Towards a Universal Recommender System: A Linked Open Data Approach

Computing Science Masters Thesis

Hubert, G. (Gustav Hubert)

First supervisor
Dr. M.W. (Mel) Chekol

Second supervisor
Dr. Y. (Yannis) Velegrakis

Universiteit Utrecht

Department of Information and Computing Sciences
Utrecht University
Netherlands
October 2023

Abstract

Recommender systems (RSs) play a crucial role in helping users make informed decisions in the face of an ever-increasing array of choices. Most RSs are domain-specific, but domain-independent (or general purpose) RSs can be applied to a wide range of application areas, leveraging data and insights from different domains and handling large user populations. Current RS research is dominated by "improve the state-of-the-art", in terms of accuracy, speed and scale. Truly domain-independent recommender systems currently represent a gap in research, as is shown in this work.

This thesis explores the development of a general-purpose RS and the use of LOD to integrate data from different domains in an unsupervised way. Building a system that can effectively generate recommendations for any domain without prior knowledge of the data is challenging. Linked Open Data (LOD) offers a solution to this problem by enabling the integration of data from multiple domains. We explore and evaluate various unsupervised methods for acquiring and sorting through data, and using it to generate accurate recommendations. The resulting product is a truly domain-independent recommendation framework that can, in theory, be applied to a large variety of use-cases without requiring modification. Finally, this thesis suggests future research directions for building more effective general-purpose RSs.

Keywords— Recommender systems, Linked Open Data, general-purpose, domain-independent, inductive learning

Acknowledgements

I would like to thank my supervisors, Dr. M.W. (Mel) Chekol, for all the guidance in this unusual project, and Prof. dr. Yannis Velegrakis for his support. I am very grateful for having been able to work on a topic of my own choice, despite the associated challenges. Working on this thesis was very rewarding and I learned a lot.

I am also immensely grateful to my girlfriend, Yuxuan Hou, for helping me through the stressful periods and for the great feedback. Her unconditional support enabled this thesis, and her care played a pivotal role in my timely recovery from a cold just before my defense. Thank you!

I would also like to thank Prof. Tommaso Di Noia for taking time to answer some of my questions via email. His works greatly inspired me, and this thesis would not be possible without his publications.

Lastly, I want to express gratitude towards for my parents, without which none of this would have been possible in the first place, and for always believing in me.

Contents

Chapter 1

Introduction

The total volume of information on the internet is slated to reach 180 zettabytes in 2025[1]. The rapid growth of accessible data worldwide is giving users an ever-increasing array of choices. Users can now choose from a wide range of products and services available in different areas or domains, such as entertainment, e-commerce, social, services and many more.

Recommender systems are widely and successfully used in the industry to help users make sense of big data quantities. Recommendations are generated by analyzing user data, item attributes, and other contextual data to predict preferences. The objective of accurate recommendation is two-fold: improve user experience and, in most cases, sell more products, increase user engagement and improve customer satisfaction [41]. In this way, a recommender system that can target users accurately becomes a valuable asset in the competitive digital world [46].

For example, Spotify uses several independent machine learning models to generate item and user representations, which capture their unique characteristics. These representations are used in content-based and collaborative filtering systems to generate user-specific recommendations. Content-based filtering aims to describe the information by examining the content itself, whereas collaborative filtering aims to describe an item, i.e., a piece of music, with regard to its connection to other tracks on the platform by studying user interactions.[2]

Driven by industry needs[3], a lot of research has been done to improve the accuracy, scale and general performance of recommender systems. Since they primarily rely on data, the main bottleneck towards increasing the accuracy of recommendations is using higher quality data that is relevant to the application domain [105, 94]. For example, Netflix solely offers movies and TV series, meaning that its recommender system can be finely tailored towards recommending those and improved by providing it high quality data on each item or user. Conversely, most research on recommender systems focuses on a limited amount of domains as well.

In juxtaposition to common recommender systems, we define domain-independent (or general purpose or universal) recommender systems. They are not limited to specific domains, but rather can be applied to a very large range of application areas in an unsupervised way. This means that they can leverage data and insights from different domains and handle large user populations across any application area without requiring domain-specific configuration. Since most existing systems and related research either focus on a limited number of specific domains or require some form of supervision, we would like to investigate building a general-purpose recommender system that works in an unsupervised way. We realize that such a system is not needed for most applications, but it becomes necessary on a platform that deals with a large variety of data, such as user-defined content. An example use-case of this is given in Section 1.1.

[1]Statista 2022 - https://www.statista.com/statistics/871513/worldwide-data-created/
[2]Dmitry Pastukhov for music-tomorrow.com
[3]See: Netflix Prize

We believe that there is novelty in building a domain-independent recommender system, and that it is worth proving that such a system is possible. Our initial research has shown that domain-independent recommender systems do not currently seem to be an active area of research. We read through all major related surveys on recommender systems, Linked Open Data and graph learning, which are techniques that we have identified as potential ways for building such a system [120, 115, 112, 55, 105, 41]. [112] mentions self-evolutionary RS with dynamic-graph learning as an open research question. The problem that is addressed here relates to the evolving nature of users, items and how they interact, which cannot be captured by restricting the system to a static dataset. As for the other review papers, the concept is not currently mentioned or investigated.

For further proof, we searched and analyzed papers after conducting a keyword analysis. After an initial search based on the two main terms "domain-independent RS" and "general-purpose RS" we expanded to "domain-agnostic RS" and "multimedia RS". We used both Google Scholar[4] and ResearchGate[5]. Since this keyword analysis was performed as part of our literature review, the results can be found in Table A.1 of the Appendix. Basic information, such as title and authors, is included along with the approach used in the paper. Analyzing the top search results showed no immediately relevant papers. Most results fall under the domain of transfer learning techniques, (see Section 3.2.1). Though some methods used might be relevant to our research, this domain of research focuses on a limited amount of domains (usually two) and does not function completely unsupervised. The few remaining papers that contain the relevant keywords present general techniques (frameworks) that can be applied to any specific domain(s), but not in a completely unsupervised way, which does not match the definition given here.

1.1 Roco: a Practical Use-case

Most online platforms that use recommender systems tend to focus on one or a few specific domains. For example, Amazon's recommender system is designed to provide product recommendations based on customers' purchase history and browsing behavior. Though they might sell very different products, all listings follow a curated set of attributes that are used for recommendation purposes. Similarly, Netflix's recommender system analyzes user viewing behavior to generate personalized recommendations for movies and TV shows.

Roco[6] is an (as of now theoretical) social platform based on trust-aware peer-to-peer recommendations. Its basic function is a space for users to share externally hosted content with each other, using links or by name. A core concept of this platform is the ability to share anything the user wishes: movies, artists, shows, books, or even places A recommendation system built on top of such a platform would require being domain-independent, as the shared items could have highly differing sets of attributes. We believe this constitutes a valid real-life use case for the product of this research, as existing recommendation techniques are not directly applicable here:

1. Common content-based techniques rely on analyzing the attributes or characteristics of items to recommend similar items to users. However, as mentioned, the items shared on Roco's platform may have highly differing sets of attributes, making it challenging to find meaningful similarities between them.
2. Collaborative filtering commonly relies on analyzing user behavior or preferences to recommend items that other similar users have also liked. As the system would not have to rely on content descriptions, it might seem possible to use this technique. However, the wide disparity and variety of data would accentuate the cold start problem. Collaborative filtering is thus best used to complement an existing content-based system, which we might investigate given enough time.
3. Cross-domain recommendation and transfer learning may be useful to some extent in a domain-independent recommendation system. However, these techniques typically require a significant amount of data and pre-existing knowledge about the domains/users in question, which may not be available or applicable in the case of Roco's platform. Most research on the topic is additionally limited to a small number of domains (generally two), making it hard to apply to this situation.

[4]https://scholar.google.com/
[5]https://www.researchgate.net/search
[6]See: https://goroco.me/#/

4. Graph-based recommendation approaches lend themselves well to applications where data is highly connected or varied. This means we can exploit these techniques for our research. Cross-domain recommendations remain an open research question though [112], meaning that the approach is not directly applicable to our use case.

More detail on each of these recommendation techniques is given in Chapter 3.

1.2 Research Questions

The context in which a recommender system is used typically restricts it to a particular type of item, and is, as such, also built accordingly. Depending on the targeted domain, different algorithmic approaches should typically be taken [105]. Without prior knowledge of the data handled by the recommendation system, it can be challenging to build a system that can effectively generate recommendations for any domains.

Therefore, our **main research question** is the following:

How can unsupervised learning techniques be employed to develop a domain-independent recommender system capable of providing personalized recommendations across multiple domains?

To ease into this question, we subdivide it into the following sub-research questions.

RQ1: *What data source can a domain-independent recommender system rely on for background knowledge?*

RQ2: *How can a domain-independent recommender system effectively generate personalized recommendations across multiple domains?*

RQ3: *Which methods can be used to properly evaluate a domain-independent recommender system?*

1.3 Thesis Organization

This thesis is organized as follows: Chapter 2, Preliminaries, introduces basic concepts that will be used throughout the thesis. In Chapter 3, Related Work, we introduce current research on recommender systems and relevant methods for our work. In Chapter 4, Problem Statement and Justification of Methods, we begin with a formal definition of the primary task we are addressing. Furthermore, we present further analysis of the approaches taken to address our objectives, based on the research presented in Chapter 3.

In Chapter 5, Methodology, we construct the experimental model based on the theoretical methods devised in Chapter 4. Chapter 5 provides a detailed description of the construction process of each module within our system and outlines the algorithmic flow of each component.

In Chapter 6, Experimental Evaluation, we individually evaluate the components presented in Chapter 5. We meticulously record the experimental results and present them graphically for clarity. Additionally, we analyze the results of each experiment, speculate on the reasons behind the outcomes, and draw conclusions. Due to space limitations, we include some experimental results in the Appendix A.

Finally, in Chapter 7, Conclusion, we reiterate and answer the main research question and sub-research questions mentioned in the Introduction (Chapter 1). We summarize and discuss the findings from the experiments and analysis. Moreover, we highlight potential limitations that might affect the model's performance and suggest avenues for future research.

Chapter 2

Preliminaries

This chapter serves to introduce the essential methodologies and terminologies that will be used throughout the thesis. The scope of the topics discussed here extends to both this thesis and to related work (see Chapter 3). Many of these concepts might already be common knowledge to the reader, but we would nonetheless like to ensure that they are properly understood and to provide the necessary background knowledge. While the chapter is designed to be self-contained, a basic understanding of machine learning and data science is required.

2.1 Knowledge Graphs

Knowledge Graphs (KGs) are data structures that can store relationships. Using them it is possible to integrate data from various sources, including structured and unstructured data, to create a comprehensive representation of knowledge. KGs are represented as directed graphs, where entities are nodes and relationships between entities are labeled edges. Among more common uses, this allows for semantic reasoning and inference, enabling the discovery of implicit relationships and new insights [27].

Knowledge graphs integrate diverse data sources into a structured representation of knowledge, facilitating powerful knowledge discovery and reasoning. They connect entities through labeled edges, capturing relationships and enabling semantic reasoning. This integration of data from different sources enables a comprehensive understanding of the domain and supports domain-independent recommender systems. By utilizing the structure and semantics of the knowledge graph, recommender systems can provide accurate and personalized recommendations based on deep insights and contextual information [17, 92].

2.2 Linked Open Data

Sir Tim Berners-Lee (inventor of the World Wide Web), introduced Linked Data in a note in 2006[1]. The idea is to structure data in such a way that it interlinks with other data, making it more useful through semantic queries. The four main principles he introduced are the following:

- Uniform Resource Identifiers (URIs) are used to name individual things (we will call them "items" or "entities" in this work).
- HTTP URIs should be used to allow these things to be looked up, interpreted, and subsequently "dereferenced".
- Useful information about what a name identifies should be provided through open standards.
- When publishing data on the Web, other things should be referred to using their HTTP URI-based names.

[1] https://www.w3.org/DesignIssues/LinkedData.html

Data available on the web in such a structured and standardized format essentially forms a large knowledge graph of interconnected datasets, known as the Linked Open Data (LOD) cloud. This is possible due to the previously mentioned open standards, which ensure all datasets are compatible with each other. Please refer to subsections 2.2.1 and 2.2.2 for a brief introduction to these standards.

Since its inception, LOD has grown to a huge decentralized and open data space, connecting about a thousand interlinked datasets containing data from many domains, such as finance, healthcare and everything in between[101]. The richness and variety of data has been recognized as a valuable source of data for machine learning tasks such as categorization, data mining, and knowledge discovery in general[96], as well as for information retrieval and recommendation systems[33].

2.2.1 Resource Description Framework (RDF)

The Resource Description Framework (RDF) is a W3C standard[2] for representing graph data, which consists of nodes and edges. This makes it suitable for capturing complex relationships. The fundamental unit of data in RDF is the "triple," consisting of a subject, a predicate, and an object. The subject is the resource being described, the predicate specifies an aspect or attribute of the subject, and the object is the value or another resource related to the subject.

Consider, for example, the triple (John, hasAge, 25). Here, "John" is the subject, "hasAge" is the predicate, and "25" is the object. In reality, the subject should be a URI, which can then be used to identify "John" across the web. The object here is called a literal value, which can be static values such as numbers, strings and dates. URIs can naturally also be used in order to link a subject to another object. As for the predicate, URIs are also used to identify them.

RDF uses so-called namespaces to avoid conflicts between identical terms from different vocabularies. For example, the term "title" might have different meanings in a library dataset and a movie dataset. Furthermore, knowledge representation languages can be used to add semantic meaning to RDF data, such as RDF Schema (RDFS) or Web Ontology Language (OWL). These languages allow for the definition of classes, properties, and the relationships between them, providing a richer context for machine understanding.

There are several ways to serialize or write down RDF data, including XML, N-Triples[3], Turtle[4], and JSON-LD[5]. To make it usable, RDF data is typically stored in graph-based databases known as "triplestores", which can be queried using the SPARQL query language. This enables complex queries and data retrieval from the RDF graph.

2.2.2 SPARQL Protocol and RDF Query Language (SPARQL)

SPARQL is a query language and protocol used for querying RDF datasets. It is also a W3C standard[6], ensuring functionality across various platforms and systems. Its syntax resembles SQL, with different query types to operate on a database:

- SELECT: Retrieves data in a table format.
- CONSTRUCT: Creates new RDF triples based on existing ones.
- ASK: Returns a Boolean value based on the query's success.
- DESCRIBE: Returns an RDF graph that describes resources found.

SPARQL queries consist of graph patterns that are matched against one or more RDF graphs. For example, if it is used via HTTP on the Linked Open Data cloud, the query can be executed against multiple sources at once (federated queries). Variables are used in the queries as placeholders, making it possible to write flexible and reusable queries. Similarly to SQL, SPARQL queries allow filtering using a set of built-in functions and operators.

[2]https://www.w3.org/RDF/
[3]https://www.w3.org/TR/n-triples/
[4]https://www.w3.org/TR/turtle/
[5]https://json-ld.org/
[6]https://www.w3.org/TR/rdf-sparql-query/

2.3 Optimization Problems and Associated Algorithms

This section provides an overview of some fundamental optimization problems and algorithms that will be required in this thesis. In particular, we discuss the Set Cover problem, and the Divide and Conquer, and Binary Search algorithms.

2.3.1 Set Cover

The Set Cover problem is a well-known combinatorial optimization problem. Given a universe set $U = \{u_1, u_2, \ldots, u_n\}$ of n elements and a collection $S = \{s_1, s_2, \ldots, s_m\}$ of m where $s_i \subseteq U$, the Set Cover problem seeks the smallest subset $C \subseteq S$ such that the union of the elements in C equals U, i.e., $\bigcup_{c \in C} c = U$ and $|C|$ is minimized. This problem is NP-hard, and it has applications in various domains like network design, database systems, and machine learning.

2.3.2 Divide and Conquer

Divide and Conquer is a versatile algorithmic paradigm that involves breaking a problem into smaller sub-problems, solving the sub-problems recursively, and then combining the solutions to solve the original problem. This strategy is effective for sorting, such as in the "quick sort" algorithm. It works by following three steps iteratively or recursively:

- **Divide**: Split the problem into smaller sub-problems.
- **Conquer**: Solve the sub-problems recursively.
- **Combine**: Merge the solutions to form the solution for the original problem.

2.3.3 Binary Search

Binary Search is an efficient searching algorithm that finds the position of a target value within a sorted array. The algorithm repeatedly divides the search interval in half and compares the target value to the midpoint of the interval, narrowing down the range where the target value can be. Binary Search has a time complexity of $O(\log n)$. Given an array of length n, the basic algorithm works as follows:

1. **Initialize**: Start with two pointers $\text{low} = 0$ and $\text{high} = n - 1$.
2. **Check** Midpoint: Calculate $\text{mid} = \frac{\text{low}+\text{high}}{2}$.
3. **Compare and Update**:
   - If $\text{array}[\text{mid}] = \text{target}$, return mid.
   - If $\text{array}[\text{mid}] < \text{target}$, set $\text{low} = \text{mid} + 1$.
   - Otherwise, set $\text{high} = \text{mid} - 1$.
4. **Repeat**: Continue until $\text{low} \leq \text{high}$.

Chapter 3

Related Work

The field of Recommender Systems (RS) has been the subject of extensive research since the publication of its first papers in the 1990s. They are mostly used in e-commerce, entertainment, and personalized content recommendation areas [120]. Some famous and well-used user cases of recommender systems include movie and TV show recommendations on streaming platforms like Netflix and Amazon Prime Video, music recommendations on platforms like Spotify and Apple Music, and product recommendations on e-commerce websites such as Amazon.

Today, they are commonly subdivided into four types: content-based, collaborative filtering-based, knowledge-based and hybrid-based [120]. In the following sections, we will examine the current state of research on each of these topics. We will also look into novel techniques, such as cross-domain recommendation, graph-based methods (heterogeneous or not), LOD-based RSs, automatic feature selection and semantic aware RSs. An overview of all papers mentioned in this section can be found in the Appendix in Table A.1.

3.1 Common Techniques

3.1.1 Content-based Recommendation Systems

Content-based recommender systems rely on the attributes of items (e.g. products, movies or songs) to find content that is similar to what the user has already shown interest in. Interest is represented by past actions, such as ratings or purchases, which is then used to build a user profile of preferences based on these item's attributes [74]. Content-based recommendation systems are capable of identifying the unique preferences of individual users and suggesting uncommon items that may not appeal to a wider audience. This is also a limitation since their ability to suggest items is only based on a user's existing interests [86].

Despite this, research into content-based recommender systems shows interesting ways to solve problems that are common across the field. This is because information about the "content" itself (the attributes making up items) is valuable in nearly all scenarios. It has been shown that in some application domains, content-based approaches are more effective than purely collaborative filtering based alternatives [56, 61, 73].

Traditionally, content-based methods rely on textual content [100]. **Natural language processing techniques** are often used to further exploit item descriptions and create or fill in missing attributes [44]. More recent developments also make it possible to recommend items based on non-contextual content, such as image and video. We use images a lot as they can convey information more efficiently than text, and so including them in recommender systems can help make item representations more accurate. [77] used a Convolutional Neural Network on product images to learn interesting visual features that can be used independently of the product type. Similarly, [35, 31] extract features from movies (e.g. colors, textures, lighting and motion) using neural networks to further enhance recommendations.

More input information appears to correlate to more accurate recommendations. This is further proven by papers incorporating so-called **heterogeneous information networks**, which combine different types of attributes that are separated by different types of semantic relations. [47] utilizes social networks for recommendations, with promising results. [62] used a similar network to generate explainable recommendations. On a side note, [43] showed that using content-based information can be used to efficiently generate explanations for recommendations, which makes the process more transparent and fair to the user. [57] takes the concept of heterogeneous information networks even further by introducing multidimensional user profiles and provides a proof of concept with music recommendation. Instead of only focusing on past listening activity, this system incorporates a lot of side information that give clues about a user's preferences.

Progress has also been made to improve the algorithms that use the previously described features to more efficiently generate recommendations. In the case of heterogeneous information networks, [108, 104] proposed algorithms capable of relating two objects through a sequence of multiple connections called **meta-paths**, which is useful in many domains such as movie recommendations. [119] also uses meta-paths to extract latent features of the network. These are hidden patterns derived from different types of connections to capture how users and items are related. By considering user preferences inferred from their interactions with items, personalized recommendations can be made by taking into account the diverse relationships between items and other entities.

Representation learning

A core challenge in building effective recommender systems is the representation of items (and users) in a manner that captures their underlying characteristics and relationships. Such data mining is necessary due to the large scale of the data recommender systems have to treat.

Traditionally, **latent spaces** or **TF-IDF** are used to extract meaning from data [22]. A latent space is a reduced-dimensionality representation of high-dimensional data, often generated using techniques like principal component analysis (PCA), autoencoders, or t-SNE to find directions (principal components) that maximize variance. The original data points are then mapped to the new lower-dimensional space. This effectively captures the most important features or "latent variables" from the original data, which is a form of feature selection [28]. TF-IDF, on the other hand, is a statistic that reflects the importance of a word to a document in a collection or corpus. It is commonly used in information retrieval and text mining. TF (Term Frequency) measures the frequency of a term within a specific document, while IDF (Inverse Document Frequency) measures how rare or common a word is across all documents in the corpus. The product of TF and IDF gives the TF-IDF score for a word in a particular document, which can be used to rank the relevance of single features [81].

Embeddings have emerged as an alternative to latent spaces or TF-IDF for representing data, as they can capture semantic relationships between items in a dense, continuous vector space. They are commonly used in Natural Language Processing applications, where embeddings can be created for every word from corpora of documents to capture their semantic and even syntactic relationships. Word2Vec is a commonly used word embedder [66]. The process can also be applied to graphs, though, such as networks or knowledge graphs.

Word embeddings using Word2Vec

Word2Vec [66] is a very efficient neural network-based technique for natural language processing (NLP) tasks, designed to capture semantic relations between words from raw text. It works by mapping words to high-dimensional vectors where semantically similar words are located near each other. Although it is primarily used in Natural Language Processing (NLP) applications, it can also be applied to other domains such as knowledge graphs, including RDF [97].

The algorithm offers two different embedding models, the Continuous Bag-of-Words model (CBOW) and the Skip-gram (SG) model. The first (CBOW), works by trying to predict a target word based on its context. SG, on the other hand, does the opposite as it tries to predict the context (surrounding words) given a target word. CBOW trains faster as it predicts one word based on multiple context words at once, and is generally better at understanding the frequency of words. SG, despite being slower, is better at capturing infrequent words and their semantics compared to CBOW by utilizing subsampling of frequent words [79]. The evaluation of RDF2Vec in [97] has also shown that the Skip-Gram model outperforms CBOW for recommendation tasks.

Graph embeddings (GE) can be used to represent the topological structure of entire graphs or subgraphs as vectors in a continuous space. Nodes are often treated as individual entities whose relationships (edges) need to be captured. General algorithms like Node2Vec [48] or GraphSAGE [51] use random walks or neighborhood aggregation to capture the structural and feature-based similarity between nodes. The walks or neighborhoods serve as the context, much like the surrounding words for word embeddings. Through this process, the algorithm learns to map nodes to a low-dimensional space of features that maximizes the likelihood of preserving network neighborhoods of nodes. The primary advantage of GE is that they can effectively capture the topology [18].

Knowledge graph embeddings (KGE) are also a type of graph embedding, but are tailored towards knowledge graphs, where nodes are entities and edges represent relations between these entities. While GE are primarily concerned with capturing the topological structure of the graph, KGE focus on capturing the semantics and types of relationships between nodes (entities) and edges (their relations) [111]. A commonly used algorithm for creating KGE is TransE [114], which models relationships as translations in the embedding space. For a triple $(head, relation, tail)$, the equation $head + relation = tailhead + relation = tail$ should hold in the embedding space. Many variants of TransE exist, such as TransH, TransR, TransD. These extend TransE by adding more flexibility, such as allowing different relation types to have different hyperplanes or transformations. TransE has been shown to have difficulty capturing 1-N or N-1 Relations, symmetric relations and tens to over-simplify complex relationships [114].

RDF2Vec falls under the broader category of KGE, but is specifically designed to generate embeddings for RDF graphs, meaning that it can be used to generate embeddings in the Semantic Web and Linked Data environments [97]. Because RDF graphs are rich in semantics, RDF2Vec aims to capture both the structure and the semantics of the nodes and relationships. The unsupervised technique also makes use of randomly generated walks, which are fed into word embedding algorithms such as Word2Vec as "sentences".

Regarding using embeddings for recommendation, [109] introduced Meta-Prod2Vec, a hybrid model that leverages music playlists and listening data as interactions for collaborative filtering (see Section 3.1.2), using a sequence-based approach to compute low-dimensional embeddings of item metadata for item recommendation. This way of item representation leads to higher performance in many scenarios. [117] further proved this by implementing a recommender system using embeddings that is capable of running on a graph with 3 billion nodes and 18 billion edges (extracted from the social media network pinterest.com[1]). The embeddings of items are generated using a 'graph convolutional network' algorithm that uses random walks and graph convolutions and incorporate both graph structure and node information.

3.1.2 Collaborative Filtering-based Recommendation Systems

Collaborative filtering-based recommendation systems utilize the collective behavior of a group of users to make personalized recommendations. The system analyses the past actions of similar users to predict items that the current user may also enjoy [2]. This approach does not only rely on the attributes of items like content-based systems do, but instead focuses on the user's behavior and preferences. Collaborative filtering systems are able to suggest a wide range of items, including those that are less popular or not commonly associated with the user's existing interests. However, one of the main challenges of collaborative filtering is the cold start problem, where new users with no or limited history may struggle to receive recommendations [122]. Additionally, this method may not be effective for users with unique or niche interests, as the system requires a large amount of data to generate accurate recommendations [102]. Some privacy concerns can also arise, as there is a need to share user data [36].

The simplest form of the method involves recommending items based on the preferences of similar users or the similarities between items (also called **memory-based** approach) [89]. For example, if User A has liked or rated similar items to User B in the past, the system recommends items liked by User B to User A, assuming their preferences align. Depending on the perspective, the approach can be categorized as either user-based or item-based. The first recommends items based on the preferences of similar users, while the second suggests items similar to the ones the target user has interacted with in the past. User-based filtering is effective when users have similar tastes, while item-based filtering works well when items have consistent ratings or features [89]. Beyond

[1] https://www.pinterest.com/

a memory-based approach, it is possible to make use of statistical and machine-learning models to capture user preferences more accurately [5].

The first of such model-based approaches use **clustering algorithms**. They are a form of unsupervised machine-learning that structures data based on a predefined model. Users with similar interests are grouped together to make recommendations within their “cluster” neighborhood, which have high intra-cluster similarity and low inter-cluster similarity [59]. A commonly used clustering algorithm is K-mean as it is simple to implement and provides good results [60, 1].

Association rule mining is another popular approach for collaborative filtering. The technique focuses on discovering interesting relationships in large datasets. Association rules are created by identifying frequent itemsets, which are subsets of items that frequently occur together and can be recommended. Commonly used algorithms include Apriori and FP-growth, which are especially effective in the e-commerce sector [49, 4, 71].

In addition to clustering and association rule mining algorithms, **Bayesian networks** provide valuable insights into the relationships and patterns within data. They are probabilistic graphical models that represent the relationships between users, items, and their attributes using a directed acyclic graph (DAG) [59]. Each node in the graph represents a variable, such as a user’s preferences or an item’s features, and the edges represent the probabilistic dependencies between these variables. By learning the structure and parameters of the Bayesian network from the data, the system can infer missing values and make personalized recommendations. Bayesian networks can capture both explicit and implicit dependencies among variables, making them well-suited for recommendation tasks, especially in the context of movie, social media and location-based recommendation systems [42, 15, 10].

3.1.3 Knowledge-based Recommendation Systems

Knowledge-based recommendation systems suggest items based on explicit knowledge about the user’s preferences and requirements. The recommendations are based on logical deductions rather than the behavior of the user or the attributes of items [17]. Knowledge-based systems are particularly useful in domains where the items are well-defined and where there is a large amount of domain knowledge available. They are usually applied in situations with low ratings, such as for financial services (jariha 2018). However, they are limited by their inability to suggest items that do not fit into the predefined set of requirements or preferences. Additionally, these systems may struggle with large datasets, and require significant amounts of manual input to develop a robust knowledge base. The limitation of knowledge-based recommendation system is the manual collection and construction of knowledge, meaning not much research exists on them (jariha 2018). Nonetheless, we distinguish two types of knowledge-based recommendation approaches:

Constraint-based recommendation systems suggest items based on explicit constraints defined by the user. These can be specific requirements or preferences that the user wants the recommended items to fulfill. The system then uses logical deductions to match the user’s constraints with the attributes or characteristics of the items. This works particularly well in domains where the user’s requirements are well-defined and can be explicitly specified [37, 38].

On the other hand, **case-based recommendation systems** suggest items based on similarities between past cases or experiences and the current situation. These systems analyze a collection of past cases and find similarities between the attributes of those cases and the current user’s context. By identifying relevant similarities, case-based systems recommend items that have been successful or suitable previously. This approach is useful when explicit knowledge about user preferences is limited or difficult to obtain, and when historical data or cases are available to draw from [16, 17, 75].

3.1.4 Hybrid-based Recommendation Systems

Hybrid-based recommender systems combine the strengths of multiple recommendation approaches to provide more accurate and diverse recommendations. This could be a combination of the three previously seen methods but also extend to other, more novel, methods we will look at in Section 3.2. The hybrid approach can leverage the strengths of each method while mitigating their respective weaknesses [121]. For example, we could take advantage of the collaborative filtering approach to identify items that are similar to the user's past behavior and then use the content-based approach to further refine the recommendations based on the item's attributes.

The simplest way to combine multiple techniques is to put together their results (**mixed hybrids**) [19]. Different, and more advanced hybridization approaches also exist though, the most popular of which is **weighted hybrids** [19]. This approach simply aggregates the results of multiple recommendation techniques using weighted linear functions. [11] successfully combined a collaborative-filtering approach with content-based filtering to alleviate the cold-start problem. [118] shows that it is also possible to combine one method with itself. The proposed method for social media recommendations uses three collaborative filtering-based systems to evaluate user-to-user, user-to-tags and user-item relations. The resulting scores are then linearly combined.

Another way to combine recommendation approaches is **feature combination**. In this case the output of one system is used as part of the features of another. For example, [12] uses the results of a collaborative-filtering model on book reviews which later is used as input for content-based filtering. According to the authors, this resulted in performance improvements over single techniques. A similar technique is **feature augmentation**: instead of using the output of one system as features, it is used to create new attributes in the item description. [82] uses this technique for book recommendation, where collaborative filtering is used to find "related authors" and "related titles" to be used by a subsequent system.

Similarly, **cascade hybridization** is a sequential approach of recommendation generation, where the outputs of one recommender system are used as inputs to another recommender system in a cascading manner. The order of the recommendations matters, as each step in the system refines the ranking. [68] used this technique for music recommendation using two levels: the first utilizes a multi-class SVM classifier, a commonly used supervised machine learning algorithm, to categorize songs based on their genre, while the second focuses on personality diagnosis. The approach assumes that user preferences for songs reflect their underlying personality, and it estimates the probability of active users sharing the same personality type as others. Consequently, the recommender calculates the personalized probability of an active user's likelihood to enjoy new songs. In [65], the authors build a recommendation framework that combines two collaborative filtering systems with different properties. The first module retrieves data and generates neighbor lists for users using Pearson's coefficient and Euclidean distance. In the second module, they experiment with three predictors: Bayesian estimator, Pearson's weighted sum, and adjusted weighted sum, finding that the Bayesian prediction performs the best.

3.2 Novel Techniques

3.2.1 Cross-domain Recommendation Systems

Cross-domain recommender systems (known as CDR) leverage data and insights from multiple domains and the corresponding sets of users to make personalized recommendations. They are mostly used as a way to overcome the cold start problem by utilizing data from other domains to make predictions for new users or items. However, one of the challenges of cross-domain recommendation systems is the integration of data from multiple sources, which may require extensive preprocessing and data cleaning. Additionally, the differences in data and features across domains may lead to bias or inaccuracies in the recommendations. Overall, cross-domain recommender systems are best suited for situations where the user has diverse interests and where there is a significant overlap between different domains. [124]

Research into cross-domain recommendations is subdivided into subdomains based on the number of targeted domains (single, dual or multi). Interesting approaches are used to overcome the challenge of cross-domain recommendation. [14, 69] propose content-based solutions that transfer knowledge after making connections between both domains using user and item attributes. [64, 123] similarly use semantic similarity and correlation.

3.2.2 Graph-based Recommendation Systems

Graph-based recommendation systems rely on graph structures that represent relationships between users and items. Representing data in this way allows for more flexibility and variation, and can capture both explicit and implicit relationships between users and items [112]. This can enable graph-based methods to capture complex patterns and similarities between items and users, leading to more accurate recommendations [32].

Many interesting approaches have been taken to leverage graph-based representations for recommendation. [9, 58, 8] utilize random walk-based approaches which model the implicit preferences or interactions among users and/or items by allowing a random walker to walk on a given graph, and then use the probability of the walker landing on nodes to rank candidate nodes for recommendations. However, their drawbacks include low efficiency due to generating ranking scores for all candidate items at each step, and a lack of model parameters to optimize the recommendation objective. [113, 103, 21, 52] make use of graph embedding approaches which encode the complex relationships between nodes, such as users and items, into low-dimensional vectors. This approach allows the modeling of heterogeneity and sparsity of nodes in a graph. The downsides include sensitivity to the quality and quantity of input data, computational complexity, and limitations in modeling complex dynamics.

3.2.3 Named Entity Disambiguation

While the richness of data in the LOD cloud offers many opportunities for various computational tasks, including fine-grained and contextually relevant recommendations, the task of mapping user queries to this extensive, interconnected dataset is not trivial. [78] introduced the task of mapping search engine queries to linked data. In their evaluation, they identify DBpedia[2] as "an integral part of, and interlinking hub for, the LOD cloud", which is why their efforts are directed towards mapping queries to this ontology. They found that a simple lexical match between input queries and concepts did not perform sufficiently well, and developed a new approach employing a combination of information retrieval and machine learning methods. Despite being initially designed for Dutch resources, the approach is language-independent and could also be extended to other ontologies. A major limitation of the method is its reliance on manual annotation, meaning the input queries are limited to the given system and domain.

A key factor to the problem of mapping is the ambiguity between potential matches to a given query. The task of addressing this ambiguity for named entities is called Named Entity Disambiguation (NED). Recent approaches exploit entity semantics to better distinguish between candidates. [125] utilize context given by an input search query (i.e. "What movies did John Noble play") and semantic similarity of candidate entities to improve on previous NED methods only exploiting textual similarity.

3.2.4 Linked Open Data Based Recommendation Systems

Closely related to the graph-based recommendation systems laid out in the previous section, Linked Open Data (LOD) is a promising avenue for improving existing approaches. The large amounts of data made openly and easily accessible can be used to mitigate the new-user, new-item and sparsity problems commonly encountered in recommender systems. [54, 33, 95] first showed that using LOD to build open and collaborative recommender systems is feasible, comparatively easy but also more effective. [44, 85] presents how a semantics-aware approach can further take advantage of LOD's graph structure and produce better results. [110] investigates the role of ontology-based data summarization for feature selection in recommendation tasks. [99] utilizes LOD for knowledge transfer between domains with user overlap. The technique decomposes information into user, item and domain latent factor matrices. While still dependent on the application domains, this technique is a promising approach for our study. [6] uses LOD to sort recommendations made by another recommender system and also makes them explainable. The downside to this system is that it does not exactly explain the choice of the underlying recommender system, but rather comes up with a plausible explanation.

3.2.5 Unsupervised Feature Selection

One of the main challenges associated with building recommender systems on Linked Data is related to feature selection due to the large variety of features. Linked Data is often very rich and contains much information that may be irrelevant and noisy [93]. Many papers making use of LOD manually select a subset of features that fit

[2]https://www.dbpedia.org/

their purposes, but this is not a solution when the domain of data is undefined or the scale of data is too big. In such cases, feature selection is commonly employed to solve this problem.

Commonly used techniques for choosing features in machine learning can be categorized into three categories. The first, known as "filters," employs statistical metrics like Information Gain, Entropy, or Mutual Information to score each feature individually [45]. This method typically occurs before the actual learning phase and operates independently. The second approach, referred to as "wrappers," treats the issue as a search problem and uses the learning algorithm as a black box to evaluate various feature subsets, for instance through forward selection or backward elimination [63]. Lastly, embedded methods incorporate feature selection directly into the training process, such as Nested subset techniques [50].

These techniques are sufficiently general that they can be applied to a wide variety of use cases, including feature extraction on Linked Data. Other methods that are built with Linked Data in mind can make use of its semantic properties and achieve better results. [107] is an early approach to the problem, which uses Linked Open Data for unsupervised generation of data mining features. The proposed approach uses six feature generators (SPARQL queries) to extract multiple categories of features based on the item attributes, their types, and different relations between the item and its neighbors in the LOD cloud. A subsequent feature selection step uses a simple heuristic to filter out the least complete features. Though some experimentation is needed to produce good results the results of the study are promising.

A recent paper by Di Noia et al. presents a novel method for automatic feature selection in the context of Linked Data using ontology-based data summarization [110]. The paper compares a fully automated feature selection method based on ontology-based data summaries (ABSTAT), with more classical ones (Information Gain (IG), Gain Ration and Chi Squared Test) and evaluates the accuracy of these methods using a recommender system exploiting the top-k-selected features. Their results show higher accuracy over traditional methods, meaning the technique could be leveraged to build a domain-independent RS.

3.2.6 Semantic-Aware Recommendation Systems

Semantic-aware recommendation systems retrieve and make use of semantic information found in knowledge graphs, such as in the Linked Open Data cloud. This can enhance the accuracy, relevance, and explainability of recommendations. In essence, this means capturing the underlying meaning and relationships between users, items, and their attributes. By mapping user profiles and item descriptions to a common ontology, these systems can infer explicit and implicit relationships, providing more precise and context-aware recommendations. Additionally, semantic-aware approaches utilize semantic similarity measures to identify users with similar preferences and recommend semantically related items, bridging the gap between different domains and addressing the cold start problem [44].

The availability of Linked Open Data further enriches these systems by leveraging interconnected LOD resources to deliver open and collaborative recommendation systems [83, 84, 80]. Furthermore, semantic-aware recommendation systems can provide explanations for their recommendations by utilizing semantic annotations and logical rules [40]. Automatic feature selection techniques using ontology-based data summarization reduce the complexity and dimensionality of the recommendation task, improving the accuracy and domain independence of the system [98, 110]. Overall, semantic-aware recommendation systems enhance the recommendation process by incorporating semantic information, improving accuracy, relevance, diversity, and transparency.

Chapter 4

Problem Statement and Justification for Method Selection

The objective of this research is to determine the feasibility of building a domain-independent RS. This will involve exploring and evaluating different approaches to developing our system. This section serves to justify these approaches from a theoretical point of view. We make use of the previously established preliminaries (Chapter 2) and related work (Chapter 3) to lay out the theoretical foundations for the Methodology laid out in Chapter 5.

4.1 Problem Statement

We can formally define a **domain** as a collection of items that belong to the same category and have some common attributes that can be used to describe them and measure their similarity. For example, the domain of "movies" may include items such as "Toy Story (1989)", "Toy Story 2 (1999)", and "The Room (2003)". A domain can be represented as a set $D = \{i_1, i_2, \ldots, i_n\}$.

Each item i has a **feature vector** $f(i) = [a_1(i), a_2(i), \ldots, a_j(i), \ldots, a_m(i)]$, and $a_j(i)$ is the value of the j-th **attribute** for item i. Each value $a_j(i)$ of the feature vector represents an attribute j that stays constant throughout D. The set of attributes of a domain is represented as $A(D)$. In the "movie" domain, these attributes could be "director", "composer", and "running_time". For instance, "Toy Story (1989)" could have a feature vector f`("Toy Story (1989)")=["John Lasseter (director)","Randy Newman (composer)","81 minutes (running time)"]`.

A domain-independent RS operates on a **domain-independent feature space**, which we denote as F, which is a collection of attributes that are applicable to any item in any domain, e.g., "movies", "songs", "books", "real places", ... We can define $F = \bigcup A(x)$, where $A(x)$ is the set of attributes of any domain x as defined previously.

Let us additionally define a **user** u as having a **rating history** represented by a vector $r(u) = [r_1(u), r_2(u), \ldots, r_k(u)]$, where $r_k(u)$ is the rating given to item i_k by user u. Though we use "rating", note that this does not necessarily have to be done explicitly by the user. Depending on the application, a user could simply express interest in an item by searching for it, which would result in a positive "rating". Note that the ratings of a user are given on a subset of all items across domains. For example, a given user may have rated "Toy Story (1989)" but not "The Room (2003)". The set of all users is denoted by $U = \{u_1, u_2, \ldots, u_p\}$.

The objective of the RS is to predict a score $S(u, i; D)$ for each user u (represented by their **rating history vector** as defined above) and each item i in a given domain D that has not been rated by u. This score represents the likelihood that u will like i, **based on u's previous ratings** and the **similarity between the previously rated items and i**. The attributes of D are a subset of the domain-independent feature space F.

The goal of the domain-independent recommender system is to minimize the commonly used Root Mean Squared Error (RMSE), defined as:

$$\text{Minimize RMSE} = \sqrt{\frac{\sum_{u \in U} \sum_{D} \sum_{i \in D \setminus R(u,D)} (S(u,i;D) - \hat{S}(u,i))^2}{|U| \times \sum_{D} |D \setminus R(u,D)|}}$$

Where $D \setminus R(u,D)$ is the set of items in domain D that have not been rated by user u, and $\hat{S}(u,i)$ is the true latent score for item i for user u. The value of $\hat{S}(u,i)$ is undefined until the user rates the item in question, at which time it can be compared to the prediction score.

The prediction score $S(u,i;D)$ is calculated as:

$$S(u,i;D) = f(r(u,D), sim(i, R(u,D); D))$$

Here, $sim(i, R(u,D); D)$ measures the similarity between item i and the set of items $R(u,D)$ that have been rated by u within domain D. $r(u,D)$ denotes the vector of previous ratings that user u has given to items in domain D. f is a function that maps these ratings and item-item similarities to a prediction score. **In other words, the domain-independent recommender system utilizes a user's rating history and the similarity between items to formulate an accurate prediction score for items that have not been rated by the user.** Without an existing rating history, the system acts as a retrieval system, which, given an input retrieves items similar to the input without taking into account the user since the system does not have data on them.

By providing this formal framework, the problem statement sets the foundation for building a domain-independent recommender system that is both accurate and versatile. While implementing individual objects such as users, items and ratings is not a problem, doing so across a domain-independent feature space as described above represents the challenge and contribution of this work.

4.2 Background Knowledge

Recommender systems require data to formulate accurate recommendations, and usually, the more, the better. Without information on a particular user or item, no system is capable of doing more than randomly guess likely associations. This is commonly referred to as the cold-start problem [5], and particularly affects new users or items added to the system. A common solution to this problem is to use what little data exists to supplement it from other sources. New users could simply be asked about their preferences, which then leads to first recommendations that can then be narrowed down as the user uses the system. As for new items, sufficient data points are usually added along with the item so that the system can relate it to similar items and leverage existing knowledge, such as user ratings, for example [67].

A domain-agnostic system has, by definition, no predefined knowledge of its users or items. This makes the cold-start problem particularly troublesome, since each input might come from a domain that the system has not yet seen and a relatively small subset of users might be interested in a large variety of item categories. As discussed in Section 3.1.2, this rules out using collaborative filtering to drive our recommender system.

One could imagine trying to combine many domain-specific datasets to build a large library of knowledge upon which the system can rely to learn how to generate recommendations ad-hoc, given specific scenarios. This would represent a challenging task on its own, but can luckily be circumvented thanks to the existence of Linked Open Data. As seen in Section 3.2.4, LOD can be used to leverage a wealth of information from different sources and domains in the context of graph-based recommender systems [33, 44]. It has also been shown that augmenting existing recommender systems using background knowledge from LOD can improve results without the cost of having to manage and maintain the data oneself [97]. This is highly relevant to this study, as our systems should handle a very large variety of data. Details on our implementation of this are provided in section 5.1.1.

4.3 Mapping User Input to the Linked Data Cloud

In an effort to make our system as general purpose as possible, we would like to make the user's input as minimal as possible. We cannot expect links to existing resources, or a lot of context as input to make the system work. Ideally, the user should be able to find the entity they search by simply typing its name. In order to make use of entities in the LOD cloud, our system first needs to be able to map simple textual inputs (i.e., queries) to said entities.

As seen in Section 3.2.3, the primary problem in automatic LOD mapping tasks is the ambiguity of entities in Linked Data, and a simple lexical match between input queries and concepts does not perform sufficiently well. We explored a few approaches for solving the Named Entity Disambiguation (NED) problem, but could not settle on an ideal candidate for our purposes. [78] improved on a simple matching baseline by proposing a method utilizing information retrieval and machine learning methods. The downside of this method is that it requires manual annotation for every domain, and cannot be used in an unsupervised way. The current state-of-the-art NED methods [125, 25, 24] focus on the more important issue of entity mapping for search queries. While the difference to our problem appears small, these methods are able to improve their mapping by utilizing semantic and textual context given by specific queries or surrounding words in a text. For example, the search query "What movies did John Noble play" implies the entities of 'Movie', 'John Noble' and the predicate 'play', which, in combination with semantic similarity of candidate entities, can be used to narrow down the search. Without such context, these methods cannot be applied for our use case.

DBpedia is often seen as a central interlinking hub for the LOD cloud [78, 125]. To ease access to its dataset, DBpedia offers an entity retrieval service called DBpedia Lookup[1]. It can be configured to index any data present in the DBpedia dataset, and provides a retrieval service that resolves keywords to entity identifiers. The public endpoint indexes label, comment, type and category fields, which can be queried according to a selection of query configurations, such as weighting factors, fuzzy matching, relevance boosting and relevance score among others. The service can also be ran locally off a dump.

Since our application restricts the amount of context available, and is otherwise too general to apply to existing methods, we will investigate using the DBpedia lookup tool for basic input mapping. Furthermore, we do not aim for a high mapping accuracy, since an actual user interface for our recommender system would let the user choose among options themselves instead of guessing.

4.4 Feature Selection

The objective of our system is domain-independence, which signifies that it must be capable of handling data from any domain used on it. As stated in Section 4.1, every domain is associated to a certain number of attributes. Our system retrieves the attributes associated to an item or type from the LOD cloud, as described in Section 5.1.3. Linked Data is often very rich and contains much information that may be irrelevant and noisy [93]. Feature selection is commonly employed to solve this problem.

The main focus of feature selection is choosing a subset of variables from the input data that effectively describes it while reducing the impact of noise or irrelevant variables [23]. Many approaches for feature selection exist, making use of various statistical measures. In the case of Linked Data, the semantics encoded in hierarchical ontologies can be exploited to select the most relevant features.

[93] is a recent study on schema summarization in Linked-Data-based feature selection for recommender systems in which a schema summarization tool called ABSTAT[2] is evaluated. ABSTAT is an ontology-based framework designed to automatically summarize and profile linked data to facilitate tasks like data discovery and query formulation [3]. The comparison shows higher precision over classical approaches for feature selection. The study is of particular interest to our work as it relies on background knowledge extracted from DBpedia and evaluates the results of the extraction in a recommendation scenario using the Movielens dataset. As this corresponds to our own benchmarks, we can make use of their findings to guide the feature selection of our own system.

[1] https://lookup.dbpedia.org/index.html

[2] ABSTAT summaries for several datasets are available at http://abstat.disco.unimib.it/

4.5 Representation Learning

Directly applying traditional machine learning algorithms to LOD would not be feasible due to the heterogeneity, complexity and scale of the data and its attributes. Representation learning solves this problem by translating items of the LOD cloud into vectors that are more manageable and computationally efficient. Additionally, embeddings are capable of capturing semantic and/or topological features of the graph formed by LOD. As seen in Section 3.1.1, many options exist for creating embeddings.

We started by investigating graph embeddings due to the interconnected nature of LOD. Node2Vec [48], a commonly used algorithmic framework for representational learning on graphs, has been successfully used for recommendation tasks in other works using graph data, as it is able to capture topological features well [88, 26]. It not as effective when used on Linked Data though, as it does not capture semantic relationships, which are critical in LOD [90]. While it is excellent for general graph structures, it is not optimized for the specific complexities and opportunities presented by LOD.

We also investigated knowledge graph embeddings, as they aim to solve the shortcomings of graph embeddings by capturing relations between nodes. TransE [114] and its variants are commonly used, and have also been successfully used in recommender systems [116, 70]. Unfortunately, the technique is domain specific, meaning it requires customization for specific domains and a fixed set of relationships types, making it less flexible for use in a domain-independent system.

Finally, RDF2Vec [97] appears to be the best candidate. As the technique is tailored for RDF graphs, it is able to capture the rich semantics of Linked Data. Since the technique is unsupervised, it also works cross domain-boundaries and is flexible with regard to its input. Since RDF2Vec relies on random walks generated with SPARQL, we can make use of linked resources in separate datasets which provides a broader scope of data. In their introduction [97], Ristoki et al. have shown that RDF2Vec embeddings outperform those generated by TransE and its variants in classification, entity modeling, and, most importantly to us, recommendation tasks.

For these reasons. we have chosen to use RDF2Vec for representation learning in our system.

Chapter 5

Methodology

In this section, we will present our solution to the problem statement introduced in chapter 4.1. The functionality of the proposed system can be divided into data fetching and management (see Section 5.1), and data processing (see section 5.2). An overview of the system's components and their functionalities can be found below in Figure 5.1. The processing steps used by the system are described using a flowchart in Figure 5.2. While both describe the same system, the component overview serves to highlight the interactions between components, while the flowchart highlights the flow of data from input to recommendation. The components of both Figure 5.1 and Figure 5.2 include their related section title in parentheses.

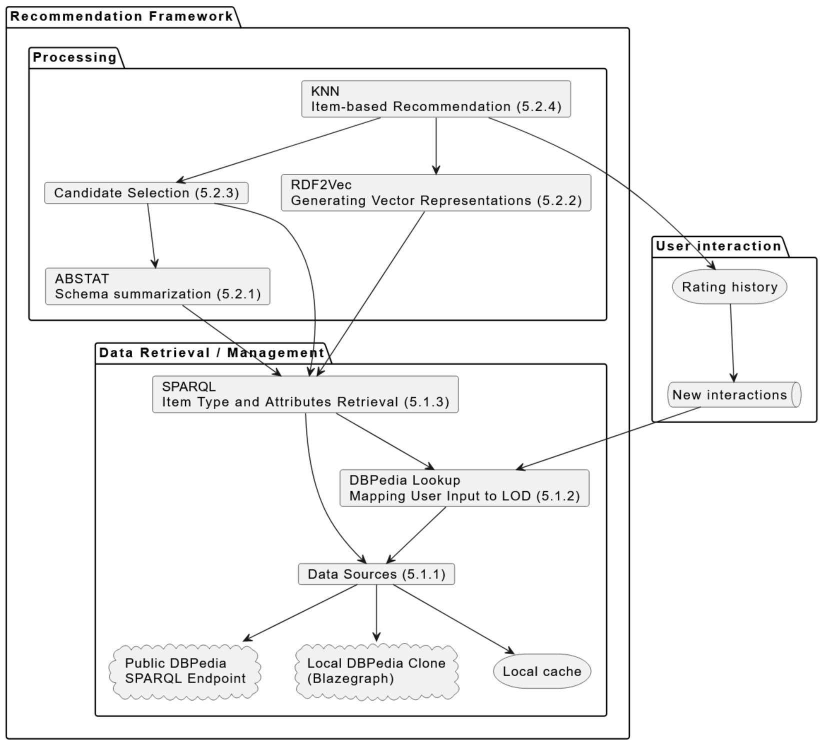

Recommendation Framework
Processing
KNN
Item-based Recommendation (5.2.4)
Candidate Selection (5.2.3)
RDF2Vec
Generating Vector Representations (5.2.2)
ABSTAT
Schema summarization (5.2.1)
User interaction
Rating history
New interactions
Data Retrieval / Management
SPARQL
Item Type and Attributes Retrieval (5.1.3)
DBPedia Lookup
Mapping User Input to LOD (5.1.2)
Data Sources (5.1.1)
Public DBPedia
SPARQL Endpoint
Local DBPedia Clone
(Blazegraph)
Local cache

Figure 5.1: An overview of components in the proposed methodology

The described system relies on the Linked Data cloud for background knowledge (Section 5.1.1). Given a user input (e.g. the name of a movie), we apply a mapping algorithm to link it to an entity in the LOD cloud (Section 5.1.2). This entity allows us to retrieve attributes, such as type, related to the item in question (Section 5.1.3). For more efficient processing and higher accuracy, we select the most relevant attributes using schema summarization (Section 5.2.1). This subset of relevant features allow us to retrieve candidates for recommendation (Section 5.2.3). To properly capture the semantic context of each item, we utilize representation learning to generate embeddings for the input item and the selected candidates (Section 5.2.2). Finally, we can use the generated embeddings to create recommendations by applying an item-based KNN with cosine similarity (Section 5.2.4).

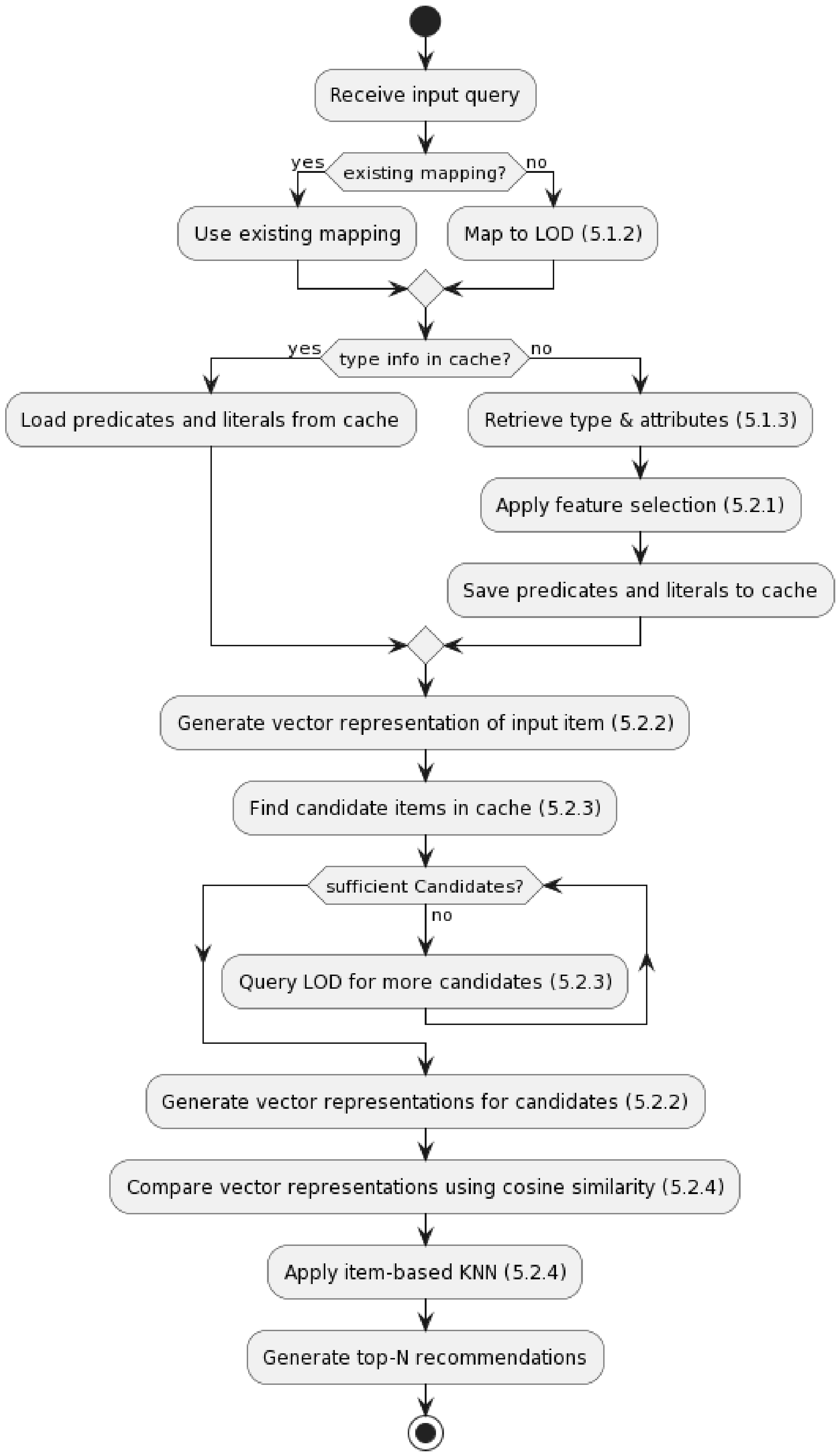

Receive input query
existing mapping?
yes
no
Use existing mapping
Map to LOD (5.1.2)
type info in cache?
yes
no
Load predicates and literals from cache
Retrieve type & attributes (5.1.3)
Apply feature selection (5.2.1)
Save predicates and literals to cache
Generate vector representation of input item (5.2.2)
Find candidate items in cache (5.2.3)
sufficient Candidates?
no
Query LOD for more candidates (5.2.3)
Generate vector representations for candidates (5.2.2)
Compare vector representations using cosine similarity (5.2.4)
Apply item-based KNN (5.2.4)
Generate top-N recommendations

Figure 5.2: The processing workflow of the proposed recommender system

Table 5.1: Comparison of Wikidata and DBpedia

Feature	**Wikidata**	**DBpedia**
Triples	12,500 million	9,500 million
Entities	107.2 million	38.3 million
Linked Datasets	55	30
Access Methods	Public SPARQL endpoint, database dumps	Public SPARQL endpoint, database dumps

5.1 Data Retrieval and Management

5.1.1 Data Sources

As justified in section 4.2, we want to investigate using Linked Open Data as background knowledge for our recommendation system. It allows us to relatively easily make use of a broad and varied range of knowledge. This section introduces the data foundations for our recommendation system. We introduce two central sources: Wikidata and DBpedia. We'll touch on our preference for DBpedia, the advantages of a local clone, and our system's capability to simultaneously connect with multiple data sources using Blazegraph.

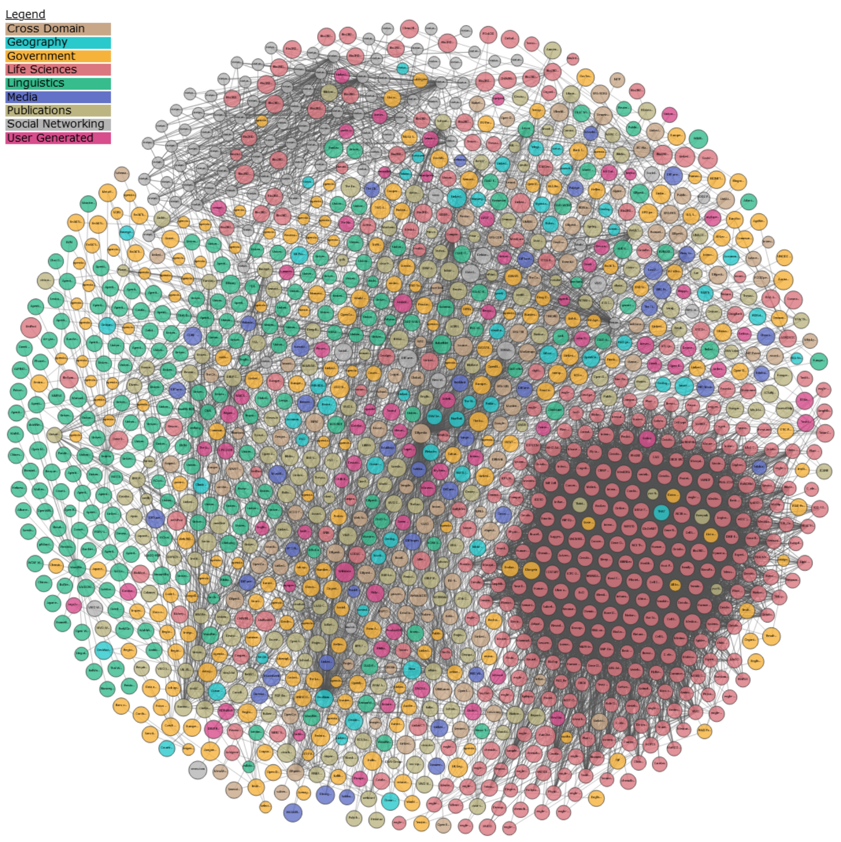
A large amount of sources exists in the LOD cloud. As the name suggests, all of them are interconnected by various properties, depending on the nature of the dataset. The Linked Open Data Cloud[1] project maintains a visualization of datasets that have been published in the Linked Data format, currently containing 1314 datasets with 16308 links. Figure 5.3(a) corresponds to the visualization as of October 2023. The datasets are colored and categorized into multiple categories: Cross-domain, Geography, Government, Life Sciences, Linguistics, Media, Publications, Social Networking and User Generated. We are of course interested in cross-domain datasets, as these also offer an entry to the rest of the cloud. In particular, DBpedia[2] and Wikidata[3] are central nodes, as can be seen from Figures 5.3(b) and 5.3(c) respectively. Both are comprehensive, widely recognized, and easily accessible.

Wikidata is a Wikimedia project and acts as a central storage for the structured data of its sister projects, including Wikipedia, Wikivoyage, Wiktionary, Wikisource, and others. At the time of writing, it contains over 12.5 billion triples in over 107 million items contributed by approximately 12 thousand active editors[4]. These items link to a total of 55 other datasets in the LOD cloud[5].The data is openly accessible through the public SPARQL endpoint, the Wikidata Query Service[6] and through regular database dumps[7].

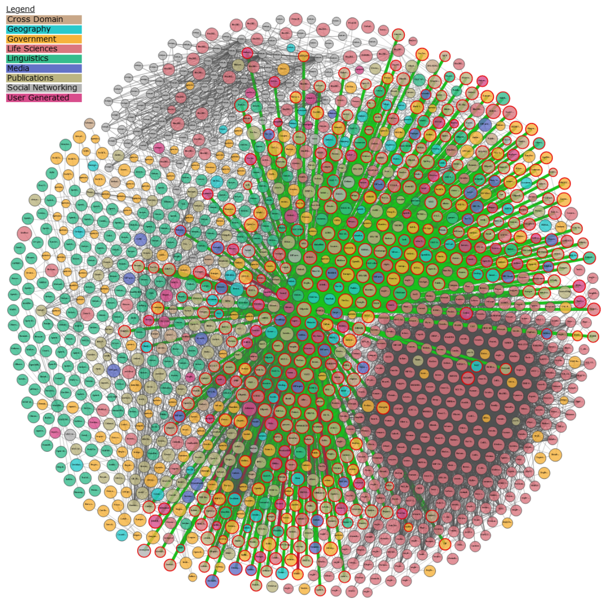
The DBpedia project also relies on Wikipedia, aiming to extract structured content from the information created in the project. At the time of writing, it contains over 9.5 billion triples in over 38.3 million items across 125 languages[8]. The items link to 30 other datasets[9]. As for Wikidata, the DBpedia data is also freely available via a SPARQL endpoint[10] and regular database dumps[11].

A comparison between Wikidata and DBpedia is drawn in Table 5.1. Looking at these statistics, it appears as though Wikidata is the better choice, as it contains more information and has more links to other datasets. But as Figure 5.3(b) illustrates well, DBpedia has evolved to serve as the nucleus for the web of open data and is recognized for its extensive coverage [7, 78]. This makes it more adequate for certain tasks, such a generating

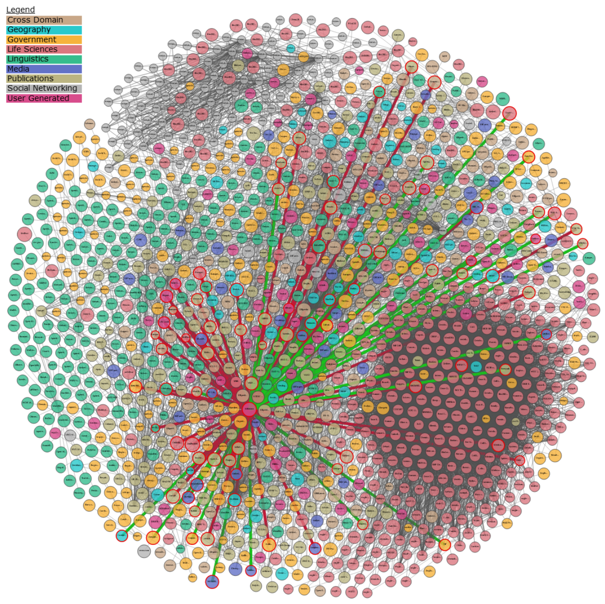
[1] https://lod-cloud.net/
[2] https://www.dbpedia.org/
[3] https://www.wikidata.org/
[4] https://www.wikidata.org/wiki/Wikidata:Statistics#noSuchAnchor
[5] https://lod-cloud.net/dataset/wikidata
[6] https://query.wikidata.org/sparql
[7] https://www.wikidata.org/wiki/Wikidata:Database_download
[8] http://wikidata.dbpedia.org/about
[9] https://lod-cloud.net/dataset/dbpedia
[10] https://dbpedia.org/sparql
[11] https://www.dbpedia.org/resources/latest-core/



(a) Without Highlighting



(b) With DBpedia and its Connections Highlighted



(c) With Wikidata and its Connections Highlighted

Figure 5.3: The Linked Open Data cloud from lod-cloud.net

vector representations for recommendation tasks [97] and named entity disambiguation [125]. Since our system relies on these tasks (see Sections 4.5 and 4.3), we chose to build our recommender system's background knowledge on DBpedia rather than on Wikidata. Please note, though, that both datasets are easily accessible and queryable using SPARQL, making it seamless to switch between them or even combine their results. The system can relatively easily be expanded to further datasets that use the Linked Open data standard. While DBpedia covers a lot of domains, further additions can be made to fill in gaps of knowledge.

While DBpedia offers a public SPARQL endpoint, it is subject to rates and limitations[12], making it impossible to run multiple extractions at once or executing complicated queries. For this reason, and to reduce latency inherent to HTTP, we opted to creating a local database using the DBpedia latest core release[13]. This database dump is the small subset of the total DBpedia releases that are loaded into the main DBpedia SPARQL endpoint. It contains factual data from articles and infoboxes of the English Wikipedia and is enriched with labels, abstracts and links to several ontologies. Our local clone utilizes Blazegraph[14], a Java-based high-performance graph database delivering similar speeds as the DBpedia SPARQL endpoint, yet much higher reliability.

Our system's architecture is designed to connect with multiple data sources simultaneously. This means it can pull data from both the public DBpedia endpoint and our local DBpedia dump concurrently. This greatly accelerates the process of information retrieval, optimizing the efficiency and responsiveness of the recommendation system.

5.1.2 Mapping User Input to the Linked Data Cloud

Since our system is general, we cannot expect the user to provide the URI to a LOD entity, but rather the name (or label) of a thing. This could be the name of a movie, song, artist, place or even person. For an improved user experience, our system should be able to automatically map simple user input to the corresponding LOD entity if it exists. This first step in our workflow will allow us to retrieve all necessary data required by other components of the system. As discussed in Section 4.3, we will be developing our own mapping system using the DBpedia Lookup service[15].

Our first implementation of a mapping algorithm simply uses the DBpedia Lookup service with the input and retrieves the first result. As can be seen in our evaluation in Section 6.2.1, this performed very poorly. We iteratively added further components to the algorithm to improve its accuracy. We based our improvements and subsequent evaluation on the assumption that the input is always the label of an entity, more details are provided in Section 6.2.1. The full workflow of the algorithm can be found in Figure 5.4 and its pseudocode is provided in Algorithm 1. The workflow provides a visual overview of the algorithm, while the pseudocode is more detailed.

Our first improvement over the baseline was an adjustment of the DBpedia Lookup query parameters. Since we assume the input to be very close to, or equal to, the searched entities label, we boost the search results that are an exact match with the input. We set `exactMatchBoost` to 100 and prioritized searching exact matches on the "label" field using the "Exact" parameter. In case the query with these parameters does not result in anything, we use two fallback queries that increasingly lower the search requirements. The last search is broad (on all entity fields index by the service) and does not have the "Exact" parameter. The algorithm stops if this search fails in returning anything as well. These searches can be found in the lines 2 to 8 of Algorithm 1.

[12]https://www.dbpedia.org/resources/sparql/
[13]https://www.dbpedia.org/resources/latest-core/
[14]https://blazegraph.com/
[15]https://lookup.dbpedia.org/index.html

Algorithm 1 Pseudocode of our Mapping Algorithm

```
1:  Input: userInput
2:  searchResults ← DBLookup(term: userInput, field: "label", exact=True)
3:  if searchResults are empty then
4:      searchResults ← DBLookup(term: userInput, field: "all", exact=True)
5:      if searchResults are empty then
6:          searchResults ← DBLookup(term: userInput, field: "all", exact=False)
7:          if searchResults are empty then
8:              return None
9:  for each result in searchResults do
10:     if userInput matches label of result then
11:         if result has attribute dbo:wikiPageRedirects then
12:             redirectTarget ← object of dbo:wikiPageRedirects
13:             return redirectTarget
14:         else
15:             return result
16: nonMatching ← []
17: for each result in searchResults do
18:     if userInput is not a substring of label of result then
19:         nonMatching ← nonMatching+[result]
20: if length of nonMatching ≠ length of searchResults then
21:     searchResults ← searchResults − nonMatching
22: bestMatch ← null
23: for each result in searchResults do
24:     if userInput contains multiple words then
25:         update bestMatch with the result whose label shares the most words with userInput
26:     else
27:         update bestMatch with the result whose label is closest in length to userInput
28: if bestMatch has attribute dbo:wikiPageRedirects then
29:     redirectTarget ← object of dbo:wikiPageRedirects
30:     return redirectTarget
31: else
32:     return bestMatch
```

Furthermore, we noticed that the first result returned by the service is not necessarily the closest match. For example, when searching for "Toy Story", the first result is the entity with label "List of Toy Story characters"[16], whereas the entity with label "Toy Story"[17] is only third in place. To solve this problem, the algorithm first retrieves n results (10 in our evaluation), and attempts to infer the best match. First, we return any entities whose label is an exact match with the input query (lines 9 to 15 of the pseudocode). If no such match exists, we attempt to match entities that include the input query as a substring of their label and exclude the rest (lines 16 to 19 of the pseudocode). Finally, we select the result that most closely matches the input, either in terms of length or in terms of common words, depending on the input (lines 22 to 27 of the pseudocode). We added a post-processing step that determines whether the selected entity redirects to another (if it has a `dbo:wikiPageRedirects` attribute), in which case we use the targeted item (lines 28 to 32 of the pseudocode.

In order to improve on the issue of ambiguity between entities, we assume that the user could provide further context to the search. We devised a second version of our algorithm that additionally filters the search results by type. While the DBpedia Lookup service supports matching types, we have found that this field is not indexed for a lot of items, rendering it unusable. Instead, we slightly adjusted the Algorithm 1 to retrieve the type of the results as described in Section 5.1.3. The workflow and pseudocode for this second algorithm can be found in the Appendix under Figure C.1 and Algorithm 4.

Note that while this system has been implemented using DBpedia Lookup, it could be expanded to work on other LOD sources (such as Wikidata). While DBpedia Lookup offers convenience and speed by indexing certain fields, its working could be easily replicated using SPARQL queries. Also note that the mapping algorithm may reuse previous searches it saved in cache for improved efficiency on future queries.

[16]DBpedia: List_of_Toy_Story_characters
[17]DBpedia: Toy_Story

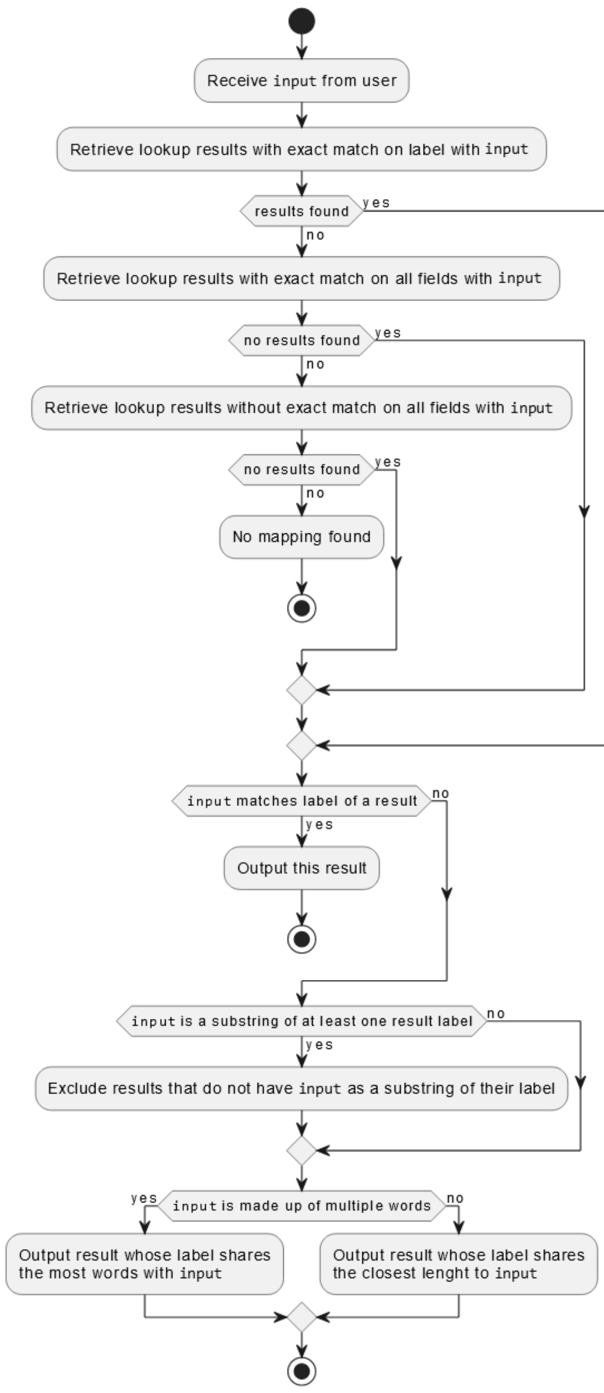

Receive input from user
Retrieve lookup results with exact match on label with input
results found
yes
no
Retrieve lookup results with exact match on all fields with input
no results found
yes
no
Retrieve lookup results without exact match on all fields with input
no results found
yes
no
No mapping found
input matches label of a result
no
yes
Output this result
input is a substring of at least one result label
no
yes
Exclude results that do not have input as a substring of their label
yes
input is made up of multiple words
no
Output result whose label shares the most words with input
Output result whose label shares the closest lenght to input

Figure 5.4: The workflow of our mapping algorithm

5.1.3 Item Type and Attributes Retrieval

After mapping the user's input to a DBpedia item, the system retrieves the corresponding attributes that will be used by subsequent parts of the recommendation pipeline (see Figure 5.2). This is easily done using SPARQL *DESCRIBE* queries against our data sources:

$$DESCRIBE < resource\_uri >$$

This returns a single result RDF graph containing RDF data about the requested resource. Unfortunately, retrieving an item's type is not as straightforward. The item type is commonly defined using the *rdf:type* predicate, but due to the interconnected nature of the LOD cloud, one item might have many types. As an example, the DBpedia resource corresponding to the 1995 film "Toy Story"[18] is linked to 50 other resources via the *rdf:type* predicate. Similar attributes, such as *dcterms:subject* are similarly unspecific. Different types may semantically overlap, such as *schema:Movie* and *dbo:Film*. Some may be more specific than others, a movie may for example have types *schema:CreativeWork*, *dbo:Work* and *owl:Thing*. Finally, many types may be included that are not directly relevant to the item's main characteristics, thus acting as noise in the data. All these reasons make retrieving the most relevant difficult.

One possible approach is to filter the types to keep only one ontology, such as the DBpedia ontology *dbo*. In our "Toy Story" example, this leaves *dbo:Work* and *dbo:Film*. We can then select the most specific type by using the class hierarchy of the ontology. Here, *dbo:Film* is a subclass of *dbo:Work*, meaning we have narrowed our search down to the most relevant type. We have designed a SPARQL query to achieve this, provided in Listing 5.1.

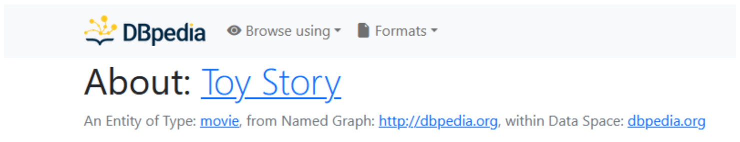
Listing 5.1: A SPARQL query to find the most specific type of *<uri>* in the DBpedia ontology

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?mostSpecificType WHERE {
  <uri> rdf:type ?type .

  FILTER(strstarts(str(?type), str(dbo:)))

  OPTIONAL {
    ?moreSpecificType rdfs:subClassOf+ ?type .
    <uri> rdf:type ?moreSpecificType .
    FILTER(strstarts(str(?moreSpecificType), str(dbo:)))
  }

  BIND(coalesce(?moreSpecificType, ?type) AS ?mostSpecificType)
}
GROUP BY ?mostSpecificType
```

A limitation of this query is that it restricts the item type to one ontology. From our evaluation (see Section 6.2.2), most items have a type in the DBpedia ontology, but not all of them. The underlying system of DBpedia also appears to derive specific types on its own. Browsing any DBpedia page reveals a header at the top of the page with basic information on the provenance and type of the item. For example, https://dbpedia.org/page/Toy_Story reads "An Entity of Type: movie" at the top of the page, as can be seen in the screenshot of the DBpedia page for "Toy Story" in Figure 5.5. This derived type is unfortunately not a part of the RDF triples of the item, and does not seem accessible outside the DBpedia HTML pages. We can retrieve this DBpedia derived type by scraping the item's corresponding page, though that is not a very clean solution.

The same process is applied to other items required in subsequent steps (see Section 5.2.3 on candidate selection). After retrieval, the system caches the retrieved type in a local knowledge base to accelerate future queries using the same input. The retrieved attributes are used for feature selection (see Section 5.2.1).

[18] https://dbpedia.org/page/Toy_Story



About: Toy Story

An Entity of Type: movie, from Named Graph: http://dbpedia.org, within Data Space: dbpedia.org

Figure 5.5: The header of the DBpedia page for “Toy Story”

5.2 Data processing

5.2.1 Schema Summarization

As previously discussed, LOD, and by extension the data our system relies on, is very rich and heterogeneous in nature. This richness should allow our recommendations to be nuanced and personalized, but it also presents challenges since not all attributes are relevant for recommendation depending on domain. For example, items in the ’movie’ domain have attributes such as dbo:runtime or dbp:extra that are less relevant for recommendation tasks than dbo:starring or dbo:director. Schema summarization allows us to rank attributes based on relevance and, as a result, improve our recommendation system’s accuracy and efficiency. In other words, the objective of this step is to identify a subset of a given domain’s available attributes, which retains sufficient information about every element in said domain [106].

As discussed in Section 4.4, we use the vocabulary-oriented summarization framework ABSTAT [106] for ad-hoc schema summarization in our system. ABSTAT takes as input a linked dataset (a subset of items of a given domain), and optionally one or more ontologies that are used in the dataset. After retrieving an item’s type as discussed in Section 5.1.3, we can use it to automatically retrieve further items pertaining to the same domain. By the same process, we can identify ontologies used by those items (such as `dbo`).

ABSTAT outputs a summary consisting of so-called *minimal type patterns* of the form (C, P, D), where C and D are types (concepts or datatypes) and P is an RDF property. These patterns imply a link between some instance of source type C to some instance of target type D through the property P [106]. For example, a minimal type pattern `(dbo:film, dbo:starring, dbo:actor)` implies that instances of type `dbo:film` are linked to instances of type `dbo:actor` by the property `dbo:starring`. These patterns are generated based on explicit or implicit typing assertions found in the input dataset. Every assertion results in one or more patterns, such that the resulting summaries are comprehensive for all relational assertions. If a given assertion results in multiple patterns due to several possible typing assertions, ABSTAT will try to select the most specific one, similarly to our type retrieval described in Section 5.1.3. As type patterns inevitably re-emerge over the course of the analysis, summaries also include the frequency of each pattern in the dataset. This finally allows ABSTAT to rank the resulting patterns according to their relevance in the input dataset.

[106, 87] additionally provide two user interfaces (ABSTATBrowse[19] and ABSTATSearch[20], and a SPARQL endpoint[21]) which enable easy retrieval and searching of existing summaries. Unfortunately, the site and associated endpoint no longer seem accessible at the time of writing.

In this work, we do not implement and evaluate ABSTAT on our own and will make use of the results of [93]. The proposed evaluation mirrors our own datasets and use case (recommendation), we can thus use their work for our purposes. The proposed implementation of ABSTAT is a top-k property feature selection defined as follows: for an input type C such as `dbo:film`, all patterns having C as the source type are selected and arranged according to frequency. Subsequently, the top-k distinct properties are extracted from this ordered list and data type properties are omitted. In their evaluation, the authors make use of a DBpedia dump that focuses on the relational content and does not include annotation properties used to link entities to external resources. Table 5.2 shows the result of their evaluation on the `dbo:film` linked dataset.

[19] http://abstat.disco.unimib.it
[20] http://abstat.disco.unimib.it/search
[21] http://abstat.disco.unimib.it/sparql

Property
dbo:starring
dbo:director
dbo:writer
dbo:producer
dbo:musicComposer
dbp:music
dbo:distributor
dbo:language
dbo:cinematography
dbo:country
dbo:editing
dbp:studio
dbp:extra
dbp:screenplay
dbp:genre

Table 5.2: Top 15 features selected by ABSTAT for the `dbo:film` Domain [93]

5.2.2 Generating Vector Representations

Having mapped and retrieved items from the LOD cloud, we need to create embeddings for each in order to compare them in our recommendation system. As established in Section 4.5, we will be using RDF2Vec for this task. To do so, we use the Python implementation "pyRDF2Vec" [22], which supports creating embeddings from RDF stores directly, whether they are remote or local. As explained in Section 5.1.1, our system utilizes a local DBpedia clone as well as querying the public DBpedia SPARQL endpoint, meaning we can parallelize the execution of the walk generation and embedding tasks for a faster result. To prevent having to query the same nodes multiply times, we employ a shared Least Recently Used (LRU) cache[23] which all threads read and write to as they retrieve data. Using this cache, each thread is able to re-use the collective history of queries and associated results to accelerate the processing. Once the cache is full, the least recently used results are replaced by new data first, as indicated by the name.

Walk generation. The algorithm first takes an item URI as input, and retrieves its attributes. Of these, only entities (called object properties) are considered, all other attributes such as relations and values (literals), are ignored. These object properties serve as the first jumping-points to building random walks across the LOD graph. The combination of these series of random jumps constitutes sentences made up of item "words", which can be fed to word embedding algorithms like Word2Vec [66]. Given sufficient walks of a sufficient length, the embedding algorithm can learn the semantic context of an item, and capture it in a continuous vector space. [97] utilized 500 random walks of depth 4 per item for the best results. Due to limited computational resources, we used 200 walks of depth 4 per item, which achieved good performance in the evaluation performed in [97]. RDF2Vec also supports reverse walks, by including the parents of the entities in the walks. From a base 200 walks, we effectively extract 40,000 walks per item. This allows Word2Vec to have a better learning window for an entity based on its parents and children, and thus predict test data with better accuracy.

Note that we have to use random walks, since generating all possible walks for all attributes results in a very large number of walks. This would make the training of Word2Vec highly inefficient. In [97], each vertex of the graph (attribute linking two entities in the LOD graph), is used to generate only a subset of all possible walks. To generate the walks, the outgoing edge to follow from the currently observed vertex is selected based on the edge weight. In other words, the probability for selecting an edge is based on the set of all outgoing edges from the current vertex. While there are many possibilities to set the weight of the edges (sampling strategies), [97] only

[22] https://github.com/IBCNServices/pyRDF2Vec
[23] https://en.wikipedia.org/wiki/Cache_replacement_policies

consider equal weights, all outgoing edges are equally considered. [29] presents many sampling strategies that could improve results. Due to technical limitations, pyRDF2Vec only supports these sampling strategies on knowledge graphs loaded into memory, and not via SPARQL queries. This is why we also use equal weight sampling (simply called "RandomWalker" in the library.)

Feature pre-processing. Due to the random nature of the walks and the unpredictability of the predicates each entity in the LOD cloud has, we cannot specify a specific set of attributes to use for the walk generation. While all walks originate from an item with a known type, subsequent nodes are very likely to have a different domain and attributes. To improve the quality of our embeddings and make our system more scalable, we can instead remove certain predicates for use in the random walks. Some attributes are very specific to each item, such as `owl#differentFrom`, and also likely very sparse. Other properties such as `dbo:wikiPageExternalLink` and `owl#sameAs`, on the other hand, always have different and unique values, which are not very informative in the recommendation task. We have manually selected 19 such attributes, which often occur in DBpedia items. Table B lists all selected predicates.

Embedding with Word2Vec. After the extraction is completed, all extracted walks for all entities are used to create a large corpus of sequences, which is used to fit the underlying Word2Vec neural network. While Word2Vec [66] is a very efficient neural network-based technique for natural language processing (NLP) tasks, it can also be applied to knowledge graphs, as seen in Section 3.1.1. For our application, words are the nodes encountered during the random walk generation. The evaluation of RDF2Vec in [97] has also shown that the Skip-Gram (SG) model outperforms the Continuous Bag of Words (CBOW) for recommendation tasks. We also make use of an SG model with similar parameters for this reason. We used a window size of 5, 10 iterations and 25 negative samples to generate vectors with a size of 200.

The pyRDF2Vec module also supports FastText[24], which is another word embedding algorithm that works on the character n-grams, while Word2Vec works on the word level. This allows it support creating embeddings for entities that are not in the initially provided vocabulary. The main drawback of this algorithm is its high memory usage, which makes it unusable for our application.

Our system cannot generate all embeddings it requires at once, but iteratively, as new input or items are provided by the user. This poses a problem, as Word2Vec requires a large corpus of walks from other entities to find sufficient context for each new entity and to avoid overfitting the model. Luckily, the gensim[25] implementation of Word2Vec supports online training[26], meaning it can efficiently update an existing corpus with a new one, without needing to retrain the whole model every time. In this way, we can iteratively update our system's underlying Skip-Gram model as new data is added and re-use all previously generated walks as context. Additionally, our system caches all generated embeddings for faster processing of future recommendation tasks involving previously seen entities.

5.2.3 Automatic Selection of Candidates for Recommendation

Creating entity embeddings is an expensive task, and we are not able to create embeddings for all entities in the LOD cloud. We roughly estimate that such a task would take two years of continuous processing to create embeddings for the 3.4 million concepts of DBpedia using our hardware and sub-ideal parameters. Entities in LOD are additionally evolving over time, and would need to be updated regularly. For this reason, we need to carefully choose which entities we select as candidates for recommendation, as we require them. Therefore, the objective of this task is as follows: given an input item mapped to a LOD entity x, retrieve the top n entities from the LOD cloud that are most likely to be similar to the input entity after creating embeddings for them. We define the similarity between entities below.

To help guide this selection process, we can make use of the attributes of the LOD entities, as discussed in Section 5.1.3. A simple heuristic for candidate selection using LOD attributes is item relatedness. We can say that two entities are related in the LOD cloud if they are directly connected by certain attributes. Items that are related to an input entity are linked using attributes, which can be retrieved with a simple SPARQL query:

[24] https://fasttext.cc/
[25] https://github.com/RaRe-Technologies/gensim
[26] https://github.com/RaRe-Technologies/gensim/releases/tag/0.13.3

Listing 5.2: A SPARQL query to find items related to *<uri>* in the DBpedia ontology

```
SELECT DISTINCT ?related WHERE {
    { ?related ?p1 <uri> } UNION
    { <uri> ?p2 ?related . FILTER(!isLiteral(?related)) }
} LIMIT <limit> OFFSET <offset>
```

Similarity, on the other hand, measures how alike two entities are, and takes into account a narrower range of relations than relatedness. For example, “Lion” and “Tiger” have high similarity and relatedness score because they are both large carnivorous cats. On the other hand, “Lion” and “Savannah” are not similar at all, but are highly related, as lions typically inhabit savannas. “Tiger” and “Savannah” are not similar at all and have a lower relatedness value compared to the first pair of entities because tigers do not typically inhabit savannas; they are more commonly found in forests and grasslands.

While a recommendation system based on entity relatedness might provide interesting results, the general use-case for a recommender system is finding similar items. Therefore, we have devised a method for finding items that are similar to a given entity, based on the attributes they share. For more efficiency and accuracy, we only consider the attributes selected by the schema summarization step described in Section 5.2.1. These type-specific features carry the most significant information about the entities in the domain. Through the selection process, we also know that they are the least sparse features of the domain, meaning we can safely rely on them to match candidates. Finally, the output of the schema summarization step is a ranked list based on the importance of each feature. We can use that information to choose candidates more accurately. For example, in the `dbo:film` domain, the `dbo:starring` attribute is more important than `dbp:genre`, which, rather surprisingly, is missing for many entities in the domain. Using the ranked list, we prioritize candidates that match the values in the `dbo:starring` field over candidates that match the values from `dbp:genre`.

We first investigated a solution using a single SPARQL query that specifies all searched attributes and attempts to match as many as possible:

Listing 5.3: SPARQL query to find entities that share the most attributes with *<uri>* in the DBpedia ontology

```
SELECT ?item (COUNT(?attr) AS ?score)
WHERE {
  ?item a <entity_type> .

  OPTIONAL {
    ?item <attribute_1> ?attr1 .
    FILTER(?attr1 IN (<value1>, <value2>))
    BIND("attribute_1" AS ?attr)
  }

  OPTIONAL {
    ?item <attribute_2> ?attr2 .
    FILTER(?attr2 = <value3>)
    BIND("attribute_2" AS ?attr)
  }

  OPTIONAL {
    ?item <attribute_3> ?attr3 .
    FILTER(?attr3 = "value4")
    BIND("attribute_3" AS ?attr)
  }

  ...
  FILTER(?item != <uri>)
}
GROUP BY ?item
ORDER BY DESC(?score)
LIMIT 5
```

In this query, *uri* is the entity for which we are trying to find candidates. While it is not necessary, we specify a type filter in the query to improve the efficiency of the query. To build the rest of the query, we have retrieved the values of the selected attributes for *uri*. Depending on the amount and type of values (i.e. object properties or literals), we specify `FILTER()` statements for each attribute *attribute_i* which restrict their values to the ones we retrieved from the entity *uri*. If an attribute has multiple values defined, we restrict the results to any that match the list using the `IN` operator. Otherwise, we only match results containing the exact object property or literal. The `OPTIONAL` clause ensures that the query does not fail if the `FILTER()` it encompasses cannot match the specified value(s). Finally, we sum up all found attributes by using `BIND()` and `COUNT()` operators, resulting in the score for a given item. We sort the retrieved items in descending order of their scores, which results in a ranked list of candidates.

We have set up a simple experiment using the 1989 movie "*Batman*"[27], and the `dbo:starring` and `dbo:director` fields. The resulting query searches for other movies that star at least one actor of the movie (for instance, Jack Nicholson or Micheal Keaton) and share the same director (Tim Burton). We limited the number of returned results to 10. The full query can be found in Appendix D and the results are listed in Table 5.3.

Table 5.3: Top 5 results of the similarity search SPARQL query on "*Batman (1989)*"

Retrieved Entities	**Score**
dbr:Batman_Returns	3
dbr:Batman_&_Robin_(film)	2
dbr:Batman_Forever	2
dbr:Alice_in_Wonderland_(2010_film)	1
dbr:Big_Eyes	1

The results of this simple experiment prove that the query works and would, given more attributes, retrieve more high scoring candidate entities. Unfortunately, due to the complexity of optional clauses, filters, and the `COUNT()` aggregate function, the query executes very slowly. The public DBpedia SPARQL endpoint does not run it at all due to not being able to optimize it. On our own DBpedia clone, the query took 4 min, 56 sec to complete, which is too slow for one entity using only two attributes. We thus need to reduce the complexity of the SPARQL query by splitting it into smaller queries that each handle a subset of the attributes.

Considering that the m input attributes and their associated data values result in 2^m subsets, we are not able to query each of them. The problem that we are describing is similar to the classic "Set Cover" optimization problem [30], but with some variations. In the classic Set Cover problem, the goal is to find the smallest collection of sets whose union equals the universal set. In our case, the goal is to find n items that cover as many attributes as possible from the ranked list of attributes. However, the ranking of attributes, requiring exactly n items, and needing the items to be as specific as possible introduce additional complexity. We will investigate two commonly used search optimization algorithms: binary search [72] and divide-and-conquer [91].

The first optimization approach we investigated, using binary search, relies on the attribute ranking returned by the feature selection step described in Section 5.2.1. Assuming the ranking is accurate, it allows us to adopt a heuristic-based approach by choosing attribute subsets based on the notion that the attributes that are higher up in the ranking carry more significant semantic information about the entities in question. The idea is to only consider m subsets, going from most specific to least specific. For a given subset s_i with $i \in [0, m]$, we select the top i attributes from the ranked list, as we consider them the most valuable for determining similarity between

[27] http://dbpedia.org/resource/Batman_(1989_film)

entities. We use these subsets in queries that attempt to match all entities that match all specified attribute values. A query using a less specific subset, i.e., using fewer attributes, is more likely to return results than a query using more attributes, i.e., that is more specific. On the other hand, the results of a less specific query are less significant than those returned by a more specific query, since it matches more attributes.

This heuristic assumes that entities that match an attribute that is less important in the ranked list, likely also matches the attributes that are deemed "more important." We can make this assumption since the attribute ranking produced by our feature selection method described in Section 5.2.1 is based on the frequency of each attribute in the entities of the target domain. As a trade-off for much improved performance, we cannot cover edge-cases such as entities matching many "less important" attributes but not higher ranking ones.

A simple algorithm could iterate over the m generated subsets and retrieve results for each, then select the top n most specific results as candidates. Considering that querying the LOD cloud is the slowest step in this process, we can make use of binary search to improve on this algorithm. Instead of starting with the most specific query, which is very likely to result in nothing, or with the least specific query, which will likely return many unspecific results, we start by using the subset with $\lfloor \frac{m}{2} \rfloor$ attributes. The result of this query then indicates whether to relax the requirements by using fewer attributes if no results were found, or, on the contrary, tighten them if any were found. The objective, and stopping condition, of the search is finding the most specific subset of attributes s_x for which a query can find results, such that the query using the subset s_{x+1} does not return anything. In a worst case scenario, i.e., all subsets result in the same set of results, the search has time complexity $O(\log m)$ in the number of attributes, and thus, attribute subsets. Using the selected subset s_x, we can retrieve the n most specific entities by iteratively querying using less specific attribute subsets, starting with the subset s_{x-1}.

The pseudocode of the candidate selection algorithm using binary search is provided in Algorithm 2. As input, the algorithm receives the URI of the entity candidates are searched for, **`entity_uri`**, its domain **`rdf_type`**, the ranked list of attributes and associated values **`attributes`** and the number of candidates to retrieve **`n`**. The **`low`** and **`high`** parameters serve to window the search, which starts using the subset with half the attributes $\lfloor \frac{\texttt{low+high}}{2} \rfloor$ as described above. The **`Query()`** function on lines 7 and 25 uses the input parameters and selected subset of attributes to construct and execute a simple SPARQL query that searches for **`n`** entities of type **`rdf_type`** which match the provided attributes. Based on the found results, we iteratively adjust the search window until finding the most specific subset that produces results (lines 5 to 18). As described previously, we use this subset to retrieve and finally return all our candidates (lines 21 to 28).

While we expect this algorithm to be fast, the assumption we make to select a low number of attribute subsets might not hold true on real data and certain domains, resulting in poor quality results, i.e., which match few attributes. Thus, we have developed a second algorithm based on the Divide and Conquer method, which considers more subsets. The Divide and Conquer approach is an algorithm design technique where a problem is broken down into smaller sub-problems, which are easier to solve. The solutions to the sub-problems are then combined to solve the original problem. This makes the algorithm more efficient than solving the problem in a straightforward manner.

Our algorithm, described in Algorithm 3, receives the same input as Algorithm 2. It will recursively call an inner **`DivideAndConquer`** function, which splits the attributes into two halves recursively until reaching subsets of 2. With m attributes, the maximum depth of recursion is $\log m$. At each recursion, we retrieve the items matching the current subset. If no results are found, we can stop all searches along that branch of the recursive call tree: if a subset of attributes does not match with any entities, then adding an attribute to the subset, i.e., making it more specific, will not match any new items. Since the number of recursive layers is $O(\log m)$, the total number of **`Query()`** calls is $O(m^2 \log m)$. Finally, we sort the items in ascending number of attributes they match.

Note that both of these algorithms evaluate for content-based similarity, whereas the final recommendation algorithm would rank items based on a user's history as well (see Section 5.2.4). In a real application, this candidate selection algorithm would be executed for every item the user interacts with, thus building an unbiased (at least not by the user's preferences) foundation for recommendations based on the user's specific preference. Additionally, we can cache the results of this algorithm and make use of them for future queries that might involve the same input or one of the candidates as input.

Algorithm 2 The pseudocode of our “Binary-Search” candidate selection algorithm

```
 1: Input: rdf_type, entity_uri, attributes, n
 2: low ← 0, high ← length(attributes)
 3: result_uris ← empty list
 4: most_specific_subset_i ← high
 5: while True do
 6:     mid ← $\lfloor \frac{low+high}{2} \rfloor$
 7:     result ← QUERY(()rdf_type, entity_uri, attributes[:mid], n)
 8:     result_uris[mid] ← result
 9:     if length(result) == 0 then
10:         if mid − 1 ∈ result_uri and result_uris[mid − 1] > 0 then
11:             most_specific_subset_i ← mid − 1
12:             break
13:         high ← mid
14:     else
15:         if mid + 1 ∈ result_uri and result_uris[mid + 1] == 0 then
16:             most_specific_subset_i ← mid + 1
17:             break
18:         low ← mid
19: best_candidates ← []
20: i ← most_specific_subset_i
21: while i > 0 and length(best_candidates) < n do
22:     if i ∈ result_uris then
23:         result ← result_uris[i]
24:     else
25:         result ← QUERY(()rdf_type, entity_uri, attributes[:i], n)
26:     for uri in result do best_candidates ← best_candidates + [uri]
27:     i ← i-1
28: return best_candidates[:n]
```

Algorithm 3 The pseudocode of our "Divide-and-Conquer" candidate selection algorithm

```
1: Input: rdf_type, entity_uri, ranked_attributes, n
2: all_items ← empty dictionary
3: procedure DivideAndConquer(attributes)
4:     if length(attributes) < 2 then
5:         return -1
6:     mid ← length(attributes) // 2
7:     left ← attributes[:mid]
8:     right ← attributes[mid:]
9:     found ← DivideAndConquer(left)
10:    found ← found + DivideAndConquer(right)
11:    if found == 0 then
12:        return 0
13:    items ← Query(rdf_type, entity_uri, attributes, n)
14:    for item in items do
15:        all_items[item] ← all_items[item] + attributes
16:    return length of items
17: DivideAndConquer(ranked_attributes)
18: sorted_items ← sort all_items by number of matching attributes
19: return sorted_items[:n]
```

5.2.4 Item-based Recommendation using Vector Representations

Before generating recommendations, the system has mapped (5.1.2) and retrieved attributes (5.1.3) related to an input entity, selected the most relevant features to the target domain (5.2.1), used those features to identify potential candidates from the LOD cloud (5.2.3) and generated embeddings for each (5.2.2). Finally, we use the gathered data to generate recommendations. The task at hand is the top-N recommendation task, in which we generate a ranked list of N recommendations based on a given user's history of ratings.

We investigate using an item-based K-Nearest Neighbor approach [94] with cosine similarity in our system. This approach assesses the similarity between items by calculating the cosine similarity of their respective feature vectors. It then identifies a selection of these item "neighbors" for each individual item. For our system and its evaluation, we used $N = 5$. These neighbors are subsequently employed to estimate a score $r^*(u, i)$ that indicates how likely the user u is to like the item i. We have:

$$r^*(u,i) = \frac{\sum_{j \in \text{history}(u)} \text{cosineSim}(i,j) \cdot r_{u,j}}{\sum_{j \in \text{history}(u)} |\text{cosineSim}(i,j)|}$$

where history(u) is the set of ratings $r_{u,j}$ given by the user u for item j. cosineSim(i, j) is the cosine similarity between the embeddings of items i and j. It is defined as:

$$\text{cosineSim}(i,j) = \frac{i \cdot j}{\|i\|\|j\|}$$

Chapter 6

Experimental Evaluation

We have assessed the performance of our system using experimental evaluation of the automatic mapping (Section 5.1.2), type retrieval (Section 5.1), candidates for recommendation selection (Section 5.2.3), generation of vector representations (Section 5.2.2) and item-based KNN (Section 5.2.4). This chapter contains the setup, evaluation, results and analysis of each experiment we performed.

6.1 Experiments Setup

This section provides information on the datasets used in our evaluations, as well as details on the implementation of the framework. Due to the varying nature of the baselines used, we have elected to not include these in this section, but rather along the description of each experiment in Section 6.2.

6.1.1 Implementation Details

All experiments were built using Python3.9 and executed on a DELL PE R730@32 core machine with 256GB memory, running on Scientific Linux 7.9. As discussed in Section 5.1.1, we utilize DBpedia as a source of background knowledge for all components of our system. All parameters used can be found in Chapter 5 or along the experiments descriptions in Section 6.2.

All code can be found on GitHub[1]. Due to file size limits the data used to generate the results could not be included.

6.1.2 Datasets

To evaluate each of the components in our recommendation framework, we chose two well known datasets used to evaluate recommender systems. These are hetrec2011-Last.FM-2k [20] (which we will denote as the **Last.FM** dataset) and Movielens1M [20].

The MovieLens1M dataset is published by GrouLens[2] and built using 1 million ratings from 6000 users on 4000 movies taken from their MovieLens[3] movie recommendation system. Since its release in 2003, the dataset (and its variants) have been widely used in education, research, and industry [53].

The Last.FM dataset we use was released as part of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011) and contains 92,800 artist listening records from 1892 users

[1]https://github.com/Aed3r/UniversalRS
[2]https://grouplens.org/
[3]https://movielens.org/

Table 6.1: The data coverage for the Last.FM and Movielens1M datasets

Dataset	Identified Items	Total Items
Last.FM	10,180	17,632
Movielens1M	3,300	3,883

on 17632 artists. While the MovieLens1M dataset is based off explicit ratings (on a scale from 1 to 5), the Last.FM dataset is based on implicit feedback given by users when they, e.g., listen to the song of a given artist. The dataset has also become widely used in research and is still being used as a baseline for recommendation systems [97].

6.2 Evaluation

6.2.1 Mapping Algorithms

Setup. The objective of this evaluation is to compare the performance of three mapping algorithms against existing mappings. We used existing mappings to DBpedia resources of items in the MovieLens1M and Last.FM datasets. A mapping for the LibraryThing dataset also exists, but the dataset itself is unfortunately no longer available, which is why we decided to disregard it for this and further experiments.

All mappings we used are available on GitHub[4]. They were semi-manually created by Di Noia et al. [39, 34] using DBpedia Lookup. The authors applied filters, resolved redirects and finally checked the mappings manually. Table 6.1 provides statistics on the final mappings. "Total Items" represents the total number of items in each dataset, whereas "Identified Items" represents the number of mappings [39, 34] found for these. While approximately 84% of the items in the MovieLens1M dataset have mappings, only approximately 58% of artists in the Last.FM dataset have corresponding mappings. This is primarily due to missing coverage on DBpedia. Also note that we have observed that not all mappings are correct, which could be explained by the evolving nature of LOD. We will nonetheless use these mappings as ground truth for the rest of our experiments.

We define the experiment as follows: for a given dataset D, mapping for that dataset m_D and algorithm A, we use the label of the items present in m_D as input for A. The output of A is either `None` if no mappings could be found, or the URI to a DBpedia entity u. If u exists, we compare it to the URI present in the mapping. We evaluate each algorithm based on a simple accuracy metric. Additionally, we keep track of the percentage of items for which no mapping could be found, the number of items that are redirects and the number of ambiguous results returned by DBpedia Lookup.

Simple mapping algorithm. The first algorithm we tested is a simple algorithm that retrieves the first result returned by DBpedia Lookup with default parameters. We call it the "**Simple**" mapping algorithm, which will serve as a baseline for further experiments.

'WithTitleProc' mapping algorithm. After our initial experiment, we found that many items in the MovieLens1M dataset have labels of the following form:

- 'Title (YEAR)'
- 'Title, The (YEAR)'
- 'Title, A(n) (YEAR)'

Examples of these cases are provided in Table 6.2. In these cases, the labels formatted as 'Title, The' or 'Title, A' refer to 'The Title' or 'A Title'. This was most likely done to keep the items in alphabetical order. Most entities in the LOD cloud do not have this formatting, e.g., the production year in their label, which results in wrongly mapped items when trying to match exact labels.

[4]https://github.com/sisinflab/LODrecsys-datasets/tree/master

Table 6.2: The example of wrong mapping due to the format of the input label

Label Pattern	**Item Label**	**Mapping Result**	**Correct Mapping**
“Title (YEAR)”	Suicide Kings (1997)	dbr:Royal_Navy	dbr:Suicide_Kings
“Title, The (YEAR)”	Rapture, The (1991)	dbr:Eudicots	dbr:Rapture_(1965_film)
“Title, A(n) (YEAR)”	Fistful of Dollars, A (1964)	dbr:Ennio_Morricone	dbr:A_Fistful_of_Dollars

Table 6.3: The experimental results for our mapping algorithms on two datasets

Alg.	Data	C. Map.	Inc. Map.	Miss. Map.	Redir.	Acc. (%)
Simple Algorithm	MovieLens1M	480	2822	0	-	14.54
	Last.FM	6449	3727	4	-	63.35
WithTitleProc	MovieLens1M	1077	2225	0	-	32.60
	Last.FM	-	-	-	-	-
ExactMatch	MovieLens1M	1591	1711	0	36	48.82
	Last.FM	7031	3149	0	134	69.01
TypeFilter	MovieLens1M	2276	1026	0	60	68.80
	Last.FM	8354	1826	0	107	**82.06**

Thus, for the next iteration of our mapping algorithm for the MovieLens1M dataset, we did some pre-processing of the input data. We reordered the title name, and cleaned the ‘(YEAR)’ string after each title name if present. We call this second algorithm “**WithTitleProc**”, which is a slightly improved version using adjusted query parameters as well. These changes were not necessary for the Last.FM dataset.

’ExactMatch’ and ’TypeFilter’ mapping algorithms. The final algorithm using only a label as input is the “**ExactMatch**” algorithm, as presented in Section 5.1.2. We also presented a second version of our mapping algorithm, which takes additional context in the form of a type, which we will call the “**TypeFilter**” algorithm. In the case of the MovieLens1M dataset, we filtered the results using the “*http://dbpedia.org/ontology/Film*” type. For Last.FM, we utilized both “*http://dbpedia.org/ontology/Artist*” and “*http://dbpedia.org/ontology/Band*” since the dataset is composed of both individual artists and bands of musicians.

Results. Table 6.3 holds a summary of our findings. The abbreviations used are the following: **Alg.** - Algorithm, **Data** - Dataset, **C. Map.** - Correct Mappings, **Inc. Map.** - Incorrect Mappings, **Miss. Map.** - Missing Mappings, **Redir.** - Redirects, **Acc.** - Accuracy. Redirects are missing for the Simple and WithTitleProc Algorithms, since neither resolve them (see Section 5.1.2). Additionally, the WithTitleProc Algorithm is only defined for the MovieLens1M dataset, as it serves as an adjusted baseline as explained previously. All algorithms were able to find mappings, except for the Simple Algorithm on the Last.FM dataset. The accuracy metric was calculated using the fraction of correct mappings in the total dataset. We have split further analysis of our mapping algorithms based on the dataset used, i.e., MovieLens1M and Last.FM. Let us start by looking at the results of our algorithms on the MovieLens1M dataset, depicted in Figure 6.1.

Firstly, the low retrieval of correct mappings using the Simple Algorithm can be attributed to the input formatting, as described in Section 6.2.1. This was solved by pre-processing the input in our second algorithm, which doubled its accuracy over the baseline. The “ExactMatch” algorithm, which attempts to match an exact result if it exists and otherwise selects the closest one, further improved retrieval and reaches almost 50% correct mappings. Finally, the “TypeFilter” algorithm leverages the additional input for the highest accuracy, with 68.8%.

Additionally, we have analyzed why the “ExactMatch” and “TypeFilter” algorithms retrieve wrong mappings. As discussed in Section 3.2.3, the primary difficulty for matching entities in LOD is the ambiguity between named entities. Figures 6.3 and 6.3 show the distribution of ambiguous results for incorrect mappings retrieved using DBpedia Lookup. In our case, two search results are ambiguous if both a potential match for the searched term. As defined in Section 5.1.2, a search result is a potential match if it features the searched term as a substring of its label. For example, “Toy Story 2” and “Toy Story 3” are ambiguous results for the search term “Toy Story”.

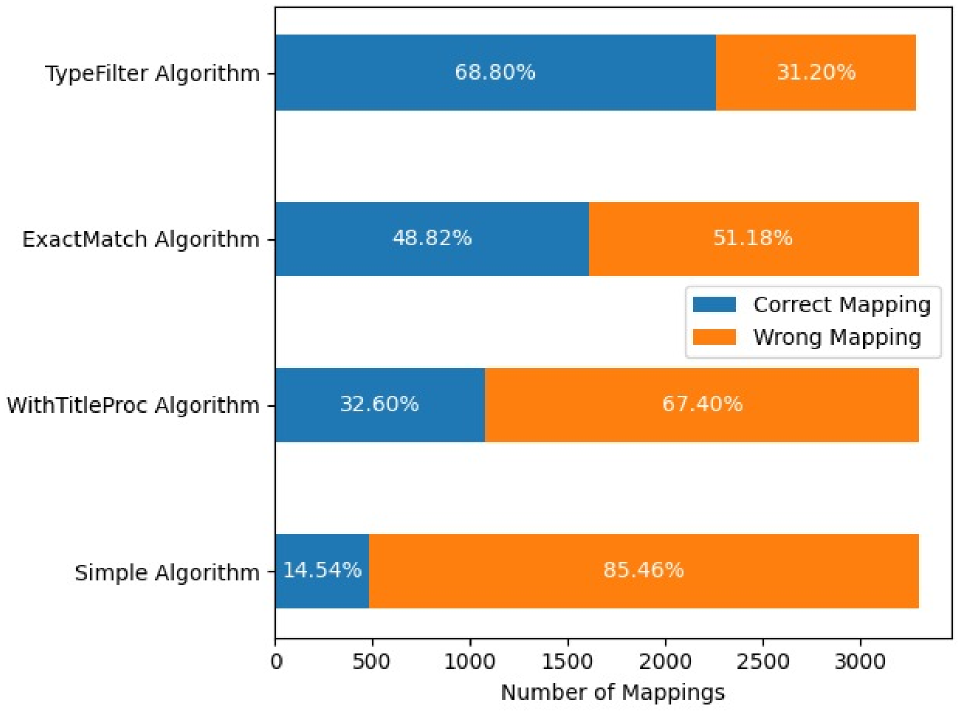


Figure 6.1: The accuracy of our mapping algorithms on the MovieLens1M dataset

We also include the cases where one or no result is matching to draw a comparison.

As shown by Figure 6.2, approximately 91.6% of the incorrect mappings generated by the "ExactMatch" algorithm can be attributed to ambiguous entity labels for the corresponding search. In the remaining 8.4% incorrect mappings, the searched entity was not returned within the first 10 search results by DBpedia Lookup. This is caused by data discrepancies, such as entities having different labels than their name or simply being deleted since the original mapping was created. Without additional context, we are thus not able to improve our results much.

As our "TypeFilter" algorithm shows, giving additional context does improve results considerably. Only 60% of incorrectly mapped items are due to ambiguous results. Due to filtering out many otherwise wrong results, more incorrect mappings are due to one or no matching results.

Let us now take a look at the results of our algorithms on the Last.FM dataset, as depicted in Figure 6.4. Our simple algorithm appears to retrieve a lot more correct mappings than it does for the MovieLens1M dataset. We attribute this to artist and band names being more unique than movie titles, which commonly use existing names. Due to this, the "ExactMatch" algorithm only improves correct retrieval by approximately 5.6%. Finally, filtering by type appears to improve results greatly, reaching 82.06% accuracy. This is corroborated by the distributions of ambiguous results, depicted in Figures 6.5, 6.6 for "ExactMatch" and "TypeFilter" respectively. As observed for the MovieLens1M dataset, filtering by type removes a lot of ambiguity from the results.

This experimental evaluation has shown us that a mapping algorithm using only the label of an entity is possible, though not very accurate. Giving more context to the search by filtering on item type greatly improves the likelihood of finding the right entity, but not sufficiently for a fully unsupervised system. Rather, we believe the output of these algorithms is best used with a user interface that lets the user choose among a selection of options.

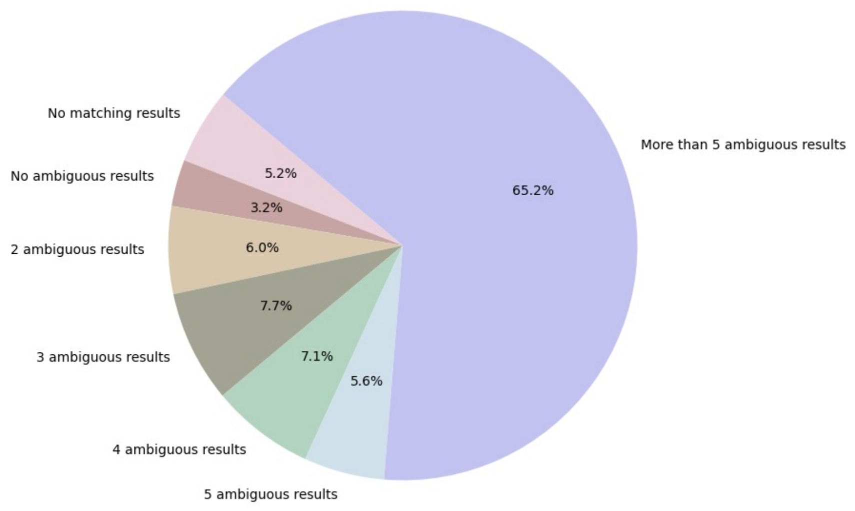


Figure 6.2: The distribution of ambiguous results for incorrect mappings generated by the "ExactMatch" algorithm for the MovieLens1M dataset

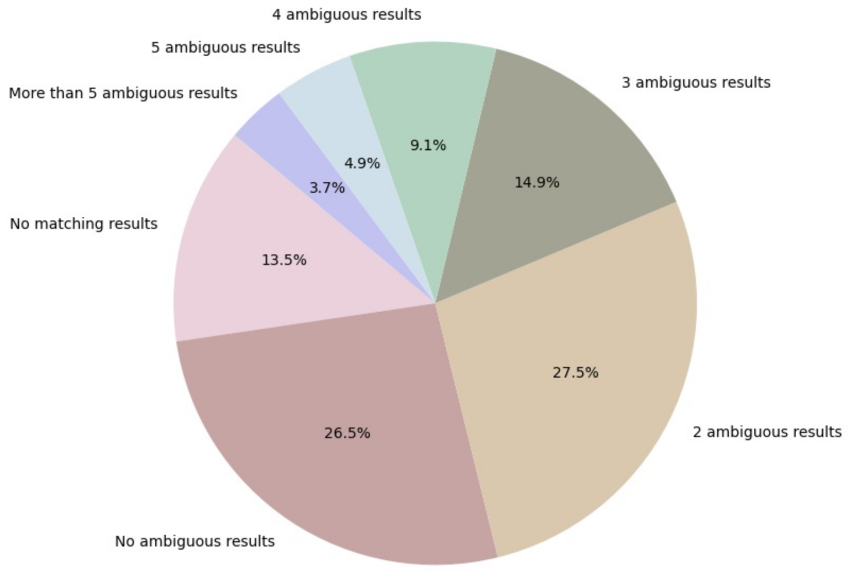


Figure 6.3: The distribution of ambiguous results for incorrect mappings generated by the "TypeFilter" algorithm for the MovieLens1M dataset

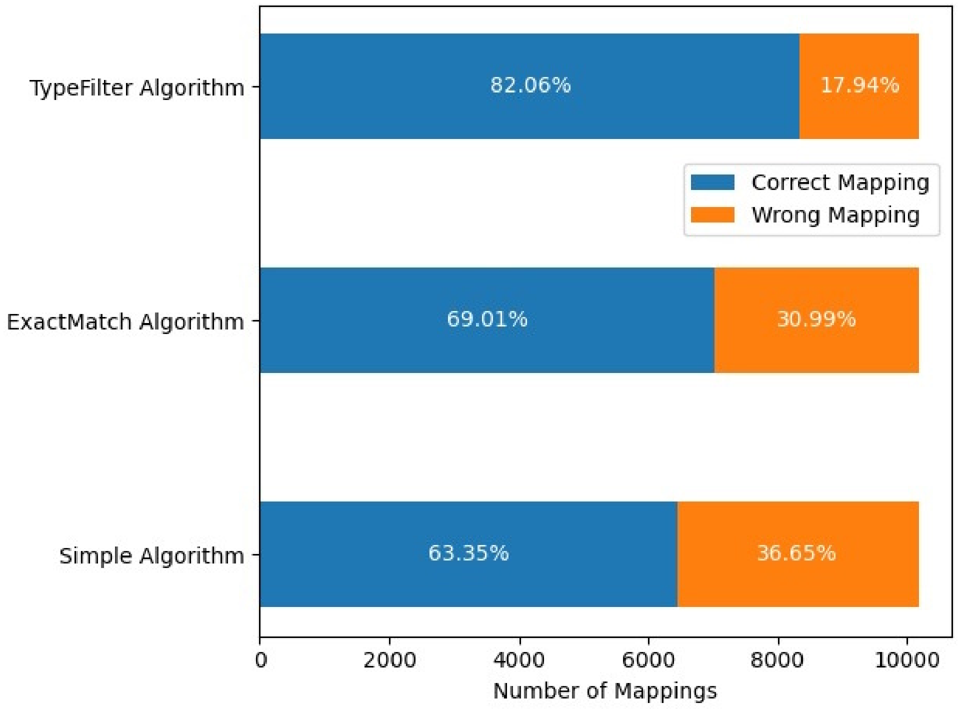


Figure 6.4: The accuracy of our mapping algorithms on the Last.FM dataset

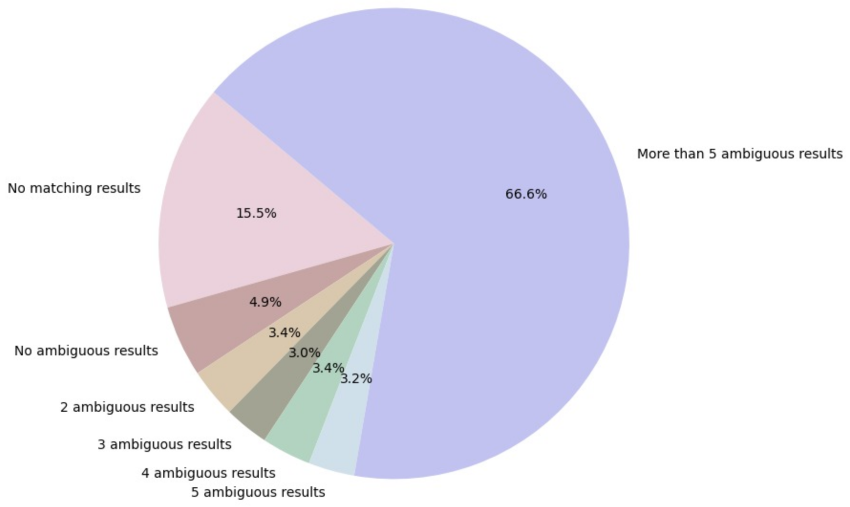


Figure 6.5: The distribution of ambiguous results for incorrect mappings generated by the "ExactMatch" algorithm for the Last.FM dataset

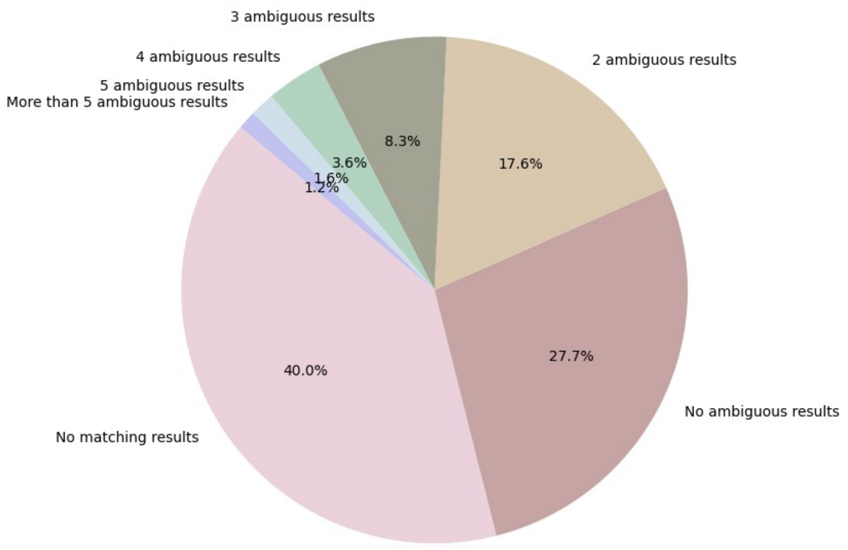


Figure 6.6: The distribution of ambiguous results for incorrect mappings generated by the "TypeFilter" algorithm for the Last.FM dataset

6.2.2 Type Retrieval Algorithm

Setup. Retrieving the specific type of a given LOD entity allows our system to generate more accurate recommendations. It is required for our schema summarization task (Section 5.2.1) which, in turn, allows more accurate selection of candidates for a final recommendation (Section 5.2.3). In this section, we lay out the experimental setup for the evaluation of the two type retrieval algorithms introduced in Section 5.1.3.

The first algorithm, which we will denote by "**SPARQL-based**", relies on the `rdf:type` attributes of entities and attempts to retrieve the most specific option. The second algorithm uses the types inferred by DBpedia, which are only available by scraping the DBpedia webpages. We will denote this algorithm as "**DBpedia-based**" type retrieval.

To test our algorithms, we will make use of the previously mentioned MovieLens1M and Last.FM datasets. For both datasets, we will attempt to retrieve types for each item using the two algorithms. For the first dataset, we expect entities of type "dbo:film[5]" or similar, and for the second we expect "dbo:MusicalArtist[6]" and "dbo:band[7]" or similar. We will evaluate the results based on accuracy.

Results. Table 6.4 is a summary of the types found by both algorithms for the entities of the MovieLens1M dataset, and Table 6.5 is a summary of the Last.FM dataset entity types. For brevity, we have omitted types that appear five times or less, and categorized them under "Minor Related" if they have a tie to the target type, and "Unrelated" otherwise. The abbreviations used are the following: **dbo.** - `http://dbpedia.org/ontology/`, **yago** - `http://dbpedia.org/class/yago/`, **owl** - `http://www.w3.org/2002/07/owl#`, **umbel** - `http://umbel.org/umbel/rc/`.

For both datasets, the DBpedia-based method retrieves a much wider range of types. From the MovieLens1M entities, the DBpedia-based algorithm retrieved 60 different types, compared with only 9 retrieved using the

[5]http://dbpedia.org/ontology/MusicalArtist
[6]http://dbpedia.org/ontology/MusicalArtist
[7]http://dbpedia.org/ontology/Band

Table 6.4: Types retrieved by our type retrieval algorithms for the entities of the MovieLens1M dataset

Types (abbreviated)	**SPARQL-based**	**DBpedia-based**
dbo:Film	3131	2988
dbo:TelevisionShow	9	36
owl:Thing	0	149
umbel:Movie_CW	0	11
Minor Related	8	33
Unrelated	21	79
Failed	137	10

Table 6.5: Types retrieved by our type retrieval algorithms for the entities of the Last.FM dataset

Types (abbreviated)	**SPARQL-based**	**DBpedia-based**
dbo:Band	6412	244
dbo:MusicalArtist	3402	134
dbo:Person	93	46
dbo:Athlete	17	0
dbo:MusicalWork	13	0
dbo:Agent	23	5561
dbo:Animal	0	3777
owl:Thing	0	207
umbel:Band_MusicGroup	0	13
yago:Person100007846	0	43
Minor Related	4	34
Unrelated	45	53
Failed	195	70

SPARQL-based method. Similarly, in the case of the Last.FM dataset, it retrieved 55 different types in contrast to the 21 retrieved by the SPARQL-based method. This has the advantage that the DBpedia-based method less frequently fails to find a type, but if it does, it is less likely to have found the right one. DBpedia type inferring appears to work differently to ours, as it selected `dbo:Agent` and `dbo:Animal` for a very large majority of artists and bands. As a result, only approximately 3.7% of the selected types are accurate, compared to the approximately 96.4% our simple SPARQL-based algorithm finds. The results are closer for entities of the MovieLens1M dataset.

Regarding the SPARQL-based algorithm, the number of missing types for both datasets can be attributed to items that do not have types in the DBpedia Ontology (dbo), in which case nothing is returned. A future version of the algorithm could easily implement fallback ontologies to raise the likelihood of retrieving a type. Despite its lower performance, the DBpedia-based algorithm could also be used as a fallback to further improve retrieval.

This experimental evaluation has shown that the SPARQL-based type retrieval method outperforms using the type inferred by DBpedia. We will thus use this method in our system and for further experiments.

6.2.3 Automatic Candidate Selection Algorithm

As for previous experiments, we make use of the MovieLens1M dataset to evaluate our two candidate selection algorithms: using binary search and divide and conquer. For the attribute selection, we use the attributes found in [93] using ABSTAT and also reported in Section 5.2.1. Due to lacking baselines or benchmarks for this task, we will utilize multiple performance metrics to evaluate the algorithms based on their output:

- (1) Number of successful searches.

Table 6.6: The performance comparison of our candidate selection algorithms on entities of MovieLens1M dataset

Metric	**Algorithm using binary search**	**Algorithm using divide and conquer**
(1) (#)	2926	2946
(2) (#)	375	355
(3) (#)	4.83	110.11
(4) (s)	11.97	88.67
(5) (#)	14.57	15.0
(6) (%)	10.2	35.4
(7) (#)	0.17	0.25

- (2) Number of failed searches.
- (3) Average number of queries necessary to find n candidates.
- (4) Average time needed to find n candidates in seconds.
- (5) Average number of candidates found per item.
- (6) Average percentage of candidate attributes that match with the input entity.
- (7) Average number of candidates also found in the MovieLens1M dataset per item.
- (8) Distribution of the number of matching attributes for all candidates.

The results of our evaluation can be found in Table 6.6, excluding metric (8) which is included separately below. Firstly, the failed searches can be attributed to missing data in the input entities. A manual check reveals that a lot of them redirect to other entities or simply do not exist anymore. Our algorithms need to retrieve the values associated to the entity to match it with others, without which they cannot start the searching process.

While the algorithm using binary search is a lot faster, it also appears to perform less well than the algorithm using divide and conquer. It is able to find n candidates (15 in our case) in almost all cases, but their quality is poor: they only have about 1.5 attributes in common with the input entity.

As a trade-off for speed, the second algorithm gains in accuracy. It can always find n candidates, of which 5.3 attributes match with the input entity on average. These candidates are more likely to be similar to the input entity. Table 6.7 presents an example of the candidates selected by this algorithm for the DBpedia entity of "Toy Story", along with the number of attributes they have in common. Unsurprisingly, the list contains sequels to the movie in question, as well as other well-known animation films from the same studio and/or director.

Figures 6.7 and 6.8 show the distribution of the number of matching attributes for candidates selected by the algorithms using binary search and divided and conquer, respectively. Comparing their results further proves the improved quality of the candidates found by the second algorithm: in no case did less than three attributes of the input entity match, and for 63,2% of all selected candidates five or more attributes matched. In the case of the algorithm using binary search, 80.5% of all selected candidates matched with only attribute of the input entity.

We suspect that the low performance of the algorithm using binary search is due to the assumption that higher ranking attributes are more common, does not work on real data. While the highest ranking attribute (`dbo:starring`) is present in a lot of `dbo:Film` entities, combinations with other attributes are a lot more varied and cannot be captured on a single scale. We will therefore prefer the algorithm using divide and conquer, despite its slower speed.

6.2.4 Recommendation using Item Embeddings

Setup. We will evaluate the quality of our item embeddings generated using RDF2Vec as described in Section 5.2.2 using the item-based KNN recommendation algorithm introduced in Section 5.2.4. As before, we will make use of the MovieLens1M and Last.FM datasets for this experiment. As baseline, we make use of the experimental

Table 6.7: Top 15 candidates selected by the algorithm using divide and conquer using `dbr:Toy Story` as input

Ranking	**Candidate name**	**Number of matching attributes**
1	`dbr:A_Bug's_Life`	9
2	`dbr:Toy_Story_2`	8
3	`dbr:Toy_Story_3`	6
4	`dbr:Quest_for_Camelot`	6
5	`dbr:Cars_(film)`	6
6	`dbr:Cars_2`	6
7	`dbr:Toy_Story_4`	5
8	`dbr:The_Ladykillers_(2004_film)`	5
9	`dbr:Pretty_in_Pink`	5
10	`dbr:Monsters,_Inc.`	5
11	`dbr:Cars_3`	5
12	`dbr:James_and_the_Giant_Peach_(film)`	5
13	`dbr:10_Things_I_Hate_About_You`	5
14	`dbr:Captain_(2018_film)`	5
15	`dbr:Tiny_Toy_Stories`	4

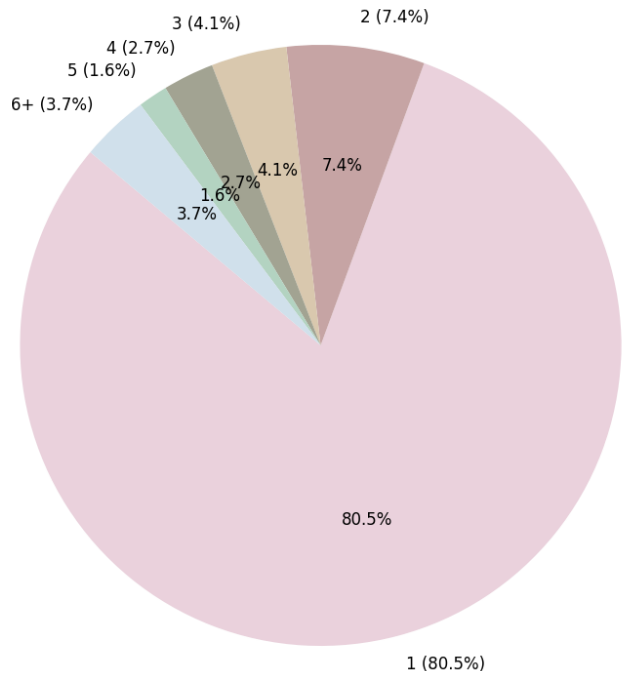


Figure 6.7: The distribution of the number of matching attributes for candidates found by the candidate selection algorithm using binary search

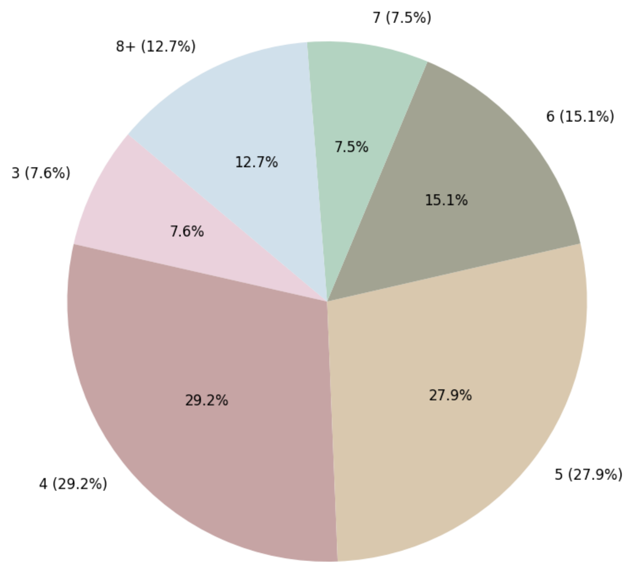


Figure 6.8: The distribution of the number of matching attributes for candidates found by the candidate selection algorithm using divide and conquer

evaluation using a recommender system performed in [97]. In their work, Ristoki et al. introduced RDF2Vec and also made use of an item-based KNN with cosine similarity to evaluate its performance against other vector representation learning methods. We use their results to show that our system achieves comparable performance.

To make the comparison fair, we applied the same evaluation protocol and data pre-processing as [97]. Firstly, to remove popularity biases, we removed the 1% most popular items from our datasets. We based popularity on the total number of ratings or interactions an item received. We removed users with less than 50 ratings in the MovieLens1M dataset, which reduces the sparsity of its ratings with regard to the total number of items. This, in turn, makes the dataset more challenging for our dataset. As for the Last.FM dataset, we removed users that have given less than 5 ratings and items that were given less than 5 ratings. The final statistics of the three datasets are reported in Table 6.8. Finally, we perform a standard random 80/20 training/testing split of both datasets for our evaluation, as [97] does not mention how their split was performed.

Table 6.8: The statistics about the datasets

	MovieLens	**Last.fm**
Number of users	4,193	1,870
Number of items	3,183	2,430
Number of ratings	822,904	44,757
Data sparsity	0.61%	0.0098%

As discussed in Section 5.2.4, our algorithm produces a ranked list of N items for the user. We use $N = 10$ as this is what was also used in [97] and corresponds to a realistic amount of recommendations to present to a user. To evaluate the produced recommendations, we utilize the *all unrated items* methodology that was also used in [97]: instead of only predicting scores for the items that have ratings in the test set, we predict scores for all items that are unrated, i.e., for all items that are not present in the train set. This corresponds to a more realistic setting in which users have rated only a small subset of the available items.

We then compare the top-N ranked recommendation list with the test set to measure their performance using

Table 6.9: The 2x2 confusion matrix that depicts every outcome of our recommendation model

	Predicted: Yes	Predicted: No
Actual rating: Yes	True Positives (TP)	False Negatives (FN)
Actual rating: No	False Positives (FP)	True Negatives (TN)

Table 6.10: The performance comparison of our recommendation algorithm with the RDF2Vec baseline on the MovieLens1M and Last.FM datasets

Algorithm	Dataset	Precision@N	Recall@N	F1@N	nDCG@N
RDF2Vec baseline	MovieLens1M	**0.05423**	0.02693	**0.03598**	**0.31676**
	Last.FM	0.01	0.05498	0.01692	0.02911
Our algorithm	MovieLens1M	0.04328	**0.02724**	0.02864	0.3056
	Last.FM	**0.02067**	**0.1158**	**0.03393**	**0.1909**

four commonly used metrics for recommendation [94]. To calculate these metrics, we first define every outcome of our recommendation model in terms of True Positives (TP), False Negatives (FN), False Positives (FP) and True Negatives (TN) as defined in Table 6.9. This allows us to calculate the following metrics:

- precision@N indicates the fraction of relevant items in the top-N list. It is defined as

$$precision@N = \frac{TP}{TP + FP}$$

- recall@N represents the fraction of relevant items, in the test set, occurring in the top-N recommendations. It is defined as

$$recall@N = \frac{TP}{TP + FN}$$

- The F1 score is the harmonic mean of precision and recall. It is defined as

$$F1@N = \frac{precision@N \times recall@N}{precision@N + recall@N}$$

- The normalized Discounted Cumulative Gain nDCG@N takes into account the position of items in the top-N recommendation list, contrary to the previous three metrics. It is defined as:

$$nDCG@N = \frac{1}{\text{IDCG}} \sum_{i=1}^{N} \frac{2^{\text{rel}(u_i)} - 1}{\log_2(1+i)}$$

  where IDCG is the Ideal Discounted Cumulative Gain, which is the maximum possible DCG up to position N and $rel(u_i)$ represents the binary relevance of the item i to the user u. Due to the *all unrated items* methodology, we need to set a "neutral" value for items without rating to calculate this metric. We use 3 for the MovieLens1M dataset (since the ratings are defined on a 5 point scale, with 1 being the worst rating and 5 the best), and calculate the median of existing interaction scores for the Last.FM dataset (since the interaction scores of the dataset have no defined range and vary between users).

The baseline results in [97] that we use for our comparison can be found along with the results of this experiment below. We refer to the configuration entitled *DB2vec SG 200w 200v 4d*, as it corresponds to our own: using DBpedia to generate 200 walks of depth 4 and embedding them with a Skip-Gram model that has vector size 200. Due to limited computational resources and time, we were not able to perform a grid search to find the optimal hyperparameters for our algorithm, but we show that our algorithm achieves comparable performance to existing systems.

Results. Table 6.10 shows the results of our recommendation algorithm on the MovieLens1M and Last.FM datasets. We have also included the results of [97] and highlighted higher scores to draw out a comparison. Our algorithm performs slightly worse on the MovieLens1M dataset in terms of precision, F1 and nDCG, but achieves a slightly higher recall. These differences can be attributed to slight differences in parameters and dataset used, and prove that our system performs competitively with the baseline on a dataset based on explicit ratings.

In the case of the Last.FM dataset, which is based on implicit feedback by the user, our algorithm performs significantly better than the baseline on all metrics, but especially in terms of precision where we achieve a 206% increase. Since our results do not deviate much from the MovieLens1M dataset, it is difficult to say why the evaluation in [97] performed so poorly.

Chapter 7

Conclusion

7.1 Summary and Discussion

In this thesis, we have identified the gap in research that are general-pupose recommendation systems. Due to industry needs, current research on recommendation systems is typically limited to one domain and supervised. We have shown that certain applications justify using a domain-independent recommender system. Thus, we set out to answer the following main research question and sub-research questions:

Is it feasible to develop a truly domain-independent recommender system to provide recommendations across multiple domains?

RQ1: *What data source can a domain-independent recommender system rely on for background knowledge?*

RQ2: *How can a domain-independent recommender system effectively generate personalized recommendations across multiple domains?*

RQ3: *Which methods can be used to properly evaluate a domain-independent recommender system?*

Firstly, we have established that Linked Open Data (LOD) can be used to build a domain-independent recommender system. LOD holds large amounts of data on many domains, and is openly accessible. The nature of Linked Data additionally makes it very easy and efficient to process, enabling a recommendation system to fully exploit the available data. This answers **RQ1**.

As analyzed in Chapter 4, implemented in Chapter 5 and evaluated in Chapter 6, we were able to devise a flexible framework for domain-independent recommendation. The final system we described and implemented uses the following components in an unsupervised way:

- Mapping of user input to Linked Open Data (LOD) entities: we have devised our own mapping algorithm that can link an input query to a LOD entity. While the performance is not ideal, this component can be used in combination with a user interface to prevent selecting the wrong entity.
- Selection of the most specific type of a LOD entity: here we have also devised our own algorithm, which outperforms the type inference performed by underlying DBpedia systems.
- Schema summarization of LOD domains: this component was shown to outperform similar methods in a related paper, and allows our system to select the most relevant attributes of a given domain.
- Selecting candidate LOD entities based on similarity
- Generation of vector representations of LOD entities: we use the RDF2Vec algorithm to create embeddings of user selected and candidate entities. These can be used in the recommendation algorithm to compare entities.

- Generation of recommendations using an item-based KNN: we utilize a K-Nearest-Neighbor approach that takes into account a user's rating history. Our evaluation has shown that the system performs competitively or even outperforms our baseline.

This answers **RQ2**: the components of our framework work together in an unsupervised way to generate recommendations across multiple domains.

As for **RQ3**, since no previous research on general-purpose recommender systems exists, we did not have access to any baselines or benchmarks to compare our system against. Furthermore, due to technical limitations and time constraints, we did not have the ability to integrate all components into one full system. For this reason, we have chosen to evaluate each component individually. While we cannot prove that the system as a whole works, we can estimate its performance based on its individual parts.

Thus, by answering all sub-research questions, we have shown that a truly domain-independent recommender system that provides recommendations across multiple domains is feasible, which answers our main research question.

7.2 Limitations and Future Work

Due to computational and time constraints, we did not have the ability to integrate all components into one full system. While we cannot show that the domain-independent recommendation system works fully unsupervised, the individual components and their evaluation hint at a full system being possible. Designing a full system using the proposed framework and evaluating it on real data would further prove its proficiency and performance. Different sources of data, mapping, type retrieval, candidate selection, schema summarizers, embedders or recommendation algorithms could also be explored, as they might provide better efficiency or performance. We designed each component of the framework to be flexible in terms of source of data and methods used, and each can be easily replaced.

While the two datasets used in our evaluation showcase the domain-independence of the framework, more experiments using datasets of varying sizes and kind should be done to fully evaluate the range and capability of the system as a whole.

We would also like to point out that using Linked Open Data as a source of background knowledge, though effective, might be prone to errors and bias. The data in these datasets is contributed on a voluntary basis and their validity cannot be guaranteed. Furthermore, the data might subject to the following biases:

- Selection bias: the available data might not reflect the entire population well. Since data collection is expensive and difficult, it might be biased towards, for example, well funded research areas or geographical regions.
- Temporal bias: the data may represent a specific time frame that may be incorrect in the present or future. This is especially applicable to rapidly changing fields.
- Cultural bias: the data can reflect the perspectives and potential prejudices of context in which they are created, whether cultural, religious or political.
- Quality bias: not all data is created with the same quality, but it is hard to distinguish between factually correct and incorrect information, which can affect the analysis of the system built upon it.
- Confirmation bias: users contributing to the Linked Open Data might only link data that confirm their pre-existing beliefs and ignore data that contradicts them.

These problems affect Linked Open Data, and similar collaborative projects, in general and cannot easily be solved. In the context of this work, it is important to keep in mind these limitations and not expect the system to always be perfectly accurate. In a future work, it could be interesting to explore to what extent these biases affect the quality of the recommendation system.

The previously mentioned biases also raise certain ethical considerations of the proposed recommender system. Given the biases that can exist in LOD, there is a risk that the system could perpetuate or even amplify existing inequalities. For instance, if the data is skewed towards a particular demographic, the recommendations could unfairly favor that group. This can be in part alleviated by making the system more transparent, i.e. by allowing the user to verify how certain recommendations are created. This is feasible since the framework is not a black box: all recommendations can be traced back to certain their origin and explained in intuitive ways. In a future work, we could explore making the system explainable, as shown in other works [6, 13, 76].

Another limitation of the system is its speed. Due to the fact that it requires querying for data ad-hoc, a domain-independent recommender system will never achieve the speed and efficiency of a classic recommender system that has access to fixed datasets and data types. In a real application, this system could only be used in an asynchronous way, meaning the recommendations are processed in the background and shown to the user once they are computed. Nevertheless, the current implementation of our system supports caching results, which prevents it from querying and processing the same entities multiple times, as well as parallelization to accelerate data retrieval and processing to some degree. The slowest step of the system is the walk extraction required to create vector representations of LOD items. In a future work, we could explore using different and perhaps more efficient walk generation techniques.

Bibliography

[1] Muyeed Ahmed, Mir Tahsin Imtiaz, and Raiyan Khan. “Movie recommendation system using clustering and pattern recognition network”. In: *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. Jan. 2018, pages 143–147. DOI: 10.1109/CCWC.2018.8301695.

[2] Bushra Alhijawi and Yousef Kilani. “Using genetic algorithms for measuring the similarity values between users in collaborative filtering recommender systems”. In: *International Conference on Interaction Sciences* (June 2016), pages 1–6. DOI: 10.1109/icis.2016.7550751.

[3] Renzo Arturo Alva Principe, Andrea Maurino, Matteo Palmonari, Michele Ciavotta, and Blerina Spahiu. “ABSTAT-HD: a scalable tool for profiling very large knowledge graphs”. en. In: *The VLDB Journal* 31.5 (Sept. 2022), pages 851–876. ISSN: 0949-877X. DOI: 10.1007/s00778-021-00704-2.

[4] Shadi AlZu’bi, Bilal Hawashin, Mohammad EIBes, and Mahmoud Al-Ayyoub. “A Novel Recommender System Based on Apriori Algorithm for Requirements Engineering”. In: *2018 Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. Oct. 2018, pages 323–327. DOI: 10.1109/SNAMS.2018.8554909.

[5] Sana Abida Amin, James Philips, and Nasseh Tabrizi. “Current Trends in Collaborative Filtering Recommendation Systems”. en. In: *Services – SERVICES 2019*. Edited by Yunni Xia and Liang-Jie Zhang. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pages 46–60. ISBN: 978-3-030-23381-5. DOI: 10.1007/978-3-030-23381-5_4.

[6] André Levi Zanon, Leonardo Chaves Dutra da Rocha, and Marcelo Garcia Manzato. “Balancing the trade-off between accuracy and diversity in recommender systems with personalized explanations based on Linked Open Data”. In: *Knowledge Based Systems* (June 2022), pages 109333–109333. DOI: 10.1016/j.knosys.2022.109333.

[7] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. “DBpedia: A Nucleus for a Web of Open Data”. en. In: *The Semantic Web*. Edited by Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pages 722–735. ISBN: 978-3-540-76298-0. DOI: 10.1007/978-3-540-76298-0_52.

[8] Hakan Bagci and Pinar Karagoz. “Context-Aware Friend Recommendation for Location Based Social Networks using Random Walk”. In: *Proceedings of the 25th International Conference Companion on World Wide Web*. WWW ’16 Companion. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, Apr. 2016, pages 531–536. ISBN: 978-1-4503-4144-8. DOI: 10.1145/2872518.2890466.

[9] Shumeet Baluja, Rohan Seth, D. Sivakumar, Yushi Jing, Jay Yagnik, Shankar Kumar, Deepak Ravichandran, and Mohamed Aly. “Video suggestion and discovery for youtube: Taking random walks through the view graph”. In: Apr. 2008, pages 895–904. DOI: 10.1145/1367497.1367618.

[10] Nicola Barbieri, Gianni Costa, Giuseppe Manco, and Riccardo Ortale. “Modeling item selection and relevance for accurate recommendations: a bayesian approach”. In: *Proceedings of the fifth ACM*

conference on Recommender systems. RecSys '11. New York, NY, USA: Association for Computing Machinery, Oct. 2011, pages 21–28. ISBN: 978-1-4503-0683-6. DOI: 10.1145/2043932.2043941.

[11] Javad Basiri, Azadeh Shakery, Behzad Moshiri, and Morteza Zi Hayat. "Alleviating the cold-start problem of recommender systems using a new hybrid approach". In: *2010 5th International Symposium on Telecommunications*. Dec. 2010, pages 962–967. DOI: 10.1109/ISâĎą.2010.5734161.

[12] Punam Bedi, Pooja Vashisth, Purnima Khurana, and Preeti. "Modeling user preferences in a hybrid recommender system using type-2 fuzzy sets". In: *2013 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. July 2013, pages 1–8. DOI: 10.1109/FUZZ-IEEE.2013.6622471.

[13] Vito Bellini, Angelo Schiavone, Tommaso Di Noia, Azzurra Ragone, and Eugenio Di Sciascio. "Knowledge-aware Autoencoders for Explainable Recommender Systems". In: *DLRS@RecSys* (Oct. 2018), pages 24–31. DOI: 10.1145/3270323.3270327.

[14] Shlomo Berkovsky, Tsvi Kuflik, and Francesco Ricci. "Cross-Domain Mediation in Collaborative Filtering". en. In: *User Modeling 2007*. Edited by Cristina Conati, Kathleen McCoy, and Georgios Paliouras. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pages 355–359. ISBN: 978-3-540-73078-1. DOI: 10.1007/978-3-540-73078-1_44.

[15] Alex Beutel, Kenton Murray, Christos Faloutsos, and Alexander Smola. "CoBaFi - Collaborative Bayesian filtering". In: Apr. 2014, pages 97–108. DOI: 10.1145/2566486.2568040.

[16] Derek Bridge, Mehmet Goker, Lorraine Mcginty, and Barry Smyth. "Case-based recommender systems". In: *Knowledge Engineering Review* 20 (Sept. 2005). DOI: 10.1017/S0269888906000567.

[17] Robin Burke. "Knowledge-based recommender systems". In: *Encyclopedia of Library and Information Science* (Jan. 2000).

[18] HongYun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. "A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications". In: *IEEE Transactions on Knowledge and Data Engineering* 30.9 (Sept. 2018), pages 1616–1637. ISSN: 1558-2191. DOI: 10.1109/TKDE.2018.2807452.

[19] Erion Çano and Maurizio Morisio. "Hybrid Recommender Systems: A Systematic Literature Review". In: *Intelligent Data Analysis* 21.6 (Nov. 2017), pages 1487–1524. ISSN: 1088467X, 15714128. DOI: 10.3233/IDA-163209.

[20] Ivan Cantador, Peter Brusilovsky, and Tsvi Kuflik. "Second workshop on information heterogeneity and fusion in recommender systems (HetRec2011)". In: *Proceedings of the fifth ACM conference on Recommender systems*. RecSys '11. New York, NY, USA: Association for Computing Machinery, Oct. 2011, pages 387–388. ISBN: 978-1-4503-0683-6. DOI: 10.1145/2043932.2044016.

[21] Yukuo Cen, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou, and Jie Tang. "Representation Learning for Attributed Multiplex Heterogeneous Network". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, July 2019, pages 1358–1368. DOI: 10.1145/3292500.3330964.

[22] Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. "A Web Service Recommender System Using Vector Space Model and Latent Semantic Indexing". In: *2011 IEEE International Conference on Advanced Information Networking and Applications*. Mar. 2011, pages 602–609. DOI: 10.1109/AINA.2011.99.

[23] Girish Chandrashekar and Ferat Sahin. "A survey on feature selection methods". In: *Computers & Electrical Engineering*. 40th-year commemorative issue 40.1 (Jan. 2014), pages 16–28. ISSN: 0045-7906. DOI: 10.1016/j.compeleceng.2013.11.024.

[24] S. Chatterjee. "An Entity-Oriented Approach for Answering Topical Information Needs". English. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 13186 LNCS (2022), pages 463–472. ISSN: 0302-9743. DOI: 10.1007/978-3-030-99739-7_57.

[25] S. Chatterjee and L. Dietz. "Entity Retrieval Using Fine-Grained Entity Aspects". English. In: 2021, pages 1662–1666. ISBN: 978-1-4503-8037-9. DOI: 10.1145/3404835.3463035.

[26] Jinyin Chen, Yangyang Wu, Lu Fan, Xiang Lin, Haibin Zheng, Shanqing Yu, and Qi Xuan. "N2VSCDNNR: A Local Recommender System Based on Node2vec and Rich Information Network". In: *IEEE Transactions on Computational Social Systems* 6.3 (June 2019), pages 456–466. ISSN: 2329-924X. DOI: 10.1109/TCSS.2019.2906181.

[27] Xiaojun Chen, Shengbin Jia, and Yang Xiang. "A review: Knowledge reasoning over knowledge graph". en. In: *Expert Systems with Applications* 141 (Mar. 2020), page 112948. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2019.112948.

[28] Evangelia Christakopoulou and George Karypis. "Local Latent Space Models for Top-N Recommendation". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '18. New York, NY, USA: Association for Computing Machinery, July 2018, pages 1235–1243. ISBN: 978-1-4503-5552-0. DOI: 10.1145/3219819.3220112.

[29] Michael Cochez, Petar Ristoski, Simone Paolo Ponzetto, and Heiko Paulheim. "Biased graph walks for RDF graph embeddings". In: *Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics*. WIMS '17. New York, NY, USA: Association for Computing Machinery, June 2017, pages 1–12. ISBN: 978-1-4503-5225-3. DOI: 10.1145/3102254.3102279.

[30] Graham Cormode, Howard Karloff, and Anthony Wirth. "Set cover algorithms for very large datasets". In: *Proceedings of the 19th ACM international conference on Information and knowledge management*. CIKM '10. New York, NY, USA: Association for Computing Machinery, Oct. 2010, pages 479–488. ISBN: 978-1-4503-0099-5. DOI: 10.1145/1871437.1871501.

[31] Yashar Deldjoo, Mehdi Elahi, Massimo Quadrana, and Paolo Cremonesi. "Using visual features based on MPEG-7 and deep learning for movie recommendation". en. In: *International Journal of Multimedia Information Retrieval* 7.4 (Nov. 2018), pages 207–219. ISSN: 2192-662X. DOI: 10.1007/s13735-018-0155-1.

[32] Yue Deng. "Recommender Systems Based on Graph Embedding Techniques: A Review". en. In: *IEEE Access* 10 (2022), pages 51587–51633. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3174197.

[33] Tommaso Di Noia, Roberto Mirizzi, Vito Claudio Ostuni, Davide Romito, and Markus Zanker. "Linked open data to support content-based recommender systems". In: *International Conference on Semantic Systems* (Sept. 2012), pages 1–8. DOI: 10.1145/2362499.2362501.

[34] Tommaso Di Noia, Vito Ostuni, Paolo Tomeo, and Eugenio Di Sciascio. "SPrank: Semantic Path-Based Ranking for Top-N Recommendations Using Linked Open Data". In: *ACM Transactions on Intelligent Systems and Technology* 8 (Oct. 2016). DOI: 10.1145/2899005.

[35] Mehdi Elahi, Yashar Deldjoo, Farshad Bakhshandegan Moghaddam, Leonardo Cella, Stefano Cereda, and Paolo Cremonesi. "Exploring The Semantic Gap for Movie Recommendation". In: *RecSys '17: Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, Sept. 2017. DOI: 10.1145/3109859.3109908.

[36] Zekeriya Erkin, Michael Beye, Thijs Veugen, and Reginald L. Lagendijk. "Privacy-preserving content-based recommender system". In: *Proceedings of the on Multimedia and security*. MM&amp;Sec '12. New York, NY, USA: Association for Computing Machinery, Sept. 2012, pages 77–84. ISBN: 978-1-4503-1417-6. DOI: 10.1145/2361407.2361420.

[37] A. Felfernig and R. Burke. "Constraint-based recommender systems: technologies and research issues". In: *Proceedings of the 10th international conference on Electronic commerce*. ICEC '08. New York, NY, USA: Association for Computing Machinery, Aug. 2008, pages 1–10. ISBN: 978-1-60558-075-3. DOI: 10.1145/1409540.1409544.

[38] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. "Developing Constraint-based Recommenders". In: *Recommender Systems Handbook*. Jan. 2011, pages 187–215. DOI: 10.1007/978-0-387-85820-3_6.

[39] Ignacio Fernández-Tobías, Paolo Tomeo, Iván Cantador, Tommaso Di Noia, and Eugenio Di Sciascio. "Accuracy and Diversity in Cross-domain Recommendations for Cold-start Users with Positive-only Feedback". In: *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys

'16. New York, NY, USA: Association for Computing Machinery, Sept. 2016, pages 119–122. ISBN: 978-1-4503-4035-9. DOI: 10.1145/2959100.2959175.

[40] Anna Formica and Francesco Taglino. "Semantic relatedness in DBpedia: A comparative and experimental assessment". en. In: *Information Sciences* 621 (Apr. 2023), pages 474–505. ISSN: 0020-0255. DOI: 10.1016/j.ins.2022.11.025.

[41] Francesco Ricci, Lior Rokach, and Bracha Shapira. "Recommender Systems: Techniques, Applications, and Challenges". In: *Recommender Systems Handbook* (Nov. 2021), pages 1–35. DOI: 10.1007/978-1-0716-2197-4_1.

[42] Shan Gao, Guibing Guo, Yusong Lin, Xingjin Zhang, Yongpeng Liu, and Zongmin Wang. "Pairwise Preference Over Mixed-Type Item-Sets Based Bayesian Personalized Ranking for Collaborative Filtering". In: Nov. 2017, pages 30–37. DOI: 10.1109/DASC-PICom-DataCom-CyberSciTec.2017.22.

[43] Fatih Gedikli, Dietmar Jannach, and Mouzhi Ge. "How should I explain? A comparison of different explanation types for recommender systems". en. In: *International Journal of Human-Computer Studies* 72.4 (Apr. 2014), pages 367–382. ISSN: 1071-5819. DOI: 10.1016/j.ijhcs.2013.12.007.

[44] Marco de Gemmis, Pasquale Lops, Cataldo Musto, Fedelucio Narducci, and Giovanni Semeraro. "Semantics-Aware Content-Based Recommender Systems". In: *Recommender Systems Handbook* (Jan. 2015), pages 119–159. DOI: 10.1007/978-1-4899-7637-6_4.

[45] Xiubo Geng, Tie-Yan Liu, Tao Qin, and Hang Li. "Feature selection for ranking". In: *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. SIGIR '07. New York, NY, USA: Association for Computing Machinery, July 2007, pages 407–414. ISBN: 978-1-59593-597-7. DOI: 10.1145/1277741.1277811.

[46] Abhijeet Ghoshal, Subodha Kumar, and Vijay Mookerjee. "Impact of Recommender System on Competition Between Personalizing and Non-Personalizing Firms". In: *Journal of Management Information Systems* 31.4 (Jan. 2015), pages 243–277. ISSN: 0742-1222. DOI: 10.1080/07421222.2014.1001276.

[47] Jennifer Golbeck. "Generating Predictive Movie Recommendations from Trust in Social Networks". en. In: *Trust Management*. Edited by Ketil Stølen, William H. Winsborough, Fabio Martinelli, and Fabio Massacci. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pages 93–104. ISBN: 978-3-540-34297-7. DOI: 10.1007/11755593_8.

[48] Aditya Grover and Jure Leskovec. *node2vec: Scalable Feature Learning for Networks*. July 2016. DOI: 10.48550/arXiv.1607.00653.

[49] Yan Guo, Minxi Wang, and Xin Li. "Application of an improved Apriori algorithm in a mobile e-commerce recommendation system". In: *Industrial Management & Data Systems* 117.2 (Jan. 2017), pages 287–303. ISSN: 0263-5577. DOI: 10.1108/IMDS-03-2016-0094.

[50] Isabelle Guyon and André Elisseeff. "An introduction to variable and feature selection". In: *The Journal of Machine Learning Research* 3.null (Mar. 2003), pages 1157–1182. ISSN: 1532-4435.

[51] William L. Hamilton, Rex Ying, and Jure Leskovec. *Inductive Representation Learning on Large Graphs*. Sept. 2018. DOI: 10.48550/arXiv.1706.02216.

[52] Xiaotian Han, Chuan Shi, Senzhang Wang, Philip Yu, and Li Song. "Aspect-Level Deep Collaborative Filtering via Heterogeneous Information Networks". In: July 2018, pages 3393–3399. DOI: 10.24963/ijcai.2018/471.

[53] F. Maxwell Harper and Joseph A. Konstan. "The MovieLens Datasets: History and Context". In: *ACM Transactions on Interactive Intelligent Systems* 5.4 (Dec. 2015), 19:1–19:19. ISSN: 2160-6455. DOI: 10.1145/2827872.

[54] Benjamin Heitmann and Conor Hayes. "Using Linked Data to Build Open, Collaborative Recommender Systems." In: (Mar. 2010).

[55] Hui Wang, Hui Wang, Hui Wang, Zichun Le, Zichun Le, Zichun Le, ZiChun Le, Xuan Gong, and Xuan Gong. “Recommendation System Based on Heterogeneous Feature: A Survey”. In: *IEEE Access* 8 (2020), pages 170779–170793. DOI: 10.1109/access.2020.3024154.

[56] Dietmar Jannach and Kolja Hegelich. “A case study on the effectiveness of recommendations in the mobile internet”. In: *Proceedings of the third ACM conference on Recommender systems*. RecSys ’09. New York, NY, USA: Association for Computing Machinery, Oct. 2009, pages 205–208. ISBN: 978-1-60558-435-5. DOI: 10.1145/1639714.1639749.

[57] Dietmar Jannach, Iman Kamehkhosh, and Lukas Lerche. “Leveraging multi-dimensional user models for personalized next-track music recommendation”. In: *Proceedings of the Symposium on Applied Computing*. SAC ’17. New York, NY, USA: Association for Computing Machinery, Apr. 2017, pages 1635–1642. ISBN: 978-1-4503-4486-9. DOI: 10.1145/3019612.3019756.

[58] Zhengshen Jiang, Hongzhi Liu, Bin Fu, Zhonghai Wu, and Tao Zhang. “Recommendation in Heterogeneous Information Networks Based on Generalized Random Walk Model and Bayesian Personalized Ranking”. en. In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. Marina Del Rey CA USA: ACM, Feb. 2018, pages 288–296. ISBN: 978-1-4503-5581-0. DOI: 10.1145/3159652.3159715.

[59] Sneha Khatwani and M.B. Chandak. “Building Personalized and Non Personalized recommendation systems”. In: *2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT)*. Sept. 2016, pages 623–628. DOI: 10.1109/ICACDOT.2016.7877661.

[60] Kyoung-jae Kim and Hyunchul Ahn. “A recommender system using GA K-means clustering in an online shopping market”. en. In: *Expert Systems with Applications* 34.2 (Feb. 2008), pages 1200–1209. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2006.12.025.

[61] Evan Kirshenbaum, George Forman, and Michael Dugan. “A Live Comparison of Methods for Personalized Article Recommendation at Forbes.com”. In: Springer, Sept. 2012, pages 51–66. ISBN: 978-3-642-33485-6. DOI: 10.1007/978-3-642-33486-3_4.

[62] Akiva Kleinerman, Ariel Rosenfeld, and Sarit Kraus. “Providing explanations for recommendations in reciprocal environments”. In: *Proceedings of the 12th ACM Conference on Recommender Systems*. RecSys ’18. New York, NY, USA: Association for Computing Machinery, Sept. 2018, pages 22–30. ISBN: 978-1-4503-5901-6. DOI: 10.1145/3240323.3240362.

[63] Ron Kohavi and George John. “Wrappers for Feature Subset Selection”. In: *Artificial Intelligence* 97 (Dec. 1997), pages 273–324. DOI: 10.1016/S0004-3702(97)00043-X.

[64] Anil Kumar, Nitesh Kumar, Muzammil Hussain, Santanu Chaudhury, and Sumeet Agarwal. “Semantic clustering-based cross-domain recommendation”. In: *2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*. Dec. 2014, pages 137–141. DOI: 10.1109/CIDM.2014.7008659.

[65] Matevž Kunaver, Tomaž Požrl, Matevž Pogačnik, and Jurij Tasič. “Optimisation of combined collaborative recommender systems”. en. In: *AEU - International Journal of Electronics and Communications* 61.7 (July 2007), pages 433–443. ISSN: 1434-8411. DOI: 10.1016/j.aeue.2007.04.003.

[66] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. “From Word Embeddings To Document Distances”. en. In: *Proceedings of the 32nd International Conference on Machine Learning*. PMLR, June 2015, pages 957–966.

[67] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. “Addressing cold-start problem in recommendation systems”. In: *Proceedings of the 2nd international conference on Ubiquitous information management and communication*. ICUIMC ’08. New York, NY, USA: Association for Computing Machinery, Jan. 2008, pages 208–211. ISBN: 978-1-59593-993-7. DOI: 10.1145/1352793.1352837.

[68] Aristomenis S. Lampropoulos, Paraskevi S. Lampropoulou, and George A. Tsihrintzis. “A Cascade-Hybrid Music Recommender System for mobile services based on musical genre classification and

personality diagnosis". en. In: *Multimedia Tools and Applications* 59.1 (July 2012), pages 241–258. ISSN: 1573-7721. DOI: 10.1007/s11042-011-0742-0.

[69] Cane Wing-ki Leung, Stephen Chi-fai Chan, and Fu-lai Chung. "Applying Cross-Level Association Rule Mining to Cold-Start Recommendations". In: *2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*. Nov. 2007, pages 133–136. DOI: 10.1109/WI-IATW.2007.22.

[70] Haotian Li, Yong Wang, Songheng Zhang, Yangqiu Song, and Huamin Qu. "KG4Vis: A Knowledge Graph-Based Approach for Visualization Recommendation". In: *IEEE Transactions on Visualization and Computer Graphics* 28.1 (Jan. 2022), pages 195–205. ISSN: 1941-0506. DOI: 10.1109/TVCG.2021.3114863.

[71] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang. "Pfp: parallel fp-growth for query recommendation". In: *Proceedings of the 2008 ACM conference on Recommender systems*. RecSys '08. New York, NY, USA: Association for Computing Machinery, Oct. 2008, pages 107–114. ISBN: 978-1-60558-093-7. DOI: 10.1145/1454008.1454027.

[72] Anthony Lin. "Binary search algorithm". In: *WikiJournal of Science* 2.1 (Nov. 2020), pages 1–13. DOI: 10.3316/informit.573360863402659.

[73] Jiahui Liu, Peter Dolan, and Elin Rønby Pedersen. "Personalized news recommendation based on click behavior". In: *Proceedings of the 15th international conference on Intelligent user interfaces*. IUI '10. New York, NY, USA: Association for Computing Machinery, Feb. 2010, pages 31–40. ISBN: 978-1-60558-515-4. DOI: 10.1145/1719970.1719976.

[74] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. "Content-based Recommender Systems: State of the Art and Trends". In: *Recommender Systems Handbook*. Edited by Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. Boston, MA: Springer US, 2011, pages 73–105. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_3.

[75] Fabiana Lorenzi and Francesco Ricci. "Case-Based Recommender Systems: A Unifying View". en. In: *Intelligent Techniques for Web Personalization*. Edited by Bamshad Mobasher and Sarabjot Singh Anand. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pages 89–113. ISBN: 978-3-540-31655-8. DOI: 10.1007/11577935_5.

[76] Ludovik Coba, Roberto Confalonieri, and Markus Zanker. "RecoXplainer: A Library for Development and Offline Evaluation of Explainable Recommender Systems". In: *IEEE Computational Intelligence Magazine* 17.1 (Feb. 2022), pages 46–58. DOI: 10.1109/mci.2021.3129958.

[77] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. "Image-Based Recommendations on Styles and Substitutes". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '15. New York, NY, USA: Association for Computing Machinery, Aug. 2015, pages 43–52. ISBN: 978-1-4503-3621-5. DOI: 10.1145/2766462.2767755.

[78] Edgar Meij, Marc Bron, Laura Hollink, Bouke Huurnink, and Maarten de Rijke. "Mapping queries to the Linking Open Data cloud: A case study using DBpedia". In: *Journal of Web Semantics*. JWS special issue on Semantic Search 9.4 (Dec. 2011), pages 418–433. ISSN: 1570-8268. DOI: 10.1016/j.websem.2011.04.001.

[79] Tejas Menon. "Empirical Analysis of CBOW and Skip Gram NLP Models". In: *University Honors Theses* (July 2020). DOI: 10.15760/honors.956.

[80] Rouzbeh Meymandpour and Joseph G. Davis. "Enhancing Recommender Systems Using Linked Open Data-Based Semantic Analysis of Items". In: *Australasian Web Conference* (Jan. 2015), pages 11–17.

[81] Ram Krishn Mishra, Siddhaling Urolagin, and Angel Arul Jothi J. "A Sentiment analysis-based hotel recommendation using TF-IDF Approach". In: *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*. Dec. 2019, pages 811–815. DOI: 10.1109/ICCIKE47802.2019.9004385.

[82] Raymond J. Mooney and Loriene Roy. "Content-based book recommending using learning for text categorization". In: *Proceedings of the fifth ACM conference on Digital libraries*. DL '00. New York, NY, USA: Association for Computing Machinery, June 2000, pages 195–204. ISBN: 978-1-58113-231-1. DOI: 10.1145/336597.336662.

[83] Cataldo Musto, Pierpaolo Basile, Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. "Introducing linked open data in graph-based recommender systems". In: *Information Processing and Management* 53.2 (Mar. 2017), pages 405–435. DOI: 10.1016/j.ipm.2016.12.003.

[84] Cataldo Musto, Pasquale Lops, Pierpaolo Basile, Marco de Gemmis, and Giovanni Semeraro. "Semantics-aware Graph-based Recommender Systems Exploiting Linked Open Data". In: *User Modeling, Adaptation, and Personalization* (July 2016), pages 229–237. DOI: 10.1145/2930238.2930249.

[85] Cataldo Musto, Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. "Semantics-aware Recommender Systems Exploiting Linked Open Data and Graph-based Features". In: *Knowledge Based Systems* 136 (Apr. 2018), pages 457–460. DOI: 10.1145/3184558.3186233.

[86] Ana Régia de M. Neves, Álvaro Marcos G. Carvalho, and Célia G. Ralha. "Agent-based architecture for context-aware and personalized event recommendation". en. In: *Expert Systems with Applications* 41.2 (Feb. 2014), pages 563–573. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2013.07.081.

[87] Matteo Palmonari, Anisa Rula, Riccardo Porrini, Andrea Maurino, Blerina Spahiu, and Vincenzo Ferme. "ABSTAT: Linked Data Summaries with ABstraction and STATistics". en. In: *The Semantic Web: ESWC 2015 Satellite Events*. Edited by Fabien Gandon, Christophe Guéret, Serena Villata, John Breslin, Catherine Faron-Zucker, and Antoine Zimmermann. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pages 128–132. ISBN: 978-3-319-25639-9. DOI: 10.1007/978-3-319-25639-9_25.

[88] Enrico Palumbo, Giuseppe Rizzo, Raphaël Troncy, Elena Baralis, Michele Osella, and Enrico Ferro. "Knowledge Graph Embeddings with node2vec for Item Recommendation". en. In: *The Semantic Web: ESWC 2018 Satellite Events*. Edited by Aldo Gangemi, Anna Lisa Gentile, Andrea Giovanni Nuzzolese, Sebastian Rudolph, Maria Maleshkova, Heiko Paulheim, Jeff Z. Pan, and Mehwish Alam. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pages 117–120. ISBN: 978-3-319-98192-5. DOI: 10.1007/978-3-319-98192-5_22.

[89] Jyoti Pareek, Maitri Jhaveri, Abbas Kapasi, and Malhar Trivedi. "SNetRS: Social Networking in Recommendation System". en. In: *Advances in Computing and Information Technology*. Edited by Natarajan Meghanathan, Dhinaharan Nagamalai, and Nabendu Chaki. Advances in Intelligent Systems and Computing. Berlin, Heidelberg: Springer, 2013, pages 195–206. ISBN: 978-3-642-31552-7. DOI: 10.1007/978-3-642-31552-7_21.

[90] Guangyuan Piao and John G. Breslin. "A Study of the Similarities of Entity Embeddings Learned from Different Aspects of a Knowledge Base for Item Recommendations". en. In: *The Semantic Web: ESWC 2018 Satellite Events*. Edited by Aldo Gangemi, Anna Lisa Gentile, Andrea Giovanni Nuzzolese, Sebastian Rudolph, Maria Maleshkova, Heiko Paulheim, Jeff Z. Pan, and Mehwish Alam. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pages 345–359. ISBN: 978-3-319-98192-5. DOI: 10.1007/978-3-319-98192-5_52.

[91] Eric A. Posner, Kathryn E. Spier, and Adrian Vermeule. "Divide and Conquer". In: *Journal of Legal Analysis* 2.2 (Oct. 2010), pages 417–471. ISSN: 2161-7201. DOI: 10.1093/jla/2.2.417.

[92] Chuan Qin, Hengshu Zhu, Fuzhen Zhuang, Hengshu Zhu, Qingyu Guo, Xing Xie, Xing Xie, Qi Zhang, Le Zhang, Hui Xiong, Chao Wang, Chao Wang, Chao Wang, Qing He, Enhong Chen, and Hui Xiong. "A survey on knowledge graph-based recommender systems". In: *IEEE Transactions on Knowledge and Data Engineering* 50.7 (July 2020), pages 937–956. DOI: 10.1360/ssi-2019-0274.

[93] Azzurra Ragone, Paolo Tomeo, Corrado Magarelli, Tommaso Di Noia, Matteo Palmonari, Andrea Maurino, and Eugenio Di Sciascio. "Schema-summarization in linked-data-based feature selection for recommender systems". In: Apr. 2017, pages 330–335. DOI: 10.1145/3019612.3019837.

[94] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors. *Recommender Systems Handbook*. en. Boston, MA: Springer US, 2011. ISBN: 978-0-387-85819-7 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3.

[95] Petar Ristoski, Eneldo Loza Mencía, Heiner Stuckenschmidt, and Heiko Paulheim. "A Hybrid Multi-Strategy Recommender System Using Linked Open Data". In: *SemWebEval@ESWC* (May 2014), pages 150–156. DOI: 10.1007/978-3-319-12024-9_19.

[96] Petar Ristoski and Heiko Paulheim. "Semantic Web in data mining and knowledge discovery: A comprehensive survey". In: *Journal of Web Semantics* 36 (Jan. 2016), pages 1–22. ISSN: 1570-8268. DOI: 10.1016/j.websem.2016.01.001.

[97] Petar Ristoski, Jessica Rosati, Tommaso Di Noia, Renato De Leone, Heiner Stuckenschmidt, and Heiko Paulheim. "RDF2Vec: RDF graph embeddings and their applications". In: *Social Work* 10.4 (Jan. 2019), pages 721–752. DOI: 10.3233/sw-180317.

[98] Anisa Rula, Anisa Rula, Tommaso Di Noia, Matteo Palmonari, and Andrea. "Experimental Data: Using Ontology-Based Data Summarization To Develop Semantics-Aware Recommender Systems (Eswc2018)". In: (Jan. 2018). DOI: 10.5281/zenodo.1205712.

[99] Ashish Kumar Sahu and Pragya Dwivedi. "Knowledge transfer by domain-independent user latent factor for cross-domain recommender systems". en. In: *Future Generation Computer Systems* 108 (July 2020), pages 320–333. ISSN: 0167-739X. DOI: 10.1016/j.future.2020.02.024.

[100] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. en. McGraw-Hill, 1983. ISBN: 978-0-07-054484-0.

[101] Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. "Adoption of the Linked Data Best Practices in Different Topical Domains". en. In: *The Semantic Web – ISWC 2014*. Edited by Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pages 245–260. ISBN: 978-3-319-11964-9. DOI: 10.1007/978-3-319-11964-9_16.

[102] Meenakshi Sharma and Sandeep Mann. "A Survey of Recommender Systems: Approaches and Limitations". In: 2013.

[103] Chuan Shi, Binbin Hu, Wayne Xin Zhao, and Philip S. Yu. *Heterogeneous Information Network Embedding for Recommendation*. Nov. 2017. DOI: 10.48550/arXiv.1711.10730.

[104] Chuan Shi, Zhiqiang Zhang, Ping Luo, Philip S. Yu, Yading Yue, and Bin Wu. "Semantic Path based Personalized Recommendation on Weighted Heterogeneous Information Networks". In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM '15. New York, NY, USA: Association for Computing Machinery, Oct. 2015, pages 453–462. ISBN: 978-1-4503-3794-6. DOI: 10.1145/2806416.2806528.

[105] Pradeep Kumar Singh, Pijush Kanti Dutta Pramanik, Pijush Kanti Dutta Pramanik, Avick Kumar Dey, and Prasenjit Choudhury. "Recommender systems: an overview, research trends, and future directions". In: *International Journal of Business and Systems Research* 15.1 (2021), pages 14–52. DOI: 10.1504/ijbsr.2021.10033303.

[106] Blerina Spahiu, Riccardo Porrini, Matteo Palmonari, Anisa Rula, and Andrea Maurino. "ABSTAT: Ontology-Driven Linked Data Summaries with Pattern Minimalization". In: volume 9989. May 2016, pages 381–395. ISBN: 978-3-319-47601-8. DOI: 10.1007/978-3-319-47602-5_51.

[107] Heiner Stuckenschmidt, Heiko Paulheim, and Johannes Fümkranz. "Unsupervised generation of data mining features from linked open data". In: (June 2012), page 31. DOI: 10.1145/2254129.2254168.

[108] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. "PathSim: meta path-based top-K similarity search in heterogeneous information networks". en. In: *Proceedings of the VLDB Endowment* 4.11 (Aug. 2011), pages 992–1003. ISSN: 2150-8097. DOI: 10.14778/3402707.3402736.

[109] Flavian Vasile, Elena Smirnova, and Alexis Conneau. "Meta-Prod2Vec: Product Embeddings Using Side-Information for Recommendation". In: *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys '16. New York, NY, USA: Association for Computing Machinery, Sept. 2016, pages 225–232. ISBN: 978-1-4503-4035-9. DOI: 10.1145/2959100.2959160.

[110] Vito Walter Anelli, Vito Walter Anelli, Tommaso Di Noia, Andrea Maurino, Andrea Maurino, Matteo Palmonari, and Anisa Rula. "Ontology-based Linked Data Summarization in Semantics-aware Recommender Systems." In: *Sistemi Evoluti per Basi di Dati* (Jan. 2018). DOI: 10.1007/978-3-319-93417-4_9.

[111] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. "Knowledge Graph Embedding: A Survey of Approaches and Applications". In: *IEEE Transactions on Knowledge and Data Engineering* 29.12 (Dec. 2017), pages 2724–2743. ISSN: 1558-2191. DOI: 10.1109/TKDE.2017.2754499.

[112] Shoujin Wang, Liang Hu, Yan Wang, Xiangnan He, Quan Z. Sheng, Mehmet A. Orgun, Longbing Cao, Francesco Ricci, and Philip S. Yu. "Graph Learning based Recommender Systems: A Review". en. In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*. Montreal, Canada: International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pages 4644–4652. ISBN: 978-0-9992411-9-6. DOI: 10.24963/ijcai.2021/630.

[113] Zekai Wang, Hongzhi Liu, Yingpeng Du, Zhonghai Wu, and Xing Zhang. "Unified embedding model over heterogeneous information network for personalized recommendation". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. IJCAI'19. Macao, China: AAAI Press, Aug. 2019, pages 3813–3819. ISBN: 978-0-9992411-4-1.

[114] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. "Knowledge Graph Embedding by Translating on Hyperplanes". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 28.1 (June 2014). ISSN: 2374-3468. DOI: 10.1609/aaai.v28i1.8870.

[115] Shiwen Wu, Wentao Zhang, Fei Sun, Fei Sun, Fei Sun, and Bin Cui. "Graph Neural Networks in Recommender Systems: A Survey". In: *arXiv: Information Retrieval* (2020). DOI: 10.1145/3535101.

[116] Xu Yang, Ziyi Huan, Yisong Zhai, and Ting Lin. "Research of Personalized Recommendation Technology Based on Knowledge Graphs". en. In: *Applied Sciences* 11.15 (Jan. 2021), page 7104. ISSN: 2076-3417. DOI: 10.3390/app11157104.

[117] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. "Graph Convolutional Neural Networks for Web-Scale Recommender Systems." In: *Knowledge Discovery and Data Mining* (June 2018). DOI: 10.1145/3219819.3219890.

[118] Chi-Chih Yu, Toru Yamaguchi, and Yasufumi Takama. "A hybrid recommender system based non-common items in social media". In: *2013 International Joint Conference on Awareness Science and Technology & Ubi-Media Computing (iCAST 2013 & UMEDIA 2013)*. Nov. 2013, pages 255–261. DOI: 10.1109/ICAwST.2013.6765443.

[119] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. "Personalized entity recommendation: a heterogeneous information network approach". en. In: *Proceedings of the 7th ACM international conference on Web search and data mining*. New York New York USA: ACM, Feb. 2014, pages 283–292. ISBN: 978-1-4503-2351-2. DOI: 10.1145/2556195.2556259.

[120] Yu Peng. "A Survey on Modern Recommendation System based on Big Data". In: *ArXiv* (2022). DOI: 10.48550/arxiv.2206.02631.

[121] Anna V. Zagranovskaya and Dmitriy Y. Mitura. "DESIGNING HYBRID RECOMMENDER SYSTEMS". In: *International scientific journal* 5 (Jan. 2019), pages 88–94. DOI: 10.34286/1995-4638-2019-68-5-88-94.

[122] Feng Zhang, Ti Gong, Victor E. Lee, Gansen Zhao, Gansen Zhao, Gansen Zhao, Chunming Rong, and Guangzhi Qu. "Fast algorithms to evaluate collaborative filtering recommender systems". In: *Knowledge Based Systems* 96 (Mar. 2016), pages 96–103. DOI: 10.1016/j.knosys.2015.12.025.

[123] Qian Zhang, Peng Hao, Jie Lu, and Guangquan Zhang. “Cross-domain Recommendation with Semantic Correlation in Tagging Systems”. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. July 2019, pages 1–8. DOI: 10.1109/IJCNN.2019.8852049.

[124] Feng Zhu, Yan Wang, Chaochao Chen, Jun Zhou, Longfei Li, and Guanfeng Liu. “Cross-Domain Recommendation: Challenges, Progress, and Prospects”. In: Apr. 2021. DOI: 10.24963/ijcai.2021/639.

[125] Ganggao Zhu and Carlos A. Iglesias. “Exploiting semantic similarity for named entity disambiguation in knowledge graphs”. In: *Expert Systems with Applications* 101 (July 2018), pages 8–24. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2018.02.011.

Appendix A

Literature Review Results

Title	Author	Year	Publisher	Approach	Citation
Content-based Recommender Systems: State of the Art and Trends	Lops et al.	2011	Springer	Content-based	[74]
Agent-based architecture for context-aware and personalized event recommendation.	Neves et al.	2014	Science Direct	Content-based	[86]
A case study on the effectiveness of recommendations in the mobile internet	Jannach et al.	2009	ACM	Content-based	[56]
A Live Comparison of Methods for Personalized Article Recommendation at Forbes.com	Kirshenbaum et al.	2012	Springer	Content-based	[61]
Personalized news recommendation based on click behavior	Liu et al.	2012	ACM	Content-based	[73]
Introduction to Modern Information Retrieval	Salton et al.	1983	Google Books	Content-based	[100]
Semantics-Aware Content-Based Recommender Systems	de Gemmis et al.	2015	Springer	Content-based	[44]
Image-Based Recommendations on Styles and Substitutes	McAuley et al.	2015	ACM	Content-based	[77]
Exploring The Semantic Gap for Movie Recommendation	Elahi et al.	2017	ACM	Content-based	[35]
Using visual features based on MPEG-7 and deep learning for movie recommendation	Deldjoo et al.	2018	Springer	Content-based	[31]
Generating Predictive Movie Recommendations from Trust in Social Networks	Golbeck, Jennifer	2006	Springer	Content-based	[47]
Providing explanations for recommendations in reciprocal environments	Kleinerman et al.	2018	ACM	Content-based	[62]
How should I explain? A comparison of different explanation types for recommender systems	Gedikli et al.	2014	ScienceDirect	Content-based	[43]
Leveraging multi-dimensional user models for personalized next-track music recommendation	Jannach et al.	2017	ACM	Content-based	[57]

PathSim: meta path-based top-K similarity search in heterogeneous information networks	Sun et al.	2011	ACM	Content-based	[108]
Semantic Path based Personalized Recommendation on Weighted Heterogeneous Information Networks	Shi et al.	2015	ACM	Content-based	[104]
Personalized entity recommendation: a heterogeneous information network approach	Yu et al.	2014	ACM	Content-based	[119]
Graph Convolutional Neural Networks for Web-Scale Recommender Systems	Ying et al.	2018	ACM	Content-based	[117]
Using genetic algorithms for measuring the similarity values between users in collaborative filtering recommender systems	Alhijawi et al.	2016	IEEE	Collaborative-filtering based	[2]
Personalized entity recommendation: a heterogeneous information network approach	Yu et al.	2014	ACM	Collaborative-filtering based	[119]
A Survey of Recommender Systems: Approaches and Limitations	Sharma et al.	2013	Semantic Scholar	Collaborative-filtering based	[102]
Privacy-preserving content-based recommender system	Erkin et al.	2012	ACM	Collaborative-filtering based	[36]
SNetRS: Social Networking in Recommendation System	Pareek et al.	2013	ACM	Collaborative-filtering based	[89]
Current Trends in Collaborative Filtering Recommendation Systems	Amin et al.	2019	Springer	Collaborative-filtering based	[5]
Building Personalized and Non Personalized recommendation systems	Khatwani et al.	2016	IEEE	Collaborative-filtering based	[59]
A recommender system using GA K-means clustering in an online shopping market	Kim et al.	2008	ScienceDirect	Collaborative-filtering based	[60]
Movie recommendation system using clustering and pattern recognition network	Ahmed et al.	2018	IEEE	Collaborative-filtering based	[1]

Application of an improved Apriori algorithm in a mobile e-commerce recommendation system	Guo et al.	2017	Emerald	Collaborative-filtering based	[49]
Pfp: parallel fp-growth for query recommendation	Li et al.	2008	ACM	Collaborative-filtering based	[71]
Pairwise Preference Over Mixed-Type Item-Sets Based Bayesian Personalized Ranking for Collaborative Filtering	Gao et al.	2017	ResearchGate	Collaborative-filtering based	[42]
CoBaFi - Collaborative Bayesian filtering	Beutel et al.	2014	ResearchGate	Collaborative-filtering based	[15]
Modeling item selection and relevance for accurate recommendations: a bayesian approach	Barbieri et al.	2011	ACM	Collaborative-filtering based	[10]
Constraint-based recommender systems: technologies and research issues	Felfernig et al.	2008	ACM	Knowledge-based	[37]
Developing Constraint-based Recommenders	Felfernig et al.	2011	ResearchGate	Knowledge-based	[38]
Case-based recommender systems	Bridge et al.	2005	ResearchGate	Knowledge-based	[16]
Knowledge-based recommender systems	Burke, Robin	2000	ResearchGate	Knowledge-based	[17]
Case-Based Recommender Systems: A Unifying View	Lorenzi et al.	2005	Springer	Knowledge-based	[75]
Recommender systems based on graph embedding techniques: A comprehensive review	Yue, Deng	2022	IEEE	Hybrid-based	[32]
Hybrid Recommender Systems: A Systematic Literature Review	Çano et al.	2017	IDA	Hybrid-based	[19]
Alleviating the cold-start problem of recommender systems using a new hybrid approach	Basiri et al.	2010	IEEE	Hybrid-based	[11]
A hybrid recommender system based non-common items in social media	Yu et al.	2013	IEEE	Hybrid-based	[118]
Modeling user preferences in a hybrid recommender system using type-2 fuzzy sets	Bedi et al.	2013	IEEE	Hybrid-based	[12]

Content-based book recommending using learning for text categorization	Mooney et al.	2000	ACM	Hybrid-based	[82]
A Cascade-Hybrid Music Recommender System for mobile services based on musical genre classification and personality diagnosis	Lampropoulos et al.	2012	Springer	Hybrid-based	[68]
Optimisation of combined collaborative recommender systems	Kunaver et al.	2007	ScienceDirect	Hybrid-based	[65]
Cross-Domain Recommendation: Challenges, Progress, and Prospects	Zhu et al.	2021	ResearchGate	Cross-domain	[124]
Cross-Domain Mediation in Collaborative Filtering	Berkovsky et al.	2007	Springer	Cross-domain	[14]
Applying Cross-Level Association Rule Mining to Cold-Start Recommendations	Leung et al.	2007	IEEE	Cross-domain	[69]
Semantic clustering-based cross-domain recommendation	Kumar et al.	2014	IEEE	Cross-domain	[64]
Cross-domain Recommendation with Semantic Correlation in Tagging Systems	Zhang et al.	2019	IJCNN	Cross-domain	[123]
Graph learning based recommender systems: a review	Wang et al.	2021	IJCAI	Graph-based	[112]
Recommender systems based on graph embedding techniques: A comprehensive review	Yue, Deng	2022	IEEE	Graph-based	[32]
Video suggestion and discovery for youtube: Taking random walks through the view graph	Baluja et al.	2008	ResearchGate	Graph-based	[9]
Recommendation in Heterogeneous Information Networks Based on Generalized Random Walk Model and Bayesian Personalized Ranking	Jiang et al.	2018	ACM	Graph-based	[58]

Context-Aware Friend Recommendation for Location Based Social Networks using Random Walk	Bagci et al.	2016	ACM	Graph-based	[8]
Unified embedding model over heterogeneous information network for personalized recommendation	Wang et al.	2019	ACM	Graph-based	[113]
Heterogeneous Information Network Embedding for Recommendation	Shi et al.	2017	IEEE	Graph-based	[103]
Representation Learning for Attributed Multiplex Heterogeneous Network	Cen et al.	2019	ACM	Graph-based	[21]
Aspect-Level Deep Collaborative Filtering via Heterogeneous Information Networks	Han et al.	2018	ResearchGate	Graph-based	[52]
Using Linked Data to Build Open, Collaborative Recommender Systems	Heitmann et al.	2010	AAAI	LOD-based	[54]
Linked open data to support content-based recommender systems	Di Noia et al.	2012	ACM	LOD-based	[33]
A Hybrid Multi-Strategy Recommender System Using Linked Open Data	Ristoski et al.	2014	Springer	LOD-based	[95]
Semantics-Aware Content-Based Recommender Systems	de Gemmis et al.	2014	Springer	LOD-based	[44]
Semantics-aware Recommender Systems Exploiting Linked Open Data and Graph-based Features	Musto et al.	2018	ScienceDirect	LOD-based	[85]
Ontology-based Linked Data Summarization in Semantics-aware Recommender Systems	Vito Walter et al.	2018	Springer	LOD-based	[110]
Knowledge transfer by domain-independent user latent factor for cross-domain recommender systems	Sahu et al.	2020	ScienceDirect	LOD-based	[99]

Balancing the trade-off between accuracy and diversity in recommender systems with personalized explanations based on Linked Open Data	André et al.	2022	ScienceDirect	LOD-based	[6]
Unsupervised generation of data mining features from linked open data	Stuckenschmidt et al.	2012	ScienceDirect	Feature extraction	[107]
Semantics-Aware Content-Based Recommender Systems	de Gemmis et al.	2015	ScienceDirect	Semantics Aware	[44]
Introducing linked open data in graph-based recommender systems	Musto et al.	2017	ScienceDirect	Semantics Aware	[83]
Semantics-aware Graph-based Recommender Systems Exploiting Linked Open Data	Musto et al.	2016	ACM	Semantics Aware	[84]
Enhancing Recommender Systems Using Linked Open Data-Based Semantic Analysis of Items	Meymandpour et al.	2015	ResearchGate	Semantics Aware	[80]
Semantic relatedness in DBpedia: A comparative and experimental assessment	Formica et al.	2023	SciendeDirect	Semantics Aware	[40]
Experimental Data: Using Ontology-Based Data Summarization To Develop Semantics-Aware Recommender Systems	Rula et al.	2018	Eswc	Semantics Aware	[98]
Ontology-based Linked Data Summarization in Semantics-aware Recommender Systems.	Vito Walter et al.	2018	Springer	Semantics Aware	[110]

Table A.1: Papers included in our Literature Review

Appendix B

Predicates Excluded from the Random Walks Used by RDF2Vec

Excluded Predicates
www.w3.org/1999/02/22-rdf-syntax-ns#type
http://xmlns.com/foaf/0.1/depiction
http://dbpedia.org/property/wikiPageUsesTemplate
http://dbpedia.org/ontology/thumbnail
http://dbpedia.org/property/wordnet_type
http://www.w3.org/ns/prov#wasDerivedFrom
http://xmlns.com/foaf/0.1/isPrimaryTopicOf
http://dbpedia.org/ontology/wikiPageDisambiguates
http://xmlns.com/foaf/0.1/primaryTopic
http://xmlns.com/foaf/0.1/isPrimaryTopicOf
http://dbpedia.org/property/wikiPageExternalLink
http://dbpedia.org/property/wikiPageID
http://dbpedia.org/property/wikiPageRevisionID
http://www.w3.org/2002/07/owl#sameAs
http://dbpedia.org/ontology/wikiPageWikiLink
http://schema.org/sameAs
http://purl.org/linguistics/gold/hypernym
http://www.w3.org/2000/01/rdf-schema#seeAlso
http://www.w3.org/2002/07/owl#differentFrom

Appendix C

Pseudocode and Workflow of Our Mapping Algorithm using Type Filtering

Algorithm 4 Pseudocode of our mapping algorithm using type filtering

```
 1: Input: userInput, filterType
 2: Output: outputResult
 3: searchResults ← DBLookup(term: userInput, field: "label", exact=True, filter: filterType)
 4: if searchResults are empty then
 5:     searchResults ← DBLookup(term: userInput, field: "all", exact=True, filter: filterType)
 6:     if searchResults are empty then
 7:         searchResults ← DBLookup(term: userInput, field: "all", exact=False, filter: filterType)
 8:         if searchResults are empty then
 9:             searchResults ← DBLookup(term: userInput filterType, field: "all", exact=False, filter: filterType)
10:             if searchResults are empty then
11:                 return None
12: for each result in searchResults do
13:     if userInput matches label of result then
14:         if result has attribute dbo:wikiPageRedirects then
15:             redirectTarget ← object of dbo:wikiPageRedirects
16:             return redirectTarget
17:         else
18:             return result
19: nonMatching ← []
20: for each result in searchResults do
21:     if userInput is not a substring of label of result then
22:         nonMatching ← nonMatching+[result]
23: if length of nonMatching ≠ length of searchResults then
24:     searchResults ← searchResults − nonMatching
25: bestMatch ← null
26: for each result in searchResults do
27:     if userInput contains multiple words then
28:         update bestMatch with the result whose label shares the most words with userInput
29:     else
30:         update bestMatch with the result whose label is closest in length to userInput
31: if bestMatch has attribute dbo:wikiPageRedirects then
32:     redirectTarget ← object of dbo:wikiPageRedirects
33:     return redirectTarget
34: else
35:     return bestMatch
```

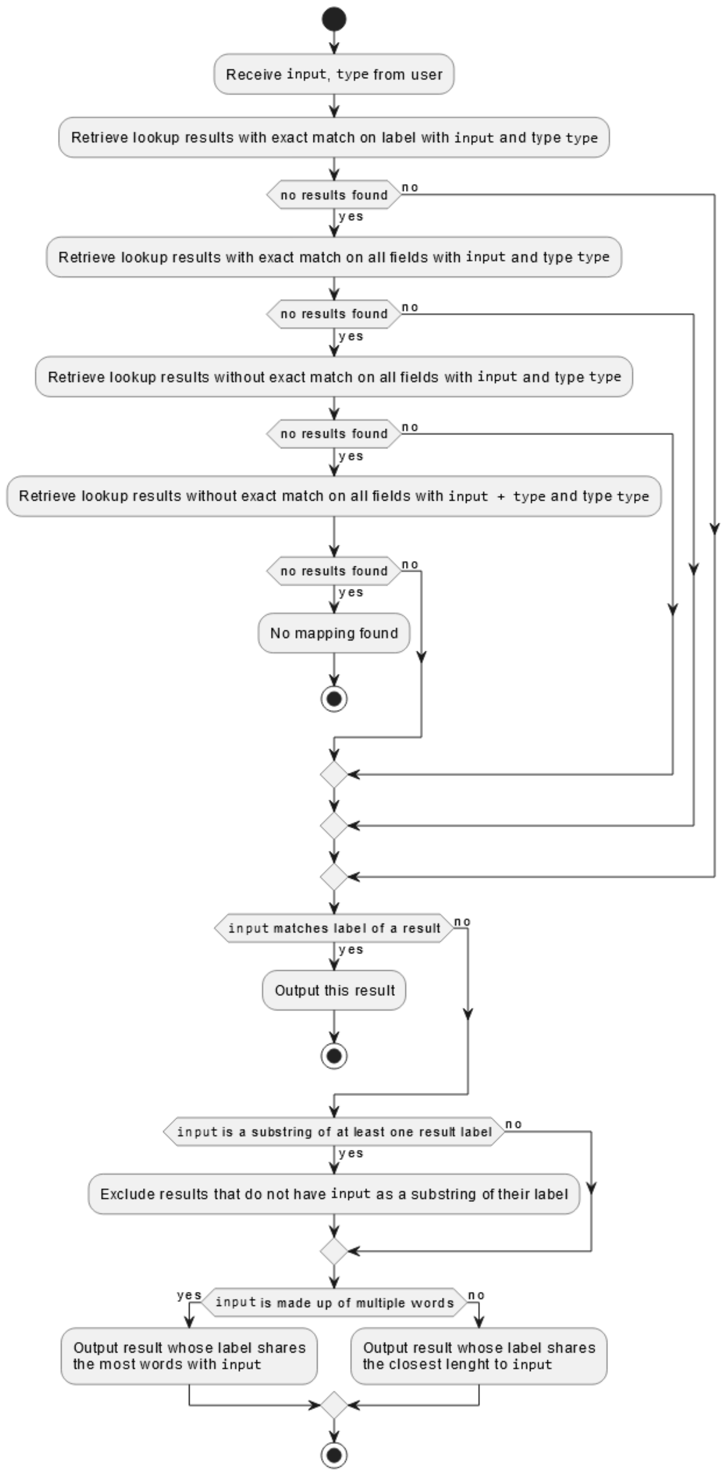

Receive input, type from user
Retrieve lookup results with exact match on label with input and type type
no results found
no
yes
Retrieve lookup results with exact match on all fields with input and type type
no results found
no
yes
Retrieve lookup results without exact match on all fields with input and type type
no results found
no
yes
Retrieve lookup results without exact match on all fields with input + type and type type
no results found
no
yes
No mapping found
input matches label of a result
no
yes
Output this result
input is a substring of at least one result label
no
yes
Exclude results that do not have input as a substring of their label
yes
input is made up of multiple words
no
Output result whose label shares the most words with input
Output result whose label shares the closest lenght to input

Figure C.1: Workflow of our mapping algorithm using type filtering

Appendix D

Example of Similar Entity Retrieval using a SPARQL Query

Listing D.1: SPARQL query to find and score entities that share attributes with *<http://dbpedia.org/resource/Batman_(1989_film)>* in the DBpedia ontology

```
SELECT ?item (COUNT(?attr) AS ?score)
WHERE {
  ?item a <http://dbpedia.org/ontology/Film> .

  OPTIONAL {
    ?item <http://dbpedia.org/ontology/starring> ?attr1 .
    FILTER(?attr1 IN (<http://dbpedia.org/resource/Jack_Palance>,
    <http://dbpedia.org/resource/Robert_Wuhl>,
    <http://dbpedia.org/resource/Pat_Hingle>,
    <http://dbpedia.org/resource/Jack_Nicholson>,
    <http://dbpedia.org/resource/Michael_Gough>,
    <http://dbpedia.org/resource/Billy_Dee_Williams>,
    <http://dbpedia.org/resource/Kim_Basinger>,
    <http://dbpedia.org/resource/Michael_Keaton>))
    BIND("attribute_1" AS ?attr)
  }

  OPTIONAL {
    ?item <http://dbpedia.org/ontology/director> ?attr2 .
    FILTER(?attr2 = <http://dbpedia.org/resource/Tim_Burton>)
    BIND("attribute_2" AS ?attr)
  }

  FILTER(?item != <http://dbpedia.org/resource/Batman_(1989_film)>)
}
GROUP BY ?item
ORDER BY DESC(?score)
LIMIT 10
```