or papers that have a similar title to the reference paper. Abstracts contain keywords that might be highly relevant to the reader. These may also appear in other papers the user might show interest in. The above properties may be independent of user information, but it is possible to calculate similarities between the research papers. If a user had a positive interaction with a certain paper, they might either want to read more papers the author published or they would want to read papers, that either continue the subject or that are similar to the reference paper.

3.7.1 Implementation

To solve the cold-start problem for the arXiv dataset, the following practical part of this work presents two experimental approaches, in which the top N paper recommendations can be produced based on either authors or similar titles. Table 3.2 shows the general setup for both experiments.

Table 3.2: Code Setup

Programming Language	Python 3.9
Environment	Jupyter Notebook
Graph Database	Neo4j v. 4.4.4

Author Experiment: This experiment aims to produce research paper recommendations based on authors. As users might be interested in an author's further works, they might want to receive recommendations about other papers the author published, independent from the topic that is discussed in the respective papers. The core implementation of the author experiment is structured as follows, and will be explained in more detail in the text below:

1. First, the dataset is cleaned so that only the relevant parts are included in the dataframe. The relevant parts include the id and the title of the paper and the author names.
2. The dataset is then transformed into a knowledge graph that represents the relationships between author and title nodes: an author node is connected to a title node if the author authored the respective paper
3. Through specifying the importance measure, the query outputs the recommended papers.

Sullivan (2021) published a tutorial on how to create a graph database using the above mentioned Neo4j framework, which will be the base for the implementation of the author experiment. As seen in section 3.6, the dataset contains more than ten properties that can be used to infer recommendations from. For the author experiment, only the *authors_parsed*, the *id* and the *title* are relevant.
Figure 3.5 shows how the relationship between an author and the papers he/she wrote is visualized in Neo4j.

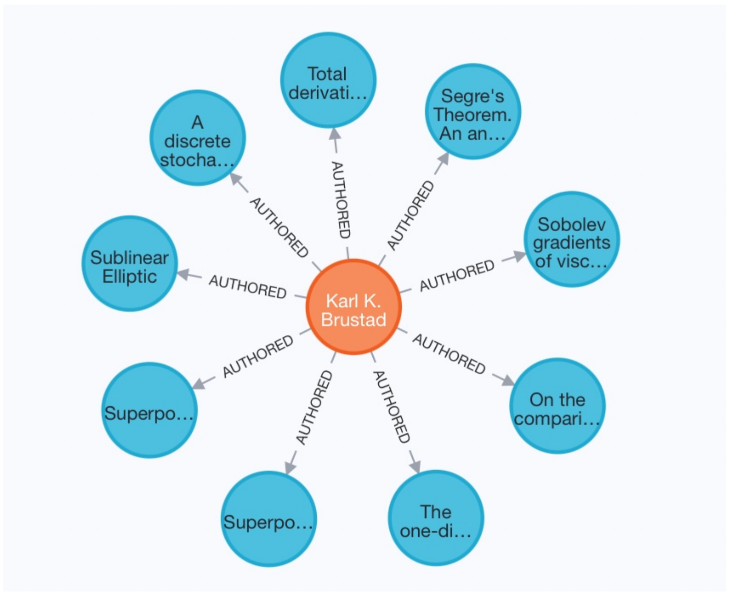
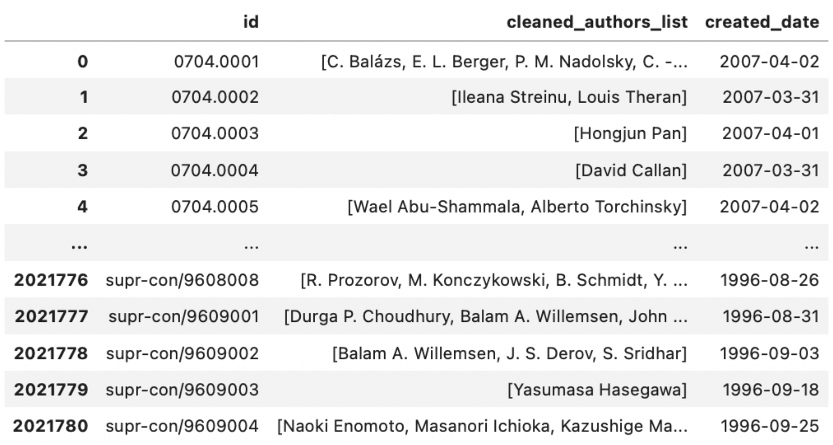


Figure 3.5: Relationship between an author and papers

.

For further processing, it is convenient to clean the data. In this case, the *authors_parsed* column is an array of arrays, each containing the name of one of the collaborating authors of the respective paper. To get a more clear representation, the authors of each paper are all added into one single array.
In addition to that, it might be useful for the user to know the release date of a paper, which is stored in the version property for each paper of the dataset. Here, the date is transformed to the YYYY-MM-DD format and stored in a new column. Both the cleaned author lists and the new date properties can be observed in the resulting dataframe depicted in Figure 3.6.
After creating a normalized representation of the data, the next step is to create the knowledge graph. For this, the graph-database Neo4j, that was introduced in section 3.2, will be used. Neo4j provides so called *Sandboxes*, that enable access to a Neo4j database online, without installing anything locally. In order to populate the database with the arXiv data that was cleaned in the first step, the implementation uses a Blank Sandbox. The connection between the notebook and the Sandbox database is established via Python, using the Bolt URL, the password and the username, that are noted in the Blank Sandbox connection details. Here, it is important to note, that Blank Sandbox instances expire after ten days. After expiry, a new Sandbox instance can be created with different connection details that afterwards need to be adjusted in the notebook.



	id	cleaned_authors_list	created_date
0	0704.0001	[C. Balázs, E. L. Berger, P. M. Nadolsky, C. -...	2007-04-02
1	0704.0002	[Ileana Streinu, Louis Theran]	2007-03-31
2	0704.0003	[Hongjun Pan]	2007-04-01
3	0704.0004	[David Callan]	2007-03-31
4	0704.0005	[Wael Abu-Shammala, Alberto Torchinsky]	2007-04-02
...	...	...	...
2021776	supr-con/9608008	[R. Prozorov, M. Konczykowski, B. Schmidt, Y. ...	1996-08-26
2021777	supr-con/9609001	[Durga P. Choudhury, Balam A. Willemsen, John ...	1996-08-31
2021778	supr-con/9609002	[Balam A. Willemsen, J. S. Derov, S. Sridhar]	1996-09-03
2021779	supr-con/9609003	[Yasumasa Hasegawa]	1996-09-18
2021780	supr-con/9609004	[Naoki Enomoto, Masanori Ichioka, Kazushige Ma...	1996-09-25

Figure 3.6: Dataframe containing the cleaned author lists and dates

.

To avoid duplicates, it is crucial to create constraint queries for the nodes by asserting the author name and the paper id to be unique.
The next step is to populate the database by creating the author and paper nodes and connecting them using Cypher. Each node has properties that can be accessed in queries. In this case, each paper node has the paper id, the creation date and the title as properties, whereas author nodes have the author name and the author id as properties. The following code example shows how the paper and author nodes are connected and how paper nodes are created:

```
def add_papers(rows, batch_size=50):

        query = '''
        UNWIND $rows as row
        MERGE (p:Paper {id:row.id}) ON CREATE SET p.title = row.title,
        p.created = row.created_date

        WITH distinct row, p
        UNWIND row.cleaned_authors_list AS author
        MATCH (a:Author {name: author})
        MERGE (a)-[:AUTHORED]->(p)
        RETURN count(distinct p) as total
        '''

        return insert_data(query, rows, batch_size)
```

The nodes are connected via the **[:AUTHORED]** relationship, indicating that an author authored a particular paper. The actual recommendation step is done by querying, which papers might be interesting for a user that is interested in a particular author. This will be visualized and discussed in more detail in the next section.

Title Experiment: The title experiment aims to recommend papers, whose titles are decisive for the recommendation process. Unlike in the author experiment, the authors of these papers do not matter. Here, all papers, that have similar titles to the title of the paper the user is currently reading, should be recommended.
Similar to the author experiment, the data needs to be cleaned first. Some of the titles in the dataset are saved in Latex format, which can cause difficulties when computing title similarities. This can be solved by converting the titles into UTF-8 format using Python libraries like **pylatexenc**.
The next step is to calculate the embedding for each title. For this task, there are plenty of useful NLP tools like the context-free model *word2vec*, *sentence2vec* or *BERT* models. The simplest approach for obtaining the vector representation of a sentence is to take the average of the word representations of each word in the sentence using word2vec, e.g. a title has 10 words and each word is represented by a vector of size 5. A problem that could arise when using this approach is information loss. As Karimi (2021) stated, for sentences containing commonly used terms that do not usually indicate sentiments, averaging over the words can result in similar representations.
Sentence2vec avoids this issue by taking the averaged vector sum to get the vector representation of the sentence. The similarity between two sentences is then calculated using the cosine similarity of their vectors, which was adressed in section 3.2.

Bratanic (2020) used SciBERT (Beltagy, Cohan, & Lo, 2019) to obtain embeddings for each title in the dataset. It is a BERT model, that was trained on scientific text data, making it a useful tool to use for the arXiv dataset. The pretrained model can be used by importing the **transformers** library into the notebook.
The embeddings can be then stored in the Neo4j database as a node property for the respective paper node. Considering this, it makes sense to use the Neo4j integrated similarity algorithms. Neo4j provides algorithms for computing the Cosine similarity of vectors as well as algortihms for Jaccard and Pearson similarity computing. An example usage can be seen in the following Cypher statement:

```
RETURN gds.alpha.similarity.jaccard([1,2,3], [1,2,4,5]) AS similarity
```

As the goal is to recommend the top N papers to the user, that are most similar to the currently interesting paper, Neo4j provides the *K-Nearest-Neighbors* algorithm, which generates new relations between each node and its k nearest neighbors by computing a distance value for all node pairs in the graph. The distance is determined using the properties of the nodes. In this case, the properties are the title embeddings of each title node, that were computed by SciBERT previously. To project the in-memory graph, the following query needs to be executed (Bratanic, 2020):

```
CALL gds.graph.create('paper_similarity','Paper','*',
              {nodeProperties:['embeddings']})
```

Assume the user wants to receive the top five recommendations that are the most similar to a particular paper, that "Y. Wang" wrote. The recommendation is then executed in the Neo4j Sandbox by the following query:

```
MATCH (p:Paper {id:"0804.1862"})-[s:SIMILAR TO]->(q)
RETURN p.title as paper, q.title as recommendation, q.score as score
ORDER BY score DESC
LIMIT 5
```

3.7.2 Visualization and Results

In the following the results of the author experiment are being visualized with Neo4j's Sandbox.
After connecting to the Neo4j Sandbox, the knowledge graph can be displayed with the following query:

```
MATCH (a:Author)-[:AUTHORED]->(p:Paper) RETURN a, p
```

The query returns the author and paper nodes and their relationships. An author node is connected to a paper node, if the author wrote the respective paper. As seen in Figure 3.7, the paper nodes are displayed in blue whereas the author nodes are displayed in orange. An author can be connected to multiple papers indicating that he/she was a co-author in many papers.
The following scenario describes how a possible recommendation is queried: assume a user is interesed in "Y. Wang"'s work and wants to receive recommendations about this authors other papers. It would be simple to output all papers this author collaborated in, but since the number of returned papers can be too large, it is necessary to narrow down the recommendation. A possible attempt would be to recommend the top five papers that ""Y. Wang collaborated in with the highest count of authors in a descending order:

```
MATCH (a:Author{name:"Y. Wang"})-[:AUTHORED]->(p:Paper)
RETURN size(()-[:AUTHORED]->(p)) as num_authors, a, p
ORDER BY num_authors DESC
LIMIT 5
```

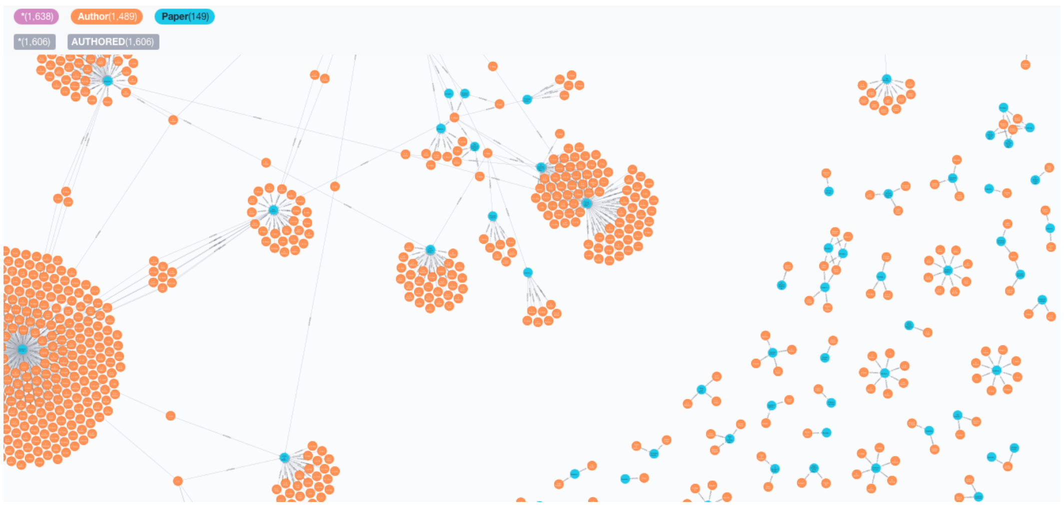


Figure 3.7: Section of the knowledge graph for the arXiv dataset
.

The resulting graph in Figure 3.8 consists of the top five most co-authored papers for the above mentioned constraints.

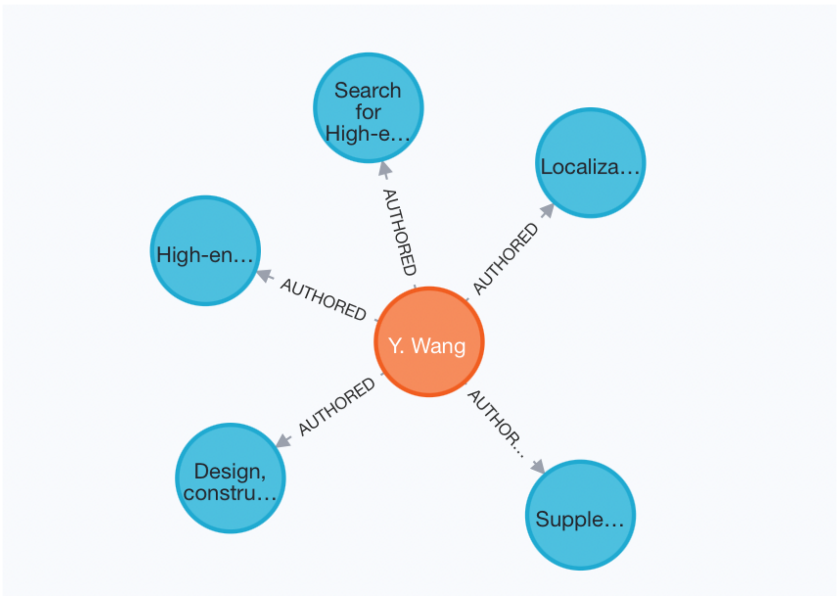


Figure 3.8: Recommendation of the top five papers with the highest count of authors, that "Y. Wang" collaborated in

Another scenario that might be interesting is to recommend the top five papers that "Y. Wang" wrote since January 1st 2021, that have the most co-authors. The resulting recommendation can be obtained by the following query:

```
MATCH (a:Author{name:"Y. Wang"})-[:AUTHORED]->(p:Paper)
WHERE p.created > "2021-01-01"
WITH p.created as date, p.title as title, size(()-[:AUTHORED]->(p)) as num_authors
ORDER BY num_authors DESC
RETURN date, title, num_authors
LIMIT 5
```

4 Conclusions

The main objective of this thesis was to outline the general idea behind recommender systems, discuss several current implementations, and refer to current applications in practice. At first, shortcomings that can occur in recommender systems and already known recommender systems that are used in practice were presented. This was followed by a detailed description of the main forms in which recommender systems can occur namely collaborative filtering, content-based and hyrid recommender systems.
Out of the many algorithms and methodologies that recommender systems can be based upon, this thesis introduced Matrix Factorization, Deep Neural Networks and Bayesian statistics, which were evaluated on whether they represent an effective solution to the cold-start problem. Besides introducing knowledge graphs, as they represent a useful tool for managing large datasets and help represent relationships between entities, the thesis presented common similarity measures and evaluaton metrics for recommender systems.
The case study focused on solving the cold-start problem of the arXiv dataset by splitting the problem into two experiments, one focusing on author-based recommendations and the other one using paper titles to create recommendations. Both experiments were based on upon knowledge graphs using Neo4j.

The author experiment was conducted in accordance with the procedures described by Sullivan (2021) and Bratanic (2020). The resulted recommender system approach shows that scenarios like the presented cold-start problem for the arXiv dataset can be approached by using graph databases and querying the desired recommendation. Note that in this experiment the recommendations can only be provided by specifying the exact author names and the given constraints such as the number of co-authors, which means that the preferences are not stored or learned. In this case, only the user who has installed this approach locally can call his personal preferences, meaning that the success of the recommendation is subjective.
In comparison to that the title experiment can be classified as a content-based approach since it takes advantage of the k-nearest-neighbors algorithm provided by Neo4j to compute similar papers and recommend those that are most similar to the queried paper.

How good these approaches perform cannot be measured using the evaluation metrics presented in section 3.3 since the accuracy of the recommendations is measured based on predicted ratings and actual user ratings, which are not present in the arXiv dataset. This means that user satisfaction can only be measured qualitatively, i.e. whether the recommended paper is useful for the user or not.
Along with the results that Fleischman and Hovy (2003) and Bai et al. (2019) pesented in their papers, this thesis demonstrates that recommendations without user preferences can be generated by content-based approaches using NLP tools.

An addition for the title experiment would be the inclusion of the paper abstracts for the calculation of similar papers in order to obtain more precise recommendations. For the purpose of creating a collaborative-filtering based recommendation engine similar to the examples that Neo4j (Neo4j, n.d.) provided it would be necessary to include user records into the arXiv dataset that contain paper ratings given by each user. A possible extension to the presented approach is the implementation of user bots that simulate user behavior by rating papers and afterwards storing the ratings in a dataset along with the papers from the arXiv dataset, resulting in a user-item interactions matrix. Based on that, the user can receive paper recommendations according to the calculation of similar users.

Bibliography

Apache Software Foundation. (2014). Spark. https://spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html. 2.2.0.

Asanov, D. A. (2011). Algorithms and methods in recommender systems.

Baesens, B., & vanden Broucke, S. (2016). What are popular ways of evaluating recommender systems? https://www.dataminingapps.com/2016/04/what-are-popular-ways-of-evaluating-recommender-systems/.

Bai, X., Wang, M., Lee, I., Yang, Z., Kong, X., & Xia, F. (2019). Scientific paper recommendation: A survey. *IEEE Access*, *7*, 9324–9339. doi:10.1109/ACCESS.2018.2890388

Beltagy, I., Cohan, A., & Lo, K. (2019). Scibert: Pretrained contextualized embeddings for scientific text. *CoRR*, *abs/1903.10676*. https://github.com/allenai/scibert. arXiv: 1903.10676

Benard Magara, M., Ojo, S. O., & Zuva, T. (2018). A comparative analysis of text similarity measures and algorithms in research paper recommender systems. In *2018 conference on information communications technology and society (ictas)* (pp. 1–5). doi:10.1109/ICTAS.2018.8368766

Blumenfeld, Z. (2021). Exploring practical recommendation systems in neo4j. https://towardsdatascience.com/exploring-practical-recommendation-engines-in-neo4j-ff09fe767782.

Bobadilla, J., Ortega, F., Hernando, A., & Bernal, J. (2012). A collaborative filtering approach to mitigate the new user cold start problem. *Knowledge-Based Systems*, *26*, 225–238. doi:https://doi.org/10.1016/j.knosys.2011.07.021

Bondarenko, K. (2019). Precision and recall in recommender systems. and some metrics stuff. https://bond-kirill-alexandrovich.medium.com/precision-and-recall-in-recommender-systems-and-some-metrics-stuff-ca2ad385c5f8.

Bratanic, T. (2020). Network analysis of arxiv dataset to create a search and recommendation engine. https://medium.com/swlh/network-analysis-of-arxiv-dataset-to-create-a-search-and-recommendation-engine-of-articles-cd18b36a185e.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, *12*. doi:10.1023/A:1021240730564

Chen, D. (2020). Recommender system — bayesian personalized ranking from implicit feedback. https://towardsdatascience.com/recommender-system-bayesian-personalized-ranking-from-implicit-feedback-78684bfcddf6.

Chiang, J. (2021). 7 types of hybrid recommendation system. https://medium.com/analytics-vidhya/7-types-of-hybrid-recommendation-system-3e4f78266ad8.

Cornell University. (n.d.). Arxiv dataset. https://www.kaggle.com/Cornell-University/arxiv.

Cremonesi, P., Turrin, R., Lentini, E., & Matteucci, M. (2008). An Evaluation Methodology for Collaborative Recommender Systems. In *2008 international conference on automated solutions for cross media content and multi-channel distribution* (pp. 224–231). doi:10.1109/AXMEDIS.2008.13

de Campos, L. M., Fernández-Luna, J. M., Huete, J. F., & Rueda-Morales, M. A. (2010). Combining content-based and collaborative recommendations: A hybrid approach based on bayesian networks. *International Journal of Approximate Reasoning*, *51*(7), 785–799. doi:https://doi.org/10.1016/j.ijar.2010.04.001

Fleischman, M., & Hovy, E. H. (2003). Recommendations without user preferences: A natural language processing approach. In *Iui '03*.

Gomaa, W., & Fahmy, A. (2013). A survey of text similarity approaches. *international journal of Computer Applications*, *68*. doi:10.5120/11638-7118

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web* (pp. 173–182). doi:10.1145/3038912.3052569

Honnibal, M. (2015). Spacy. https://spacy.io.

Karat, S. (2015). How do you build a "people who bought this also bought that"-style recommendation engine. https://datasciencemadesimpler.wordpress.com/2015/12/16/understanding-collaborative-filtering-approach-to-recommendations/.

Karimi, A. (2021). How to get vector for a sentence from word2vec of tokens.

Kishore, R. (2019). A knowledge graph understanding and implementation tutorial for beginners. https://medium.com/analytics-vidhya/a-knowledge-graph-implementation-tutorial-for-beginners-3c53e8802377.

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, *42*(8), 30–37. doi:10.1109/MC.2009.263

Kumar, B., & Sharma, N. (2016). Approaches, issues and challenges in recommender systems: A systematic review. *Indian Journal of Science and Technology*, *9*. doi:10.17485/ijst/2015/v8i1/94892

Kumar, N. K., & Shneider, J. (2016). Literature survey on low rank approximation of matrices. *ArXiv*, *abs/1606.06511*.

Liao, K. (2018). Prototyping a recommender system step by step part 2: Alternating least square (als) matrix factorization in collaborative filtering. https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1.

Lops, P., de Gemmis, M., & Semeraro, G. (2011). Content-based Recommender Systems: State of the Art and Trends. (pp. 73–105). doi:10.1007/978-0-387-85820-3_3

Luo, S. (2018). Introduction to recommender system. https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26.

McDonald, C. (2021). How to build a deep learning powered recommender system, part 2. https://developer.nvidia.com/blog/how-to-build-a-winning-recommendation-system-part-2-deep-learning-for-recommender-systems/.

Neo4j. (n.d.). Tutorial: Build a cypher recommendation engine. https://neo4j.com/developer/cypher/guide-build-a-recommendation-engine/.

Neo4j. (2010). Neo4j. https://neo4j.com. current verion: 4.4.4.

Oshiba, K. (2018). Tackling the cold start problem in recommender systems. https://kojinoshiba.com/recsys-cold-start/.

Pan, B., & Xia, S.-T. (2015). Matrix-completion-based method for cold-start of distributed recommender systems. In S. Arik, T. Huang, W. K. Lai, & Q. Liu (Eds.), *Neural information processing* (pp. 592–599). Cham: Springer International Publishing.

Ramlatchan, A., Yang, M., LIU, Q., Li, M., Wang, J., & Li, Y. (2018). A survey of matrix completion methods for recommendation systems. *Big Data Min. Anal.*, *1*, 308–323.

Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence* (pp. 452–461). Montreal, Quebec, Canada: AUAI Press.

Ricci, F., Rokach, L., & Shapira, B. (2010). Recommender systems handbook. (Vol. 1-35, pp. 1–35). doi:10.1007/978-0-387-85820-3_1

Rocca, B. (2019). Introduction to recommender systems. https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada.

Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative filtering recommender systems. In P. Brusilovsky, A. Kobsa, & W. Nejdl (Eds.), *The adaptive web: Methods and strategies of web personalization* (pp. 291–324). doi:10.1007/978-3-540-72079-9_9

Sharma, A. (2019). Neural collaborative filtering. https://towardsdatascience.com/neural-collaborative-filtering-96cef1009401.

Sharma, M., & Mann, S. (2013). A survey of recommender systems: Approaches and limitations.

Shetty, B. (2019). An In-Depth guide to how recommender systems work.

Smith, B., & Linden, G. (2017). Two decades of recommender systems at amazon.com. *IEEE Internet Computing*, *21*(3), 12–18. doi:10.1109/MIC.2017.72

Sullivan, C. (2021). Create a graph database in neo4j using python. https://towardsdatascience.com/create-a-graph-database-in-neo4j-using-python-4172d40f89c4.

Tyagi, N. (2021). 6 Dynamic Challenges in Formulating the Recommendation System. https://www.analyticssteps.com/blogs/6-dynamic-challenges-formulating-imperative-recommendation-system.

Volkovs, M., Yu, G., & Poutanen, T. (2017). Dropoutnet: Addressing cold start in recommender systems. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30), Curran Associates, Inc.

Wang, J.-H., & Chen, Y.-H. (2015). A distributed hybrid recommendation framework to address the new-user cold-start problem. (pp. 1686–1691). doi:10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.307

Wei, J., He, J., Chen, K., Zhou, Y., & Tang, Z. (2016). Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications*, *69*. doi:10.1016/j.eswa.2016.09.040

Yeung, K., Yang, Y., & Ndzi, D. (2012). A proactive personalised mobile recommendation system using analytic hierarchy process and bayesian network. *Journal of Internet Services and Applications*, *3*. doi:10.1007/s13174-012-0061-3

Yi, J., Zhong, M., Chen, Y., & Jie, A. (2020). A hybrid collaborative filtering recommendation algorithm based on user attributes and matrix completion. *IOP Conference Series: Materials Science and Engineering*, *790*, 012058. doi:10.1088/1757-899X/790/1/012058

Yuan, J., Shalaby, W., Korayem, M., Lin, D., Aljadda, K., & Luo, J. (2016). Solving cold-start problem in large-scale recommendation engines: A deep learning approach. (pp. 1901–1910). doi:10.1109/BigData.2016.7840810

Zhang, S., Yao, L., Sun, A., & Tay, Y. (2017). Deep learning based recommender system: A survey and new perspectives. doi:10.1145/3285029