Aalto University
School of Science

Bachelor's Programme in School of Science

Knowledge graph enhanced recommender systems

Andreas Bogossian

Bachelor's Thesis
2025

Aalto University
School of Science

Author Andreas Bogossian

Title Knowledge graph enhanced recommender systems

Degree programme School of Science

Major Computer Science

Teacher in charge and advisor Dr. Anchen Li

Date September 15, 2025 **Number of pages** 25 **Language** English

Abstract

In the age of the internet, the amount of information accessible to humans is overwhelming. To address this challenge, recommender systems have been introduced to filter and personalize information. Traditional recommender system implementations like collaborative filtering and content-based filtering are popular because of their effectiveness and relative ease of implementation. Despite their popularity, traditional systems struggle with cold start issues, data sparsity, and lack of explainability. Enhancing the recommender system with knowledge graphs can solve these problems.

Knowledge graph-based recommender systems solve the cold start problem and have better explainability. Knowledge graph-based systems can be divided into two categories based on how the graphs are modelled. The first category, embedding based methods, enhance item and user representations by embedding knowledge graph information to a lower dimensional space. Path-based methods use meta-paths to create explainable recommendations. Although KG-based recommender systems have many benefits, they have their weaknesses. A drawback with knowledge graph-based recommender systems is that knowledge graphs can have biases in them and that implementing a knowledge graph in the system will increase computational overhead.

The main approach of this thesis is to do a literature review. Articles on recommender system taxonomy, performance and algorithms are covered so that knowledge graph-based systems can be compared to traditional recommender systems. The thesis covers content-based filtering, collaborative filtering and hybrid methods. Then knowledge graph modelling techniques are discussed. Finally recommendation based on the graph modelling techniques is covered and the thesis ends in a discussion about the benefits of knowledge graph enhanced recommender systems.

Keywords recommender systems, collaborative filtering, content-based filtering, knowledge graphs, knowledge graph embedding, path-based methods, cold-start problem, explainable recommendation

Aalto-yliopisto
Perustieteiden
korkeakoulu

Tekijä Andreas Bogossian

Työn nimi Tietograafein parannellut suosittelusysteemit

Koulutusohjelma Perustieteiden korkeakoulu

Pääaine Tietotekniikka

Vastuuopettaja ja ohjaaja Dr. Anchen Li

Päivämäärä **Sivumäärä** 25 **Kieli** englanti

Tiivistelmä

Saatavilla olevan tiedon määrä on kuormittavaa internetin yleistyttyä. Ratkaisuksi on kehitetty suosittelualgoritmeja, joiden avulla tietoa pystyy suodattamaan ja personoimaan, jotta käyttäjät näkevät vain relevanttia tietoa. Nykyään, niin kutsutut perinteiset suosittelusysteemit, ovat laajasti implementoituja, koska ne ovat tehokkaita ja suhteellisen yksinkertaisia saada toimimaan. Vaikka perinteiset metodit ovat suosittuja ja laajasti käytössä, niilläkin on omat heikkoutensa. Merkittävimpiin heikkouksiin kuuluu kylmäkäynnistysongelmat ja suositusten huono selitettävyys. Näitä heikkouksia pystyy minimioimaan tietograafein parannetuilla suosittelusysteemeillä.

Tietograafein parannetut suosittelusysteemit pystyvät korjaamaan perinteisten metodien haasteet. Kylmäkäynnistysonglema korjaantuu käyttämällä tietograafeja suositusten tukena kylmäkäynnistystilantiessa. Selitettävämpiä suosituksia saa polkupohjaisten menetelmien avulla. Vaikka tietograafit paikkaavat monia perinteisten metodien haasteita, graafipohjaisilla menetelmillä on omat haasteensa. Yksi merkittävimmistä haasteista on että graafissa olevat harhat periytyvät suosittelusysteemiin.

Opinnäytetyö käyttää tutkimusmenetelmänä kirjallisuuskatsausta. Työssä tarkastellaan artikkeleita perinteisistä suosittelusysteemeistä ja tietograaffipohjaisista suosittelusysteemeistä. Opinnäytetyö pureutuu suosittelualgoritmien algoritmeihin, jotta lukija pystyy hahmottamaan mistä minkäkin metodin vahvuudet ja heikkoudet johtuvat.

Avainsanat suosittelusysteemit, kollaboratiivinen suodatus, sisältöpohjainen suodatus, tietograafit, tietograafiupotukset, polkupohjaiset metodit, kylmäkäynnistysonglema, selitettävä suosittelu

Contents

Abstract

Abstract (in Finnish)

Contents

Symbols and abbreviations

1 Introduction **1**

2 Recommender system prerequisites **3**
2.1 Recommender system data . . . 3
2.2 Challenges in recommender systems . . . 4
2.3 Recommender systems performance evaluation . . . 6
2.4 Lessons from the 2006 Netflix prize . . . 7

3 Traditional recommender systems **8**
3.1 Content-based filtering . . . 8
3.2 Collaborative Filtering . . . 8
3.2.1 Memory-based collaborative filtering . . . 9
3.2.2 Model-based collaborative filtering . . . 10
3.3 Hybrid methods . . . 10
3.4 Advantages and Limitations of traditional recommender systems . . . 11
3.4.1 Content-based filtering . . . 11
3.4.2 Collaborative filtering . . . 11
3.4.3 Hybrid methods . . . 11

4 Knowledge graphs **13**

5 Knowledge graph modelling methods **14**
5.1 Path-based Methods . . . 14
5.2 Embedding-based methods . . . 15
5.3 Unified methods . . . 17

6 Recommendation with knowledge graphs **19**
6.1 Path-based KG recommender methods . . . 19
6.2 Embedding-based KG recommender methods . . . 19
6.3 Unified KG recommender methods . . . 20
6.4 Advantages of knowledge graph enhanced recommender systems . . . 20
6.5 Limitations of knowledge graph enhanced recommender systems . . . 21
6.6 Open source knowledge graphs . . . 21

7 Conclusions **22**

8 Summary **22**

9 Future directions **22**

Symbols and abbreviations

Symbols

$\mathbb{R}^d$	d-dimensional vector space
$u_i \in \mathcal{U}$	A user in the set of all users
$v_j \in \mathcal{V}$	An item in the set of all items
$\hat{r}$	Predicted value of r
$p_{m \rightsquigarrow n}$	The path between the entities m and n
$f_r(\cdot)$	Loss function
$\text{proj}_W(h)$	The projection of vector h to plane W
$\mathbf{u}$	Latent vector obtained from a matrix factorization

Operators

$P(\cdot) \sim \mathcal{A}$	P follows the distribution $\mathcal{A}$
$U \times V$	The cross product of sets U and V
$\arg\max f(\cdot)$	The value of the argument at the function f maximum
$\|\| \cdot \|\|_2$	L2 norm
$\|\| \cdot \|\|_F$	Forbenius norm
$\| \cdot \|$	The number of elements in a set
$\phi : A \rightarrow B$	Function ϕ mapping the space A to space B
u^T	Vector transpose
$[h]_i$	The ith element of vector h
$diag(\cdot)$	Creates a diagonal matrix by zeroing non-diagonal entries

Abbreviations

RS	Recommender system
MAE	Mean absolute error
RMSE	Root mean squared error
AUC	Area under the curve (classification accuracy)
NDCG	Normalized discounted cumulative gain
CBF	Content-based filtering
CF	Collaborative filtering
KG	Knowledge graph
HIN	Heterogeneous information network
MF	Matrix factorization
SVD	Singular value decomposition
TDM	Translation distance model
SMM	Semantic matching model

Terminology

User	A user of a recommender system
Item	The entity that recommender systems recommend
Target user	The user that recommendations are being calculated for
Entity	A node in a knowledge graph representing a real life concept
Relation	The relationship between two nodes in a knowledge graph.

1 Introduction

With the introduction of the internet, the amount of information available has increased dramatically. This rapid increase of information makes it more difficult to find important information and can also overwhelm internet users. To combat this information overload, recommender systems have been successfully applied to filter and personalize information for users.

A study by Roy and Dutta [1] systematically reviewed research articles on recommender system applications. In their paper they filtered relevant articles discussing recommender system applications and categorized them into application sectors. They created Table 1 that shows the research distribution in each RS application sector. The table states that the biggest applications are in entertainment, social media and e-commerce consisting of 63% of all papers on RS applications.

Recommender systems are used on a daily basis on the internet. In the entertainment industry, Netflix and Spotify use recommendation algorithms to suggest music and movies, and on social media platforms such as Instagram or TikTok, pictures, people and short form content are recommended to users. In web/e-commerce, search engines such as Google use recommender systems to recommend web pages for users. In e-commerce, users can also receive personalised search results or targeted discounts from membership programs.

Sector	No	% (approx.)
Entertainment	16	26
Social media/others	12	20
Web/e-commerce	10	17
Health	8	13
Tourism	7	12
Education	7	12
Total	60	100

Table 1: Article distribution by sector [1].

The one common factor of all these applications is an overwhelming amount of content to show to the user. For businesses with an overwhelming amount of items to recommend, recommender systems play a vital role in their business model. A good example of this is Amazon. On Amazon, between 20% and 40% of the sales are due to products that do not belong to the 100,000 most sold products [2]. Another study concluded that 35% of Amazon's annual revenue comes from personalised recommendations [3]. The conclusion is that without recommender systems, many businesses would lose significant amounts of revenue.

The main benefit of recommendation is that it helps improve user engagement and retention [4]. By offering personalised recommendations, RSs keep users engaged for longer periods of time. This is valuable for all businesses because typically, the longer a user spends on a site, the more they tend to buy/consume. On social media, the longer the user spends on the site, the more ads can be shown to the user. In e-commerce, the longer the user spends on the site, the more they tend to buy. E-commerce sites might also lose out on sales because a user can not find an item on their site and resort to a competitor that is able to provide the product.

A side benefit for implementing a RS is that companies can create analytics based on the data. This helps companies understand their users better and help them understand trends better. Google has gone even further where they have integrated the use of this data into their business model. Google uses the data they gather for creating personalised search to target ads. This way the search engine can be free for the average consumer, and the companies get better targeted ads.

The goal of this thesis is to compare traditional recommender systems and knowledge

graph-based recommender systems. Traditional recommender systems suffer form data sparsity problems and knowledge graph-based methods are able to address this problem. The aim is to inspect the algorithms behind each recommender system to understand what the strengths and weaknesses of each method are. This is important to understand what are the mechanisms that cause sparsity problems and the methods for solving them.

The research is conducted as a literature review. The thesis reviews existing articles from the past 20 to 30 years mainly because the information explosion on the internet has happened in the past 30 years. Before that, there was not that much of information on the internet and finding information could be managed without a recommender system. To give context, Google was founded in 1998 and it was among the first companies to pioneer in commercial recommender systems.

The main databases for finding articles are Google scholar and Scopus. The main keywords used for search are "recommender systems" and "knowledge graph-based recommender systems". For more specialised research on a method, the name of the method is used. After entering a search query, the titles of search results are skimmed and the most relevant articles are opened. If the article seems relevant after quick inspection, the article is downloaded locally for deeper research.

After finding relevant articles, they are researched. The research is done by writing notes on the relevant topics. When a topic is completed, the notes are then organized and written in full paragraphs. Because bullet points can be organized and moved around easier than full paragraphs, the notes can be organized into a more logical and coherent message.

The thesis is divided into sections to structure the thesis The first and second sections introduce the topic and give prerequisite information about recommender systems. The second section, section 3, discusses traditional recommender systems, the algorithms used in them and their strengths and weaknesses. The third section, section 4, defines knowledge graphs. In the fourth section, section 5, knowledge graphs are defined and modelling techniques and algorithms for expressing knowledge graphs are discussed. In the fifth section, section 6, methods for applying KGs to recommendation are discussed. The modelling methods methods introduced in section 5 are put into practical use. The last sections, findings of the thesis are summarized and conclusions are made based on the conducted research.

2 Recommender system prerequisites

2.1 Recommender system data

A formal definition of a recommender system can be given as in article [5]:

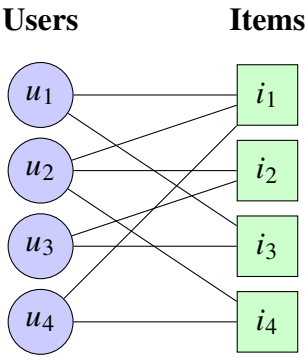
Definition 2.1 (Recommender system) *Given a set U of users, a set V of items, and the preference of user U_i for V_j denoted by $f : U \times V \rightarrow R$. The goal of the recommender system is that for any user U_i, the expectation is to find the item V_k for which the user finds the highest preference:*

$$\forall U_i \in U, V_k = \arg\max_{V_j \in V} f(U_i, V_j)$$

As stated in 2.1, the goal of a RS is to predict the item V_k that the user prefers the most. This task typically done based on user-item interaction data. The data can be represented as a bipartite graph. The bipartite graph has two distinct sets of users $u \in U$ and items $v \in V$. There are relations $r \in R$ that connect users to items and the relations represent the interactions between the user and item. The bipartite graph is visualized in Figure 1a.

Data is typically stored as a utility matrix shown in Figure 1b. This format allows developers to do matrix operations on the data. Dimensionality reduction techniques are popular when creating recommendations and the matrix form supports these operations.

Another benefit of storing data in a utility matrix is that numerical information is easier to store and process in it. While graphs typically store their connections as binary information (connected or not connected), the utility matrix can store information about ratings. Typically the binary representation is not enough and there is a need to quantify how much the user liked the item. An example of this is a movie rating system that prompts users to rate movies on a scale from one to five.



(a) Bipartite graph representation

	Item 1	Item 2	Item 3	Item 4
User 1	4	1	3	–
User 2	3	2	–	4
User 3	5	–	2	1
User 4	2	3	5	–

(b) Utility matrix

Figure 1: Recommender system data representation: graph and matrix

There are two types of user-item interaction data that can be used for recommendation: *explicit feedback* and *implicit feedback*.

Explicit feedback is feedback where the user consciously expresses their opinion on an item. When a user expresses their opinion on an item by, for example by upvoting or downvoting a social media post, it is a clear indicator of whether the user likes the content or not. This clear signalling makes this type of feedback also generalizable to different domains. An upvote on Reddit means the same thing on all other social media platforms and e-commerce sites. This is probably why explicit feedback is so well researched and understood.

Even though explicit feedback is well researched and understood, it has a large disadvantage: Users are reluctant to give explicit feedback [6]. It is not always clear why users are reluctant to give feedback, but one of the main reasons is that it requires a bit more effort to give it. Systems that only rely on explicit feedback are not top performing methods because so many user-item interactions are not interpretable if the user does not directly interact with the item. Another disadvantage of explicit feedback is that it can be faked [7]. For example, on Instagram, it is relatively easy to create bot accounts and like an Instagram picture or inflate follower counts to boost popularity.

Implicit feedback is data collected automatically by observing the behaviour of the user [6]. When the user interacts with the system, for example, information on how long the user has spent looking at an item, is collected. The main advantage of this is that this type of information is abundant. The main disadvantage of implicit feedback is that it is noisy (lot of mixed signals) and requires domain specific interpretation.

Examples of the challenges in interpreting implicit feedback can be found in multiple domains. One example of the domain specificity is a movie recommendation system tracking watch time. Generally longer watch time means that the movie is more engaging. However, that might not always be the case. What if movie was so boring that the user fell asleep during it? Or what if the user likes to have movies in the background playing when they are home alone? Because of the challenges of interpreting implicit feedback data, more research is required to fully utilize this type of data. It is also less researched because of its domain specific nature and the problems with interpretability.

Explicit and implicit feedback have their own pros and cons. Explicit feedback gives clearer signals of user preference and is easier to interpret. Implicit feedback is very abundant and reflects actual user behaviour and is hard to fake deliberately. The Cons of explicit feedback are data sparsity, prone to bias, and susceptible to manipulation. Implicit feedback suffers from ambiguity, lack of negative signals, and the fact that it requires sophisticated modelling to interpret correctly. Table 2 summarises the characteristics of explicit and implicit feedback.

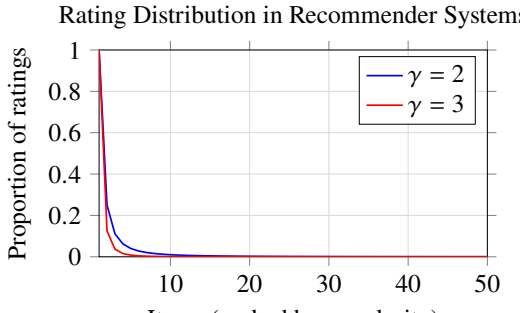
	Explicit feedback	**Implicit feedback**
Accuracy	High	Low
Abundance	Low	High
Context-sensitive	Yes	Yes
Expressivity of user preference	Positive and Negative	Positive
Measurement reference	Absolute	Relative

Table 2: Summary of the characteristics of explicit feedback characteristics [6].

2.2 Challenges in recommender systems

There are several challenges in recommender systems that decrease the performance of a recommender system. Article [7] lists eight unique challenges: data sparsity, scalability, cold start, diversity versus accuracy, vulnerability to attacks, time value, recommendation evaluation, and user interface.

When the data pool is very large in real-world recommender systems, the overlap of two users is often very small or none. This creates a problem called *data sparsity*, where



accurate recommendation is difficult due to a lack of data. Furthermore, even when the average number of evaluations per entity is high, it is distributed among the entities unevenly. Typically, the distribution follows a power law, where the form is $P(k) \sim k^{-\gamma}$ with γ being a positive exponent usually between 2 and 3 [8]. As can be seen in Figure 2, a small minority of items have the majority of ratings. A good recommender system takes this into account because this might cause problems where popular items are suggested more frequently because of the higher representation in the data.

When new users enter the system, there is no interaction data about them. This hinders the performance of the recommender system and causes inaccurate recommendations for new users. This problem is called the *cold start* problem. The cold start problem is typically tackled with methods that combine multiple recommendation methods. Most of the popular and simple-to-implement solutions perform well enough on users that have a good amount of interaction data. Then there are methods that perform well in the cold start cases. A common approach is to use auxiliary information, such as age, location, or sex of the user. The items can also have auxiliary information about them, such as item name, brand and price. Another approach is to use data from other sites to make a recommendation on the current site. This can be done by using cookies to save user preferences in the browser.

Most RS use large datasets, so *scalability* is often a bottleneck in implementations. In practice, this can be solved, for example, by computing recommendations from smaller local samples of the data rather than using global data from the whole data set. Another approach is to compute the recommendation incrementally every time new data arrives.

Good recommendation systems are capable of balancing *diversity and precision*. Diversity is the range of different items that are recommended to the user. On a music streaming platform this would mean that the user is recommended songs vastly different from each other from different genres and artists. This helps with new item type discovery, and the user does not get stuck with a single type of music. The reason why balancing between diversity and precision is difficult is because the metrics might counteract each other. Showing the user a wider range of different items might mean that items that the user absolutely like are not shown.

Another reason why balancing diversity and precision is important is that recommender systems tend to be biased towards popular items. This is because popular items have more connections to users and thus will be recommended more often. Building diversity into the RS also helps with overfitting. With more diverse recommendation, the RS is more generalisable to different domains [9].

Recommender systems are *vulnerable to attacks*. An attack on the recommender system can be a malicious actor trying to steal user information, or it can be an attempt to manipulate the algorithm (shilling attack). User data privacy is relatively well secured, with the biggest vulnerabilities being zero-day exploits (vulnerabilities unknown to users), and careless cyber security practises by developers. Manipulation of the RS, on the other hand, is easier. Users can give fake ratings to try to increase or decrease the popularity of an item. The solution to this problem is to identify the attackers quickly and remove fake ratings.

The interests of users vary over time. Interests can last from a couple of days to years. An example of this is that a user might be interested in buying a tennis racket from an e-commerce site to go to play with their friends. This interest is a short interest because the user only plays

occasionally. On the other hand, they might have a long-term interest in running and buy new shoes every year because the previous shoes are worn. Currently, many recommendation algorithms do not take the decay of interests into account, and it is an ongoing line of research [7].

When optimizing an RS, it is important to have *evaluation metrics* for benchmarking the system. The metrics are not absolute and might vary on different datasets. This makes benchmarking recommender systems challenging and makes comparing different recommendation algorithms difficult. In research, the MovieLens datasets are popular for comparing algorithms with each other. In practice, recommender systems have to be tailored to suit the application at hand. There is no universal recommender systems that is able to perform well in all applications.

A little less technical challenge in RSs is the optimization of the *user interface*. This can be divided into two parts: Firstly, the optimisation of the user interface itself. In the second part, there is the explainability of the recommender system. A common way to display recommendation is with a carousel with the most relevant recommendations being displayed at the top left corner. The further down and right an item is displayed the less likely it is for the user to see it. Explainability is important for the user to trust and use the recommender system more efficiently. Explainability can be implemented by giving reasons why certain items are presented to the user. For example, the used can be shown recommendations in categories like "Because you watched [movie name]" or "Movies directed by [actor name]".

2.3 Recommender systems performance evaluation

When building recommender systems, the goal is to predict unseen ratings as well as possible. To achieve this goal, there is a need to define performance evaluation metrics that quantify the quality of implementations. There are many ways to quantify the quality of a method, and this subsection introduces basics for evaluating RS performance.

There are two notable accuracy metrics for ranking RS accuracy: mean absolute error (MAE) and root mean squared error (RMSE). They are both used to measure the closeness of predicted ratings to real ratings.

MAE and RMSE are defined as

$$\text{MAE} = \frac{1}{|E^P|} \sum_{(i,\alpha)\in E^P} |r_{i\alpha} - \hat{r}_{i\alpha}|$$

$$\text{RMSE} = \left(\frac{1}{|E^P|} \sum_{(i,\alpha)\in E^P} (r_{i\alpha} - \hat{r}_{i\alpha})^2 \right)^{\frac{1}{2}}$$

where $r_{i\alpha}$ is the true rating of object α by user i, $\hat{r}_{i\alpha}$ is the predicted rating, and E^P is the set of hidden user-object ratings in the test set. Lower MAE and RMSE correspond to higher prediction accuracy.

In addition to accuracy metrics, there exist other performance metrics. *Classification accuracy metrics* measure the percentage of items that the RS correctly recommended to the user. The *ranking quality metrics* measure the accuracy of the order of recommended items. *Coverage* calculates the proportion of all items that are recommended when recommendations are calculated. A brief summary of popular recommendation metrics can be found in Table 3.

To benchmark a recommender system, a dataset of ratings is needed. Luckily, there exists multiple high quality open source datasets for benchmarking. The MovieLens data set is one of

Name	Symbol	Preference	Scope	Rank	L
MAE	MAE	Small	Rating accuracy	No	No
RMSE	$RMSE$	Small	Rating accuracy	No	No
Precision	$P(L)$	Large	Classification accuracy	No	Yes
Recall	$R(L)$	Large	Classification accuracy	No	Yes
F_1-score	$F_1(L)$	Large	Classification accuracy	No	Yes
AUC	AUC	Large	Classification accuracy	No	No
NDCG	$NDCG(L)$	Large	Ranking quality	Yes	Yes
Coverage	$COV(L)$	Large	Coverage and diversity	No	Yes

Table 3: A summary of common RS performance metrics. The third column indicates the preference of the metric (e.g. small MAE is better). The last two columns tell whether the method depends on the order of the recommendations and the length of the list of recommendations. Table inspired by article [7].

the most widely used data set in research. The data is gathered from the MovieLens web site and is maintained by a research lab GroupLens from the University of Minnesota. Different versions of the dataset can be found from https://grouplens.org/datasets/movielens/ containing different amounts of ratings, synthetized ratings and datasets with varying time intervals for when they were collected. MovieLens is one of the more popular open source datasets for benchmarking recommender systems, but there are other great ones out there. The GitHub page, [10], has great links other great datasets for benchmarking recommender systems.

2.4 Lessons from the 2006 Netflix prize

In 2006, Netflix organised a recommender system competition. The goal of the competition was to improve their current recommender system, Cinematch. For the competition, Netflix released a dataset containing 100 million anonymous movie ratings. The winning solution was required to improve the RMSE of Cinematch by 10%. The prize for the winner was one million dollars so the stakes were high.

The competition attracted international intrest and over 20 thousand teams signed up from 152 different countries. Despite the massive amount of participants, only 90 teams were able to improve the accuracy by more than 5%. After three years of intense competition, the 10% goal set by Netflix was reached on September 18, 2009, when the team BellKor's Pragmatic Chaos was declared the winner.

The first-year winner BellKor wrote a popular research article on their findings [11]. In the article, the importance of utilising a variety of models is emphasised. In other words, *a complete recommender system should use a combination of different algorithms to combine the strengths of the models*. The best way to choose algorithms to be combined was to use methods that utilise different aspects of the data. This way the system does not have blind spots and is able to create better recommendations.

BellKor's first year winning solution was a weighted linear method where they combined all methods they had researched. The solution used 107 different algorithms, but they stated that the amount of algorithms was unnecessarily large. In their article [11], they mention that the amount was that large because of algorithms being added to the system every time one was researched and tested. The paper published by the 2009 winning team were able to use only 18 algorithms to achieve the 10% prediction improvement required by Netflix to win the grand prize [12].

3 Traditional recommender systems

Traditional recommender systems are typically classified into content-based, collaborative, and hybrid filtering. *Content-based filtering* creates recommendations based on items similar to those that the user preferred in the past. In *collaborative filtering* the user is recommended items that people with similar tastes and preferences like. Lastly, *hybrid methods* combine the strengths of collaborative, content-based filtering, and other methods to create more powerful recommender systems. The taxonomy is visualised in Figure 3.

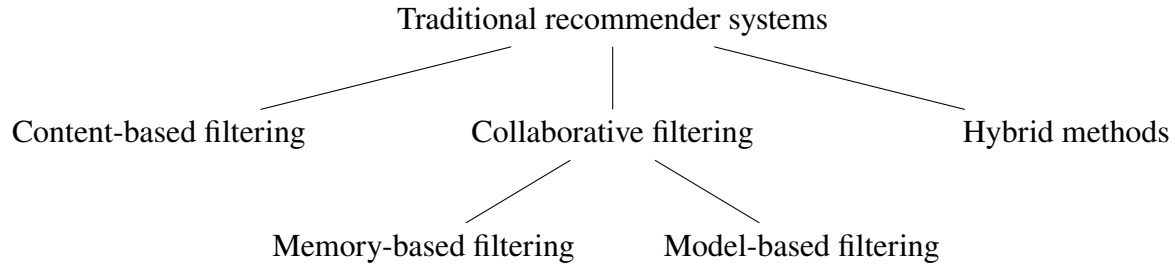


Figure 3: Types of recommender systems

3.1 Content-based filtering

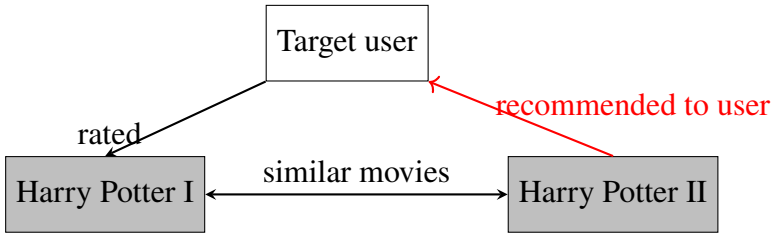


Figure 4: Content-based filtering. The main assumption is that users like similar items that they have liked in the past. Similar items are determined by item attributes.

Content-based filtering assumes that users are interested in similar items that they have liked in the past [13]. This is done by first extracting the information of the item to be recommended. Then the item information is compared to items the user has previously liked. If the item is similar to items the user has previously liked, the item is recommended.

3.2 Collaborative Filtering

The main technique behind collaborative filtering is that item and user similarity is calculated by the collaborative ratings of items [13]. This is why the technique is called collaborative filtering. In short, similar users are found by comparing their rating histories and similar items are found by comparing ratings from users. The way user ratings are used, can be further divided into two classes: *memory-based* and *model-based* techniques. Memory-based CF learns the the similarity of users or items from rating data, and then recommendations are made based on similarity data. Model-based CF uses machine learning models to predict rating data. Memory-based filtering can be further classified into *user-based* and *item-based* filtering based on if the filtering is based on user or item similarity.

3.2.1 Memory-based collaborative filtering

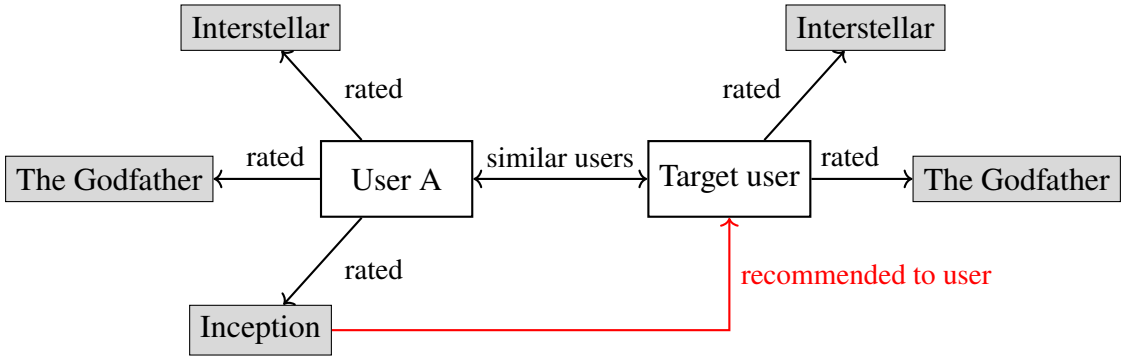


Figure 5: User-based collaborative filtering. Similar users have similar rating histories.

User-based CF uses user similarity to predict ratings. To do this, users are first modelled as vectors based on their rating profiles. Their rating vector will consist of all the ratings they have given to each item. Then these user vectors are compared with a vector similarity metric. Then the k nearest neighbours (the k most similar users) are computed. Then, all the items rated by the nearest neighbours are aggregated into one set and the items rated by the target user are removed. Then, the most frequent items in the set are recommended to the user sorted in decreasing order based on item frequency.

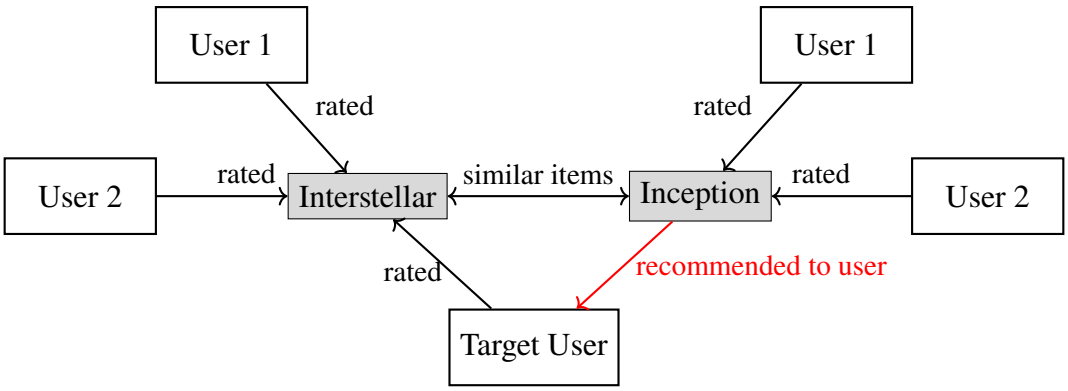


Figure 6: Item-based collaborative filtering. Similar items have similar ratings from the same users.

Item-based CF uses item similarity to predict ratings. The difference with user-based CF is that items are now represented as vectors, where the vector contains all the ratings the item has received. Then similarly to user-based CF, item similarity is calculated between the item vectors. The items most similar to the items the user has rated will then be recommended to the user, sorted in a decreasing order by item similarity.

In memory-based CF, an example of a similarity metric is cosine-based similarity [14]. Cosine-based similarity is defined as the normalized dot product of two vectors, $\text{sim}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||_2 \, ||\mathbf{B}||_2}$, and tells how similar the directions of the vectors are. With cosine-based similarity a high positive number indicates high similarity, a high negative value indicates opposites and zero is neutral.

3.2.2 Model-based collaborative filtering

The name model-based CF comes from the fact that machine learning models are used to predict ratings. Machine learning methods excel in this task because they are great at predicting non-linear relationships in the user-item interaction data. The main advantages of model-based filtering methods are that after training, they require less memory and compute than other methods. Commonly used models include Bayesian models, clustering models, and dependency networks. From these algorithms, classification algorithms can be used as CF models if user ratings are categorical, and regression models and SVD methods can be used for numerical ratings [14].

3.3 Hybrid methods

Hybrid methods combine the strengths of different methods to create even more powerful methods. The best combination is achieved by combining methods that address different aspects of the data [11]. A common example of a hybrid method is weighted hybridization. In this hybrid method, different recommendation algorithms are given importance scores (weights) based on their ability to predict correct ratings. Then the outputs of all recommendation algorithms are weighted based on predefined weights and combined to create recommendations. More popular hybrid methods can be found from Table 4.

Method name	**Description**
Weighted hybridization	The output of different recommender systems are weighted based on their ability predict ratings. Then the weighted results are combined and shown to the user.
Switching hybridization	Recommender systems are switched based on the recommendation scenario.
Mixed hybridization	Multiple recommender systems calculate recommendations. Then the outputs of these systems are mixed and shown to the user.
Cascade hybridization	A primary system creates a list of recommendations. Then a secondary system tweaks the recommendations typically addressing a problem in the primary system.
Feature combination	A secondary RS calculates feature vectors for users or items and enhances the primary system with these features.
Feature augmentation	User or item feature vectors are used as input for another recommender system.
Meta-level	A system replaces the original dataset with augmented data and another then creates recommendations based on this data.

Table 4: Popular hybrid methods [1].

3.4 Advantages and Limitations of traditional recommender systems

Traditional methods have become popular because they are easy to implement and perform well enough for most applications. This is why traditional methods typically serve as a baseline in industry. Although traditional methods are widely used in industry, it is important to understand in which scenarios traditional methods perform well, and in which scenario they tend to struggle. This way appropriate methods can be chosen for each application.

3.4.1 Content-based filtering

The main *advantages* of content-based filtering are good performance with sparse interaction data (immunity to the cold-start problem) [13] and the explainability of recommendations. CBF performs well with sparse data because the method relies only on item attribute data. Of course, the user has to first interact with a few items but then the algorithm is able to recommend similar items based on item profiles. Because CBF uses item similarity to create recommendations, the methods are more explainable. For example, an e-commerce site could have a category of "similar products to [product]".

There are three main *disadvantages* of content-based filtering: limited discovery, feature engineering, and the cold-start problem for items. Because recommendations are based only on the features of items, CBF has a hard time discovering new items [13]. The algorithm will only recommend items similar to items that have been previously liked. The second disadvantage, feature engineering, means that it is difficult to decide the item attributes recommendations should be based on. Often processing these features can also be challenging (e.g. image/video processing). The third weakness is that if an item has little or no information about it, the item will not be recommended to the user.

3.4.2 Collaborative filtering

The main *advantages* of collaborative filtering is its ability to capture complex user preferences and adapt to changing user tastes [13]. This is because recommendation is based on user-item interaction data. A side effect of recommendation being based on interaction data is that CF recommender systems generalize well to multiple domains.

Disadvantages of collaborative filtering are reduced performance on sparse data and scalability issues [4]. Because the recommendation is based on user-item interaction data, users with little to no interaction data struggle to get good recommendations. Data sparsity issues also arise in large item catalogues with many items. In this case, the user-item matrix becomes sparse, leading to difficulties computing similarities due to data sparsity. Scalability issues arise from high computational costs when dealing with large-scale datasets. Because CF uses large amounts of user-item interaction data to calculate similarities, computational costs increase with the amount of data.

3.4.3 Hybrid methods

Hybrid methods typically *perform better* than CF and CBF methods because they combine the strengths of multiple different methods. In this way, disadvantages such as data sparsity issues can be mitigated by adding methods to the system that do not suffer from data sparsity. Hybrid methods also provide flexibility when designing a recommendation system for specific applications. RSs can be optimised to perform ideally in specific applications.

The increased complexity is, on the other hand, is a *disadvantage* of hybrid methods. Increased complexity makes maintaining the system more difficult and also increases computational costs. Perfecting hybrid methods also requires a lot of tuning and optimizing to achieve the optimal performance, and this takes a lot of time, effort and skill [13].

4 Knowledge graphs

Knowledge graphs are added to recommender systems to add auxiliary data to help with recommendation. The concept of knowledge graphs was developed in the 1980s when KGs were integrated into expert systems in medical and social sciences [15]. Back then, knowledge graphs were simpler consisting of fewer entity and relation types. In 2012 Google broadened the concept to the linguistic and logical domains. At Google, they successfully applied KGs in their web search system.

After Google popularised knowledge graphs, KGs have gained attention in applications that need to exploit diverse, dynamic, and large-scale collections of data. KGs are a great tool for these applications because they can store semantic information. Another benefit of KGs is that KGs can help postpone the definition of a data schema. This means that the data can evolve more flexibly [16].

There is no widely agreed-upon definition of a knowledge graph. Generally, a knowledge graph can be described as a heterogeneous information network (contains multiple types of entities and relations). A heterogeneous information network (HIN) is a rigorously defined concept and often functions as a baseline when defining KGs. HINs can be defined as in article [17]:

Definition 4.1 (Heterogeneous information network) *A heterogeneous information network is a directed graph $G = (V, E)$ with an entity-type mapping function $\phi : V \rightarrow \mathcal{A}$ and a relation-type mapping function $\psi : E \rightarrow \mathcal{R}$. Each entity $v \in V$ belongs to an entity type $\phi(v) \in \mathcal{A}$, and each relation $e \in E$ belongs to a relation type $\psi(e) \in \mathcal{R}$. In addition, the number of entity types $|\mathcal{A}| > 1$ and/or the number of relation types is $|\mathcal{R}| > 1$.*

Figure 7: Heterogeneous information graph representation. Inspired by [18]

The main difference between HINs and KGs is the scale and diversity of the types of nodes and entities [19]. Most HINs have relatively simple schemas, which means that the graphs have a small number of types of entities and relations. On the other hand, KGs have a much richer network schema, with hundreds or even thousands of entity and relation types. The paths in the graph create facts such as $Helsinki \xrightarrow{capital\ of} Finland \xrightarrow{next\ to} Russia$. These paths store complex connections and help reason facts such as the fact that the city Helsinki is near Russia. An example of a knowledge graph can be found in Figure 8.

This thesis uses the following definition for knowledge graphs from article [20]:

Definition 4.2 (Knowledge graph) *A knowledge graph is a directed graph $\mathcal{G} = \{\mathcal{E}, \mathcal{R}, \mathcal{F}\}$, where $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{F}$ are the sets of entities, relations and facts. A fact f is denoted as a triple (h, r, t), where $h \in \mathcal{E}$, $r \in \mathcal{R}$ and $t \in \mathcal{E}$ are the head entity, relation, and tail entity of a fact $f \in \mathcal{F}$.*

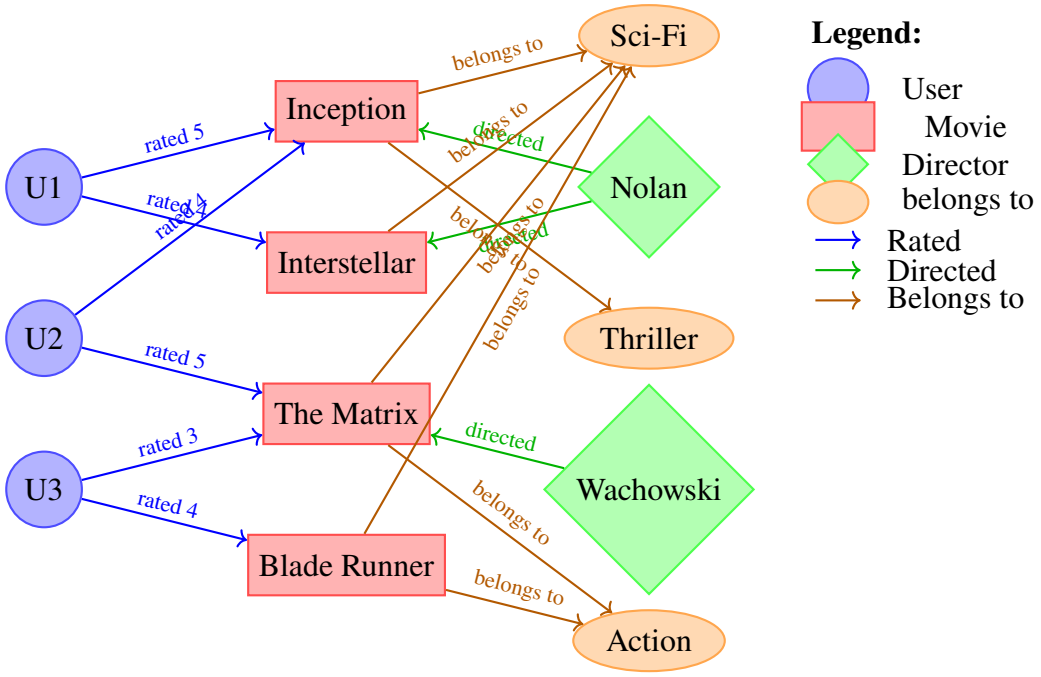


Figure 8: Knowledge graph for movie recommendation. The graph shows relationships between users (circles), movies (rectangles), directors (diamonds), and genres (ellipses).

5 Knowledge graph modelling methods

The methods for modelling knowledge graphs in RSs to are divided into three categories: *path-based*, *embedding-based* and *unified* methods (see Figure 9). Path-based methods use graph algorithms to directly explore the graph and find connection patterns between users and items. Embedding-based methods embed the graph to a low-dimensional space so that machine learning methods can be used. Lastly, unified methods combine path-based and embedding-based techniques to combine the strengths of both methods. Most unified methods involve deep learning and graph neural networks.

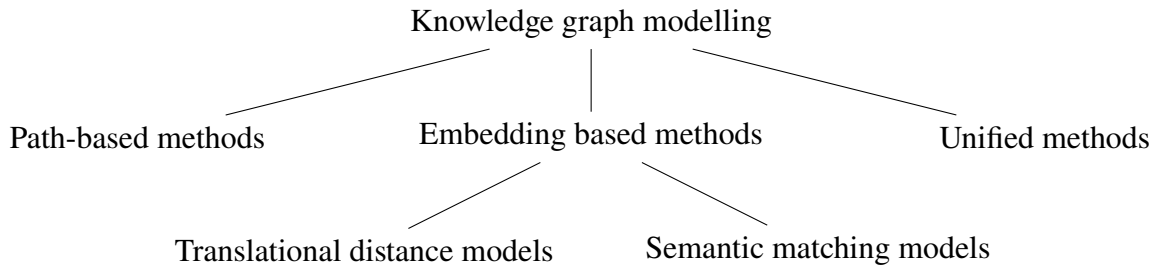


Figure 9: Knowledge graph taxonomy.

5.1 Path-based Methods

Path-based methods use connective patterns in the knowledge graph to create recommendations. These connective patterns are analysed with different graph analysis algorithms. These connective patterns are called *meta-paths* and are defined in [17] as the following:

Definition 5.1 (Meta-path) *Let a graph be defined as $\mathcal{G} = \{\mathcal{E}, \mathcal{R}\}$, where $e_i \in \mathcal{E}$ represents an entity (node) in the set of all entities (nodes) and $r_i \in \mathcal{R}$ represents a relation in the set of all relations. Then a* ***meta-path*** $\mathcal{P} = e_0 \xrightarrow{r_1} e_1 \xrightarrow{r_2} \ldots \xrightarrow{r_k} e_k$ *is a path between entities e_0 and e_k defined in the graph $\mathcal{G}$.*

Meta-paths can be used to calculate entity similarity. Similarity metrics are useful for recommendation because recommendation can then be done in a similar manner than in traditional methods; Two similar users will probably enjoy similar items and a user will enjoy similar items to the items they have enjoyed in the past. The most common metric for calculating entity similarity in a graph is a method called *PathSim* introduced by Sun, Han, Yan, *et al.* [21] in 2011:

Definition 5.2 (PathSim) *Let $p_{e_i \rightsquigarrow e_j}$ be a path between entities e_i and e_j and $\mathcal{P}$ a symmetric path (a path that is the same traversed in both directions). Then the path-based similarity score is*

$$s_{x,y} = \frac{2 \times |\{p_{x \rightsquigarrow y} : p_{x \rightsquigarrow y} \in \mathcal{P}\}|}{|\{p_{x \rightsquigarrow x} : p_{x \rightsquigarrow x} \in \mathcal{P}\}| + |\{p_{y \rightsquigarrow y} : p_{y \rightsquigarrow y} \in \mathcal{P}\}|}$$

PathSim can be applied to create latent representations of users and items. The similarity score can be used to force the latent representations close to each other by using a norm. When the similarity score is multiplied by the difference of the latent representation, the difference is minimized. This concept is formally written in [17] as:

Definition 5.3 (Entity similarity) *With $||\cdot||_F$ denoting the matrix Forbenious norm, $\Theta = [\theta_1, \theta_2, \ldots, \theta_L]$ denoting the weights of each meta path and $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m]$ denoting latent vectors of all users, $s^l_{i,j}$ denoting the similarity score (5.2) of users i and j on meta-path l and $\mathbf{V} = [\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}]$ denoting the latent vectors of all items, the similarity scores between entities are:*

User-User similarity

$$\min_{U,\Theta} \sum_{l=1}^{L} \theta_l \sum_{i=1}^{m} \sum_{j=1}^{m} s^l_{i,j} \|\mathbf{u}_i - \mathbf{u}_j\|^2_F$$

Item-Item similarity

$$\min_{V,\Theta} \sum_{l=1}^{L} \theta_l \sum_{i=1}^{n} \sum_{j=1}^{n} s^l_{i,j} \|\mathbf{v}_i - \mathbf{v}_j\|^2_F$$

User-Item similarity

$$\min_{U,V,\Theta} \sum_{l=1}^{L} \theta_l \sum_{i=1}^{m} \sum_{j=1}^{n} (\mathbf{u}_i^T \mathbf{v}_j - s^l_{i,j})^2$$

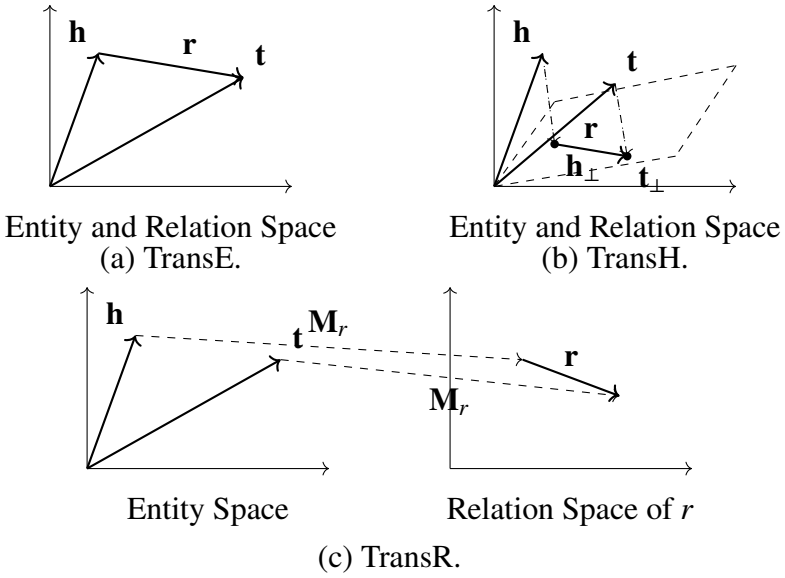
5.2 Embedding-based methods

Embedding-based methods encode the structural information of a KG into a low-dimensional vector space. When computing an embedding, the process can be divided into two steps. The first step is defining a scoring/loss function $f_r(h, t)$, where h is a head entity, t is a tail entity and r is the relation between the head and tail entity. The second step is to minimize the loss function

$f_r(h,t)$. The vector embedding that minimizes the loss function preserves as much structural information as possible.

After the loss function is minimized, it can be used in reverse to determine true relations from false ones. If the loss function is small for a triple (h,r,t), the triple is more likely to be a true triple of the original KG. If the loss function is large for a triple, the triple is likely to be false. Using this logic, the knowledge graph can be reconstructed again from the embedding using the loss function by setting a decision boundary for true and false triples.

Embedding-based methods can be divided into two classes based on how the loss function models the correctness of a triple: *translation distance models* and *semantic matching models*.



Entity and Relation Space
(a) TransE.

Entity and Relation Space
(b) TransH.

Entity Space

Relation Space of r

(c) TransR.

Figure 10: Illustrations of translation distance models TransE, TransH, and TransR [5].

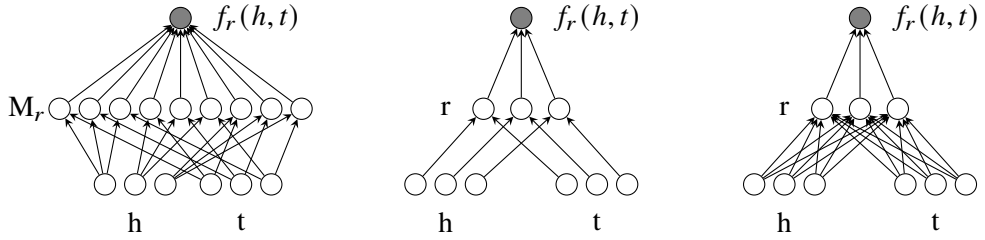
As their name states, **translation distance models** translate graph information into a low-dimensional space using distance. In general, the distance is defined as the distance between two entities, usually after some sort of a transition. To give examples of this type of scoring function, the thesis introduces three popular translation distance models TransE, TransH and TransR. Simple visualisations of the scoring functions can be found in Figure 10.

The most simple method is $TransE$. Its loss function is defined as $f_r(h,t) = ||h + r - t||_{1/2}$, where (h,r,t) are the head entity, relation and tail entity, respectively [5]. The idea behind $transE$ is that the translation r from the head entity h to the tail entity t is as accurate as possible (loss function small). Due to its simplicity, $TransE$ is not used as the state-of-the-art method. The main appeal is that it is an intuitive method for understanding translation distance models.

$TransH$ complicates the head and tail entity representations by projecting the head and tail entity to a hyperplane where the relation resides. The hyperplane is described by its normal vector w_r. Then the projections of h and t are $\text{proj}_W(h) = h - w_r^T h w_r$ and $\text{proj}_W(t) = t - w_r^T t w_r$. With these projections, the loss function is defined as $f_r(h,t) = ||\text{proj}_W(h) + r - \text{proj}_W(t)||_2^2$ [5].

$TransR$ differs from $TransE$ and $TransH$ by expressing relations in a parallel space. In practice, when calculating the loss function, the head and tail entities h and t have to be first mapped to the relation space by transformation M_r resulting in $h_r = M_r h$ and $t_r = M_r t$. The loss function is then defined as $f_r(h,t) = ||h_r + r - t_r||_2^2$ [5].

Semantic matching models aim to learn embeddings that capture semantic compatibility



(a) RESCAL. (b) DistMult. (c) HolE.

Figure 11: Simple illustrations of semantic matching models RESCAL, DistMult, and HolE [5].

between entities and relations, using neural architectures. These architectures assign high scores to semantically coherent triples $(h, r, t) \in \mathcal{F}$ and low scores to semantically false triples. To learn the semantics of entities and relations, a scoring function is used. Unlike translation distance models, the scoring function is maximized. To give practical examples, the thesis introduces three SMMs: $RESCAL$, $DistMult$ and $HolE$. Simple illustrations of these methods can be found in Figure 11.

$RESCAL$ uses a matrix M_r for the embedding. The main benefit for this is that different dimensions of the head and tail entities are able to interact with each other. Because of this, $RESCAL$ is able to capture the relation distance [18]. With the head and tail entities (h, t) and the relation matrix M_r, the scoring function is defined as $f_r(h,t) = h^T M_r t = \sum_{i=0}^{d-1} \sum_{i=0}^{d-1} [M_r]_{ij} \cdot [h]_i \cdot [t]_j$.

$DistMult$ improves slightly on time and space costs compared to $RESCAL$. This is achieved by only using the diagonal elements of M_r, $\text{diag}(M_r) = r$. Other than only using the diagonal values of matrix M_r, the scoring function is the same with $RESCAL$. It is defined as $f_r(h,t) = h^T \text{diag}(r) t = \sum_{i=0}^{d-1} [r]_i \cdot [h]_i \cdot [t]_i$ [18].

$HolE$ uses a *circular correlation operation* to combine the head and tail entities. The circular correlation operator is defined as $h \star t \in \mathbb{R}^d$, and in practise, it takes the sums along the diagonals of the outer product of two vectors. The circular correlation operator is used to capture edge direction [18]. With the circular correlation operation defined, the scoring function is defined as $f_r(h,t) = r^T(h \star t) = \sum_{l=0}^{d-1} [r]_l \sum_{k=0}^{d-1} [h]_k \cdot [t]_{(k+l) mod d}$.

Method	Category	Entity Embedding	Relation Embedding	Loss Function
TransE	TDM	$h, t \in \mathbb{R}^d$	$r \in \mathbb{R}^d$	$\|h + r - t\|_{1/2}$
TransH	TDM	$h, t \in \mathbb{R}^d$	$r, w_r \in \mathbb{R}^d$	$\|(h - w_r^T h w_r) + r - (t - w_r^T t w_r)\|_2^2$
TransR	TDM	$h, t \in \mathbb{R}^d$	$r \in \mathbb{R}^k, M_r \in \mathbb{R}^{k \times d}$	$\|M_r h + r - M_r t\|_2^2$
RESCAL	SMM	$h, t \in \mathbb{R}^d$	$M_r \in \mathbb{R}^{d \times d}$	$h^T M_r t$
DisMult	SMM	$h, t \in \mathbb{R}^d$	$r \in \mathbb{R}^d$	$h^T diag(r) t$
HolE	SMM	$h, t \in \mathbb{R}^d$	$r \in \mathbb{R}^d$	$r^T(h \star t)$

Table 5: Summary of KGE techniques. TDM stands for translation distance models and SMM for semantic matching models. Table from article [5]

5.3 Unified methods

Similar to hybrid methods in subsection 3.3, unified methods combine the strengths of embedding and path-based methods. In practise, unified methods use both meta-paths and they propagate the embedding space to create recommendations. Implementations are typically based on

frameworks that combine graph neural networks and attention mechanisms [18]. The main idea behind GNNs is to enhance node representations with message propagation. This means that the information of the neighbouring nodes is encoded into a vector representation of the nodes of the graph.

6 Recommendation with knowledge graphs

While the previous section discussed how knowledge graphs are modelled, this section will discuss how the models of KGs can be used to generate recommendations.

6.1 Path-based KG recommender methods

Path-based methods use meta-paths to create recommendations. These meta-paths connect two nodes together through a series of relations. A practical example is a movie recommender system: Let a movie domain KG have nodes for users (U), movies (item) (I) and directors (D). Then recommendation could be configured to mine meta-paths $UIDI$ and $UIUI$ for user-item-director-item or user-item-user-item paths. Using preconfigured meta-paths, the recommendations become more explainable. Different meta-paths are shown in Table 6.

Table 6: Examples of meta-paths in a movie recommender system. U is user, M is movie, D is director and G is genre.

Meta-path	Explanation
$UMUM$	"Movies that similar users like"
$UMDM$	"Movies by director D"
$UMGM$	"Movies in genre G"
$UMUMGM$	"Movies in genres that similar users like"

Among path-based approaches, *Hete-MF* [22] and *Hete-CF* [23] are two early approaches. The Hete-MF approach derives L distinct meta-paths to compute similarity measures between items within each pathway. This method incorporates item-based regularization into a weighted non-negative matrix factorization framework, enhancing the low-dimensional representations of both users and items. This then further improves the quality of recommendations.

In contrast, Hete-CF determines user preferences for unrated items by incorporating multiple similarity components (user-user, item-item, and user-item relationships) as regularization constraints within its model. The superior performance of Hete-CF compared to Hete-MF is due to its comprehensive utilization of various similarity metrics in the recommendation process [17].

6.2 Embedding-based KG recommender methods

A KG embedding-based recommendation system generally consists of three parts: a KG, a graph embedding module, and a recommendation module [5]. The three components enable recommendation based on the data in the knowledge graph. The KG stores auxiliary information about items and users. The graph embedding module creates the embedding and the recommendation module uses the embedding to create recommendations. Then lastly, the recommendations are displayed to the user.

To create recommendations based on graph embeddings, machine learning methods are used to analyse the embedding. This is typically modelled by a preference score $\hat{y}_{i,j} = f(\mathbf{u}_i, \mathbf{v}_j)$ [5], where the function $f : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$, $\mathbf{u}_i$ is the user latent vector and $\mathbf{v}_j$ is the item latent vector. The function f for calculating the preference score is typically a vector similarity calculation

because the loss functions that create the embeddings group similar vectors together. Examples of vector similarity metrics are inner product and deep neural networks [5].

The scoring functions of embeddings can also be used for recommendation. If $d(\cdot,\cdot)$ is a distance function between two vectors, link predictions between users and items can be calculated by $d_{ij}(u_i + r, v_j)$. The smaller the distance, the more likely the relation r is true in the original KG. This idea is visualized in Figure 12. The figure shows an embedding of a movie KG, where true entity-relation-entity connections have a small error in the Euclidean distance between the entities.

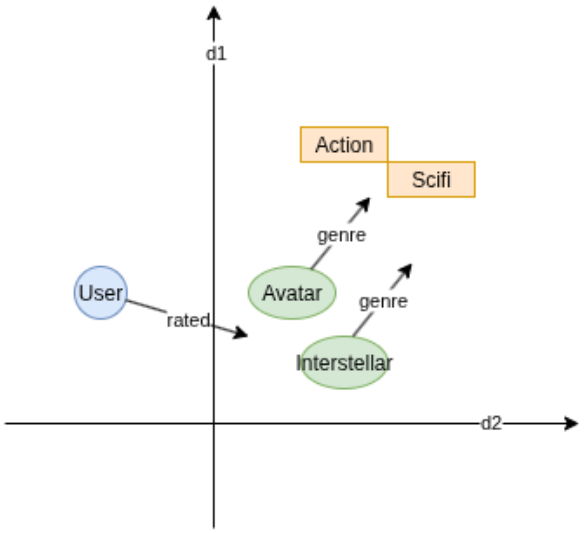


Figure 12: A 2-dimensional translation distance embedding. Avatar and Interstellar are likely similar movies because they are close to each other. The user is likely to like both of the movies.

6.3 Unified KG recommender methods

Unified methods mainly consist of graph neural networks [18] and these methods use message passing (path propagation) to improve node embeddings in the KG. The node embedding vectors will contain the semantic information of the node and its neighbours, so the embedded node representations can be processed with machine learning. The machine learning done to the embedded vectors is similar to the methods introduced in subsection 6.2.

6.4 Advantages of knowledge graph enhanced recommender systems

Compared to traditional recommender systems, KG-based systems have *better representation of data* [4]. This is because traditional recommender systems base recommendations only on user-item interaction data. KG-based systems also incorporate auxiliary information about users and items. This information can be user-side or items-side information. User-side auxiliary information could be user age, address, occupation and gender. Examples of item-side auxiliary information are brand, category and price.

Another advantage of integrating KGs into a RS is that KG-based recommendations are *more explainable* than traditional recommender systems. This is especially evident in path-based methods where recommendations are based on the connections between users and items in the KG. When recommendations are explainable, the RS is more trustable and transparent and it is

easier for the user to understand what data is being collected from them and where it is being used.

6.5 Limitations of knowledge graph enhanced recommender systems

Constructing and integrating a KG into a recommender system is challenging. First, a knowledge graph has to be acquired. There are two main ways to do this: The first is to create a custom graph only used for a specific domain. Creating a custom graph requires a system that can gather data and store it in graph form. The second approach is to use an open source graph that is readily available on the internet. For integrating an open source graph into the RS, there are challenges in mapping the nodes in the KG to nodes in the RS. The mapping process typically introduces biases to the KG because of items having different names in the RS than in the KG or synonyms and homonyms.

The way a KG is constructed can introduce biases. The paper by Voit and Paulheim [24] found that knowledge graphs deliver different recommendations based on the language version of the graph. In the article they used different language versions of DBpedia, which is a KG constructed from Wikipedia data. For example, the article found that recommendations based on the English KG version were more likely to recommend English movies.

Compared to traditional recommender systems, adding a KG recommendation module will increase computational costs, memory requirements and system complexity [4]. Higher computational and memory requirements might hinder scalability, and an increase in system complexity makes maintaining it more laborious.

6.6 Open source knowledge graphs

Knowledge graphs can be divided into two categories based on their scope: cross-domain KGs and domain-specific KGs [17]. Cross-domain KGs are typically found in application scenarios that contain cross-disciplinary information. An example of a cross-domain RS is a search engine. Examples of cross-domain knowledge graphs are *freebase*, *Wikidata*, and *DBpedia*. A popular domain-specific data set is *Bio2RDF* [17]. They provide linked data for life sciences.

Dataset	Website
DBpedia	https://www.dbpedia.org/resources/knowledge-graphs/
Freebase	https://developers.google.com/freebase/
Wikidata	https://www.wikidata.org/wiki/
Bio2RDF	https://download.bio2rdf.org/files/release/3/release.html

Table 7: Datasets with general knowledge when originally released. Table inspired by article [25].

7 Conclusions

Knowledge graph-based recommender systems solve data sparsity issues and are more explainable than traditional recommender systems. The disadvantages of knowledge graph-based recommender systems are that the graphs are difficult to construct and that they increase model complexity and computational requirements. In practise, the most performative recommender systems are constructed by combining multiple different methods that use different aspects of the data.

8 Summary

The thesis investigated how knowledge graphs can benefit recommender systems. This was done by first reviewing different types of recommender systems. The three types were content-based, collaborative and hybrid filtering. Content-based filtering detects similar items by comparing latent vectors of the items. Collaborative filtering creates recommendations based on collaborative data between users and items. Unified methods combine the strengths of different methods to create more performant methods.

Knowledge graphs can be used as auxiliary information in recommender systems. Methods for modelling knowledge graphs in recommender systems are path-based, embedding-based and unified methods. Path-based methods create recommendations based on paths in the knowledge graph. Embedding-based methods embed the knowledge graph into a low-dimensional space and unified methods combine the strengths of both path-based and embedding-based methods to create more accurate recommender systems.

Knowledge graph-based recommender systems solve data sparsity and explainability issues. They do this by using auxiliary information, in addition to user-item interaction data. Knowledge graphs can be created from scratch, or one can use free open-source graphs. Path-based methods especially create human understandable recommendations utilizing meta-paths. Embedding-based methods are able to encode the graph information as vectors so that machine learning methods can be used to create recommendations.

9 Future directions

A big problem of KGs is that they are expensive to generate. A proposed method for automatically generating KGs is *word embedding techniques* used in large language models to learn the semantic meanings of words [18]. Word embedding techniques take sentences as input and create vector embeddings based on the semantic similarities of words in the given input text. For example, the words subway and metro would be assigned similar embedding vectors because they appear in similar contexts. Additionally, words with similar relationships, such as king and queen and husband and wife, will have similar relation embeddings. A popular method for embedding words is word2vec [26].

Because constructing knowledge graphs is expensive, so is updating and maintaining them. All large-scale open-source KGs are static graphs, which means they do not update their information online. This means that static KGs are biased towards older items because they are more connected in the graph. *Introducing dynamicity* would remove these biases. Applying dynamic knowledge graphs to recommender systems would help remove time-dependent biases.

Article [27] questions the need for all relations in the KG. The article does this by comparing

the performance of recommender systems in three cases. The first case is the default case with the original KG. The second case is with false knowledge in the knowledge graph. False knowledge, in their study, means incorrect relations. The third case is with decreasing knowledge, meaning that some relations in the graph were deleted. Then these three states of the knowledge graph were benchmarked in both the cold-start and the default case.

The researchers found that decreasing authentic knowledge in the KG does not significantly reduce the performance of the recommender system. This could be caused by noisy knowledge graph information. The KGs used in the study were large open source KGs and were not optimized for the recommendation task at hand. *Knowledge graph pruning* could be used to reduce the amount of redundant relations in the graph to save both compute time and memory requirements.

References

[1] D. Roy and M. Dutta, "A systematic review and research perspective on recommender systems", *Journal of Big Data*, vol. 9, no. 1, p. 59, 2022.

[2] E. Brynjolfsson, Y. J. Hu, and M. D. Smith, "Consumer surplus in the digital economy: Estimating the value of increased product variety at online booksellers", *Management Science*, vol. 49, no. 11, pp. 1580–1596, 2003.

[3] D. Lee and K. Hosanagar, "Impact of recommender systems on sales volume and diversity", 2014.

[4] S. R. Jetti and M. K. Prasad, "Knowledge graphs and neural networks in recommendation systems: A comprehensive survey and future directions", pp. 1163–1170, 2025.

[5] J.-C. Zhang, A. M. Zain, K.-Q. Zhou, X. Chen, and R.-M. Zhang, "A review of recommender systems based on knowledge graph embedding", *Expert Systems with Applications*, vol. 250, p. 123 876, 2024.

[6] G. Jawaheer, M. Szomszor, and P. Kostkova, "Comparison of implicit and explicit feedback from an online music recommendation service", Sep. 2010.

[7] L. Lü, M. Medo, C. H. Yeung, Y.-C. Zhang, Z.-K. Zhang, and T. Zhou, "Recommender systems", *Physics reports*, vol. 519, no. 1, pp. 1–49, 2012.

[8] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks", *Rev. Mod. Phys.*, vol. 74, pp. 47–97, 1 Jan. 2002.

[9] M. Kunaver and T. Požrl, "Diversity in recommender systems–a survey", *Knowledge-based systems*, vol. 123, pp. 154–162, 2017.

[10] A. F. d. Costa, "Datasets for recommender systems: A repository of topic-centric public data sources in high quality for recommender systems", *GitHub repository*, 2024, Accessed: 2025-08-12.

[11] R. M. Bell and Y. Koren, "Lessons from the netflix prize challenge", *Acm Sigkdd Explorations Newsletter*, vol. 9, no. 2, pp. 75–79, 2007.

[12] A. Töscher, M. Jahrer, and R. M. Bell, "The bigchaos solution to the netflix grand prize", *Netflix prize documentation*, pp. 1–52, 2009.

[13] T. M. Al-Hasan, A. N. Sayed, F. Bensaali, Y. Himeur, I. Varlamis, and G. Dimitrakopoulos, "From traditional recommender systems to gpt-based chatbots: A survey of recent developments and future directions", *Big Data and Cognitive Computing*, vol. 8, no. 4, p. 36, 2024.

[14] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques", *Advances in artificial intelligence*, vol. 2009, no. 1, p. 421 425, 2009.

[15] S. S. Nurdiati and C. Hoede, "25 years development of knowledge graph theory: The results and the challenge", 2008.

[16] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. Reutter, and D. Vrgoč, "Foundations of modern query languages for graph databases", *ACM Comput. Surv.*, vol. 50, no. 5, Sep. 2017.

[17] Q. Guo, F. Zhuang, C. Qin, *et al.*, "A survey on knowledge graph-based recommender systems", *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 8, pp. 3549–3568, 2020.

[18] A. Hogan, E. Blomqvist, M. Cochez, *et al.*, "Knowledge graphs", *ACM Computing Surveys (Csur)*, vol. 54, no. 4, pp. 1–37, 2021.

[19] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Heterogeneous information networks: The past, the present, and the future", *Proceedings of the VLDB Endowment*, vol. 15, no. 12, 2022.

[20] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition, and applications", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 2, pp. 494–514, Feb. 2022.

[21] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks", *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 992–1003, 2011.

[22] X. Yu, X. Ren, Q. Gu, Y. Sun, and J. Han, "Collaborative filtering with entity similarity regularization in heterogeneous information networks", 2013.

[23] C. Luo, W. Pang, Z. Wang, and C. Lin, "Hete-cf: Social-based collaborative filtering recommendation using heterogeneous relations", pp. 917–922, 2014.

[24] M. M. Voit and H. Paulheim, "Bias in knowledge graphs – an empirical study with movie recommendation and different language editions of dbpedia", 2021.

[25] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition, and applications", *IEEE transactions on neural networks and learning systems*, vol. 33, no. 2, pp. 494–514, 2021.

[26] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space", *arXiv preprint arXiv:1301.3781*, 2013.

[27] H. Zhang, D. Wang, Z. Sun, *et al.*, "Does knowledge graph really matter for recommender systems?", *CoRR*, 2024.