



Markus Reiter-Haas, Bsc

# **Evaluation of Job Recommendations for the Studo Jobs Platform**

## **Master's Thesis**

to achieve the university degree of  
Diplom-Ingenieur (Dipl.-Ing.) equivalent to the Master of Science (MSc)  
Master's degree programme: Computer Science

submitted to

**Graz University of Technology**

Supervisor

Ass.Prof. Dipl.Ing. Dr.techn. Elisabeth Lex

Institute of Interactive Systems and Data Science  
Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Graz, January 2020

This document is set in Palatino, compiled with [pdfL<sup>A</sup>T<sub>E</sub>X2e](#) and [Biber](#).

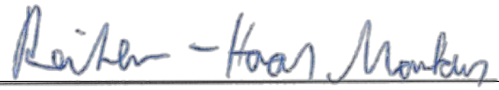
The L<sup>A</sup>T<sub>E</sub>X template from Karl Voit is based on [KOMA script](#) and can be found online: <https://github.com/novoid/LaTeX-KOMA-template>

## Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

January 23, 2020

Date

A handwritten signature in blue ink, reading "Reiter-Haar Markus", written over a horizontal line.

Signature



# Abstract English

Student job recommendations deal with the data sparsity and cold-start problem. The cold-start problem is very severe in the job market since there are frequent changes in both available job vacancies and job seekers. The data sparsity is also worse for students since they lack substantial work experience. Additionally, job postings are usually just unstructured textual descriptions. In this thesis, we incorporate the latent features of textual job descriptions into a recommender system. We achieve this by transforming the textual content into embeddings that capture the meaning of the jobs. Additionally, we apply a time decay equation from cognitive sciences to consider frequency and recency of job interactions. Our aim is to further improve the recommendations in the job domain. We evaluate the recommendations in an offline study on accuracy, beyond-accuracy, and also runtime performance. Furthermore, we examine different configurations and combinations of algorithms in a hybrid setting in order to find the best combination. We carry this out on the Austrian student job platform Studo Jobs in online evaluations for two different scenarios. A vector-based approach with a focus on recency works best for recommendations shown next to the job details, whereas, for recommendations shown on the home view, a focus on frequency is better. For both scenarios, the best combination is achieved by including collaborative filtering in a hybrid manner. The best combinations were rolled out on the Studo Jobs platform and provide recommendations in real-time.



# Abstract Deutsch

Jobempfehlungen für Studierende behandeln das Datenknappheits- und das Kaltstartproblem. Das Kaltstartproblem ist auf dem Arbeitsmarkt sehr gravierend, da es zu häufigen Veränderungen, sowohl bei den verfügbaren Stellenangeboten als auch bei den Arbeitssuchenden, kommt. Die Datendichte ist auch bei Studierende geringer, da sie keine ausreichende Berufserfahrung haben. Zusätzlich sind Stellenausschreibungen in der Regel nur unstrukturierte Textbeschreibungen. In dieser Abschlussarbeit integrieren wir die latenten Merkmale von textuellen Stellenbeschreibungen in einen Recommender System. Dies erreichen wir, indem wir den Textinhalt in Einbettungen umwandeln, welche die Bedeutung der Stellenangebote erfassen. Darüber hinaus wenden wir eine Zeiterfallsgleichung aus den Kognitionswissenschaften an, um die Häufigkeit und Aktualität von Job-Interaktionen zu berücksichtigen. Unser Ziel ist es, die Empfehlungen der Job-Domäne weiter zu verbessern. Wir werten die Empfehlungen in einer Offline-Studie auf Genauigkeit, Jenseits-Genauigkeit und Laufzeitleistung aus. Außerdem untersuchen wir verschiedene Konfigurationen und Kombinationen von Algorithmen in einer hybriden Umgebung, um die beste Kombination zu finden. Wir führen dies auf der österreichischen Jobplattform für Studierende, namens Studo Jobs, in Online-Auswertungen für zwei verschiedene Szenarien aus. Ein vektorbasierter Ansatz mit Fokus auf Aktualität funktioniert am besten für die gezeigten Empfehlungen neben den Jobdetails. Für Empfehlungen auf der Startseite ist ein Fokus auf Häufigkeit besser. Für beide Szenarien wird die beste Kombination erreicht, indem kollaboratives Filtern auf hybride Weise einbezogen wird. Die besten Kombinationen wurden auf der Studo Jobs Plattform ausgerollt und liefern Empfehlungen in Echtzeit.





# Acknowledgements

I would like to thank the Moshbit GmbH and its founders to enable me to write such an interesting thesis about the job platform. Julian Kainz (CEO) for not only being open-minded about research in a small company, but very encouraging about it. Valentin Slawicek (CTO), who always has an open ear for problems and providing the corresponding solutions. Our designer, Stefanie Horvath, for the great design of the job platform (and thus also my screenshots) and for polishing my presentations for conferences among others. Manuel Schmölder (CSO) and his legendary sales team for their success, which provided me with more data to work with. Chris Lanz, who regularly sends me interesting articles to read about AI. My colleague David Wittenbrink, which is responsible for the UI of the job platform and I worked closely with when integrating the recommendations, which he always sees as a high priority. Zoltán Sasvári which helped with architectural questions and on the data management tasks. And the rest of the Talto team for the good teamwork.

I would like to thank my supervisor Elisabeth Lex for her long-time support and short response time while writing the master thesis. Furthermore, I would like to thank my co-supervisor and mentor Emanuel Lacic, which provided me with a constant stream of ideas, insights, and improvements suggestions throughout the whole time. Moreover, I would like to thank the Know Center GmbH, especially the Social Computing team, for the fruitful collaboration.

Additionally, I would like to thank both companies once more, as well as the Graz University of Technology, for the opportunity to write research papers during my master thesis, which strengthened my research experience.

Finally, I would like to thank my family for their patience and support during this time of writing, as well as my girlfriend, Helena Adam, who also helped with proofreading the content.



# Contents

<b>Abstract</b>	<b>v</b>
<b>Acknowledgements</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Research Questions . . . . .	3
1.2 Structure . . . . .	4
<b>2 Related Work</b>	<b>5</b>
2.1 Recommendation Approaches . . . . .	5
2.2 Job Recommender Systems . . . . .	8
2.3 Deep Learning Approaches for NLP . . . . .	9
2.4 Deep Learning Recommender Systems . . . . .	10
2.5 Summary . . . . .	11
<b>3 Job Recommendation Algorithms</b>	<b>13</b>
3.1 Content Based Filtering . . . . .	13
3.2 Collaborative Filtering . . . . .	14
3.3 Doc2Vec . . . . .	15
3.3.1 Last Vector . . . . .	16
3.3.2 Average Vectors . . . . .	17
3.3.3 Base-Level Learning . . . . .	17
3.3.4 User2Vec . . . . .	18
3.4 Hybrid Recommendations . . . . .	18
3.4.1 Mixed Hybrids . . . . .	19
3.4.2 Weighted Hybrids . . . . .	19
<b>4 Studo Jobs Platform</b>	<b>21</b>
4.1 Studo App . . . . .	21
4.2 Studo Jobs . . . . .	22
4.3 Recommendation Scenarios . . . . .	24
4.3.1 Infobox Scenario . . . . .	24
4.3.2 Home Scenario . . . . .	25

## Contents

4.4	Dataset	25
4.4.1	Job Ads	26
4.4.2	CVs	29
4.4.3	Interactions	30
4.4.4	Data Processing and Enrichment	32
<b>5</b>	<b>Implementation Details</b>	<b>33</b>
5.1	Architecture	33
5.2	Integration	34
5.2.1	Data Ingestion	34
5.2.2	Feedback Loop	35
5.2.3	A/B Test Workflow	36
5.2.4	Realtime Recommendations	37
5.3	Embedding Training	38
5.3.1	Parameter Tuning	38
5.3.2	Embedding Analysis	38
<b>6</b>	<b>Evaluation Protocol</b>	<b>41</b>
6.1	Offline Protocol	41
6.2	Online Protocol	46
<b>7</b>	<b>Results</b>	<b>49</b>
7.1	Offline Evaluation	49
7.1.1	Accuracy	50
7.1.2	Beyond-Accuracy	50
7.1.3	Runtime	53
7.1.4	Considering Frequency and Recency	54
7.1.5	Offline Findings	56
7.2	Online Evaluation	56
7.2.1	Infobox Tests	57
7.2.2	Home Tests	63
7.2.3	Online Findings	66
7.3	Comparison and Summarization	66
<b>8</b>	<b>Conclusion</b>	<b>69</b>
8.1	Reflections	70
8.2	Future Work	71
	<b>Bibliography</b>	<b>75</b>

# List of Figures

3.1	PV-DBOW Model . . . . .	16
4.1	Studo App . . . . .	22
4.2	Job Ad Design . . . . .	23
4.3	Recommendation Scenarios . . . . .	25
5.1	A/B Test Workflow . . . . .	37
5.2	Parameter Tuning . . . . .	39
5.3	t-SNE Visualization . . . . .	40
5.4	UMAP Visualization . . . . .	40
7.1	Accuracy Results . . . . .	51
7.2	Beyond-Accuracy Results . . . . .	52
7.3	Runtime Results . . . . .	53
7.4	Impact of embeddings Results . . . . .	58
7.5	Influence of frequency and recency Results . . . . .	59
7.6	Merit of recency Results . . . . .	60
7.7	Considering context Results . . . . .	61
7.8	Combining embeddings Results . . . . .	62
7.9	Combining frequency and recency Results . . . . .	62
7.10	Influence of frequency and recency Results . . . . .	64
7.11	Combining frequency and recency Results . . . . .	65



# List of Tables

4.1	Dataset Statistics	26
7.1	Offline Evaluation Results	50
7.2	WSDM Results	55





# 1 Introduction

The job market is a very competitive field for both companies and job seekers alike. On the one hand, companies are struggling with filling their job vacancies. On the contrary, getting a desirable job is not an easy task to accomplish. Even a university degree is no longer a guarantee to find a good job (Jones, Schmitt, et al., 2014). In Austria, the demographic change makes things even more difficult as the people born in the low birth rate years are about to enter the job market <sup>1</sup>. Furthermore, the job market is also subject to other trends that emerge. Böhm (2013) shows that there is a shift in the job market towards the mobile phone sector.

Similarly, students are increasingly managing their studies on their phones. One example of this is the Austrian mobile application for students called Studo<sup>2</sup>. It contains various features related to student's needs, such as student mails and calendar. One feature is the Studo Jobs platform that tackles the problem of providing students with suitable jobs. This is a difficult problem to solve, since their needs differ from the normal workforce. To begin with, students are more likely to work in part-time jobs to cope with the financial pressure while studying (Robotham, 2012). Paid bachelor and master thesis are other very interesting options for them when nearing graduation, while entry-level jobs become relevant after they graduate. This limitation to entry-level jobs is due to the fact that students often do not have any substantial work experience when entering the job market (R. Liu, Rong, Ouyang, et al., 2017). One way to acquire the experience is via internships. A student job platform needs to account for these factors.

To leverage the problem of providing relevant jobs to students, a recommender system can be incorporated into the platform. Starting with the Netflix prize in 2007 (Bennett, Lanning, et al., 2007), recommender systems have become state-of-the-art solutions for various problems. Recommender systems have arrived at the job domain with business-oriented social networks like LinkedIn

---

<sup>1</sup>Derived from the Austrian birth rate data of Our World in Data (Max Roser and Ortiz-Ospina, 2019); Original data from Mitchell (1998)

<sup>2</sup><https://www.studo.com>

## 1 Introduction

and Xing pushing this trend. As such, job recommendations were the topic of the RecSys challenge in the years 2016 (Abel, Benczúr, Kohlsdorf, et al., 2016), as well as 2017 (Abel, Deldjoo, Elahi, et al., 2017), which were organized by Xing.

A job recommender is a multi-stakeholder system (Zheng, 2017). This means it has to balance the needs of two parties, the job seekers and the recruiter as outlined by Abel (2015). Thus, there must be mutual interest between both sides for a successful application. Even then, there is no guarantee of success. Typically, employers only hire one candidate per position, while job seekers only seek one job. This means that both choose their best available option, respectively. Kenthapadi, B. Le, and Venkataraman (2017) even linked the competitiveness of popular job offers with a decrease in user satisfaction.

For student job recommendation the lack of data, like work experience, makes it even harder to generate meaningful personalized recommendations (R. Liu, Rong, Ouyang, et al., 2017). Generating recommendations for a job platform is not a trivial task on its own. The dataset for job recommendation tends to be very sparse when compared to other domains like movie recommenders (Mishra and Reddy, 2016; Lee, Hong, Kim, et al., 2015). This can be attributed to the fact that users only need one successful recommendation for a job vacancy. After users find a matching job and get hired, they can leave the system altogether.

Job recommender systems are often seen as a matchmaking problem, where a CV is directly matched with a job (C. Zhu, H. Zhu, Xiong, et al., 2018; Qin, H. Zhu, T. Xu, et al., 2018). First, this assumes that job seekers provide high-quality CVs that accurately state the skills they possess. Another problem lies with the job postings themselves, which are unstructured in nature. One way to approach this problem is to consider the fact that different words can have a similar semantic meaning. Utilizing this knowledge, concepts describing the textual content can be extracted from the job description and used instead of the full-text. This is especially true for jobs in the technology domain. There the job descriptions often contain the technologies required for a given vacancy. For instance, the job title "Java developer" implies that it is a programming job. A job recommender system needs to overcome this problem in order to provide effective job recommendations based on the job description.

As such, this master thesis tackles the problem of providing students with fitting jobs for their needs by utilizing a deep learning recommender system. In particular, this thesis deals with the data sparsity and cold-start problem.

## 1.1 Research Questions

The master thesis tackles the problem of providing effective job recommendations for students with a focus on exploiting latent features on a real platform. We answer two research questions in particular. Research question 1 deals with the integration of latent features into the recommendations as detailed in Subsection [RQ1](#). Research question 2 explores the tuning of recommendations for students in Subsection [RQ2](#).

### **RQ1: How can latent features be used to improve job recommendations?**

The content of job postings is a valuable source to define whether a job is fitting for an applicant. However, job postings are usually represented as unstructured text documents. While the extraction of latent features of such documents has been thoroughly explored in the past, the actual use should be examined in greater detail. Hence, we need to extract suitable features from the textual content of the job posting.

Therefore, this thesis examines different approaches for using the hidden features that lie within the job description for job recommendations. Thus, first creating a suitable representation of those features and then using different strategies for retrieving relevant documents for the user.

To answer the research question, we extract the latent features and represent them as embeddings with the Doc2Vec algorithm of Q. Le and Mikolov (2014). For the retrieval, the content data is combined with the interaction data and used in conjunction with certain enhancements like the BLL equation from cognitive sciences (Anderson, Bothell, Byrne, et al., 2004). The results are evaluated on simulated real-world data in an offline evaluation.

### **RQ2: How to configure and combine recommendation algorithms for the best result in student job recommendations?**

Here, we research on the specific algorithm configurations and their application within a hybrid recommender system. The goal is to maximise the effectiveness of the recommendations in one particular setting.

## 1 Introduction

Therefore, the thesis explores how to create the best combination and configuration for the algorithms in the student job domain. This is carried out on the Studo Jobs platform. The results are evaluated on real users in an online study.

### 1.2 Structure

The master thesis is structured as follows. Chapter 1 describes the problem statement and outlines the contribution to the field of recommender systems. Chapter 2 surveys other works regarding recommender systems. This is followed by Chapter 3 which explains all the job recommendations algorithms relevant for the master thesis. This includes well-established methods such as collaborative filtering as well as novel approaches based on word embeddings. In Chapter 4 the focus is on introducing the mobile application called Studo as well as the job platform called Studo Jobs, which is incorporated in the mobile application. Moreover, it also describes the dataset used for generating the recommendations. The dataset was enriched with labels in a previous work (Reiter-Haas, Slawicek, and Lacic, 2017). How exactly the recommender system is structured and how it is incorporated into the platform is detailed in Chapter 5. The evaluation process is explained in Chapter 6, which consists of an offline as well as an online evaluation. The outcomes of the experiments are then discussed in Chapter 7. The results are built upon two previous publications (Lacic, Kowald, Reiter-Haas, et al., 2018; Reiter-Haas, Lacic, Duricic, et al., 2019). The master thesis is then concluded in Chapter 8, which also reflects on it critically and provides an outline for future research.

## 2 Related Work

The related work done in the field of recommender systems and deep learning is surveyed and is split into four parts. First, the different recommendation approaches are outlined in Section 2.1. Section 2.2 concentrates on recommendation systems for the job domain. Section 2.3 surveys different deep learning approaches with the focus on natural language processing. Section 2.4 focuses on the related work with deep learning recommender systems. Finally, Section 2.5 concludes the related work with a summary.

### 2.1 Recommendation Approaches

The research on recommender systems is a multi-disciplinary field mainly related to machine learning, data mining, information retrieval and human-computer interaction (Ricci, Rokach, and Shapira, 2011). Additionally, the field of recommender systems can be divided into several categories depending on the techniques used. The types described in the Recommender Systems Handbook are the following:

1. Content-based filtering (CBF)
2. Collaborative filtering (CF)
3. Demographic
4. Knowledge-based
5. Community-based
6. Hybrid recommender systems

The focus of this thesis lies in the machine learning and information retrieval field. Machine learning is used for the extraction of latent features, while information retrieval is relevant for generating and evaluating the recommendations. Of the different techniques mentioned, this thesis uses four of those types with the focus being on content-based filtering. Content-based filtering (CBF) uses content features in order to provide recommendations. The works of Pazzani and Billsus (2007) is used as a baseline approach and is being compared against. It uses the term frequency-inverse document frequency (TF-IDF) model

## 2 Related Work

for retrieval of relevant documents. The incorporation of latent features is also conducted on the content, which emphasizes the focus on content-based filtering.

Collaborative filtering (CF) is a popular technique for recommendation systems. This thesis uses the technique from Aggarwal (2016b) as the baseline. It assumes that users with a similar history have a similar taste. The personally unexplored items of similar users are likely of mutual interest. Thus, the algorithm generates recommendations by searching for similar users (i.e., candidate users) first. This is achieved by considering the interactions users have performed on the same items. Next, the algorithm considers all the items of the candidate users as candidate items. Finally, the most relevant items out of the candidate items are recommended. Relevant items are typically items that many candidate users have in common that the target user (i.e., the user who receives the recommendation) has not seen before. The content data of users can also be used to generate recommendations. The content of the user data typically consists of demographic data. Thus, this way of generating recommendations is considered a demographic recommender system. This thesis conducts one small experiment with a demographic recommender.

A combination of different approaches is also possible. These hybrid recommender systems are often utilized to improve the performance of the recommendations by exploiting multiple different aspects that might be of importance. There are multiple ways different recommendations can be combined together (Burke, 2002). For instance, collaborative filtering is often used in a hybrid setting. Combining CF with CBF eases many problems like the cold-start problem, where a user does not have any interaction. This hybrid combination uses the advantages of both. CF poses a strong baseline technique, while CBF does not require user interactions to work. This thesis uses mixed and weighted hybrids for combining recommendations. This completes the list of types of recommender systems used in this thesis. The exact details of all used algorithms are explained in Chapter 3. The two remaining types (i.e., community-based and knowledge-based) were not applied to the problem at hand.

Community-based abuse the structural information, which is available from social relations. The assumption is that the taste of individuals is similar to their community. For instance, people tend to have similar tastes to their friends and family. This type of recommender system does not apply to this thesis since the dataset lacks social data.

Knowledge-based recommendations use explicit knowledge models for the calculation. For instance, a manually crafted ontology that encodes the domain

knowledge. This type can be further divided into case-based and constraint-based recommender systems, respectively. While knowledge-based recommendations might be possible for the given scenario, the extensive modelling process required to create the knowledge base is out of scope for this thesis.

In Kowald, Pujari, and Lex (2017) the authors use the BLL equation from the cognitive sciences for recommendations. This equation models the memory retention of the human brain. The idea is to balance the importance of frequency and recency with a time decay parameter. In this thesis, we utilize the same approach to create embeddings of the user history.

Typically, the problem definition for recommendations is a user-item matrix filled with explicit ratings. The goals are predicting the missing ratings, which could be done via matrix decomposition. A popular method for this is singular value decomposition (SVD) as detailed in Koren and Bell (2015). Thus, the predictions were computed offline and then on just shown to the user on demand. However, it is hard to do this in a rapidly changing environment, since each change in the system needs a recalculation to be accurate. This is especially hard to achieve when the set of items changes frequently and many new users enter the system and leave shortly after. This scenario is especially true for job recommendations, which makes this precomputation of results infeasible. For this reason, many streaming recommendations have emerged (Yanxiang Huang, Cui, W. Zhang, et al., 2015; Chandramouli, Levandoski, Eldawy, et al., 2011; C. Chen, Yin, J. Yao, et al., 2013). In order to handle the fast-changing environment of the job domain, this thesis also provides real-time recommendations. Unlike, the other works the focus here is to provide recommendations under real-time constraints and immediately update them after interactions happen accordingly.

The performance of recommendations can be measured via various metrics. The paper of Parra and Sahebi (2013) provides an overview of such metrics for recommender systems. This thesis utilizes  $nDCG$  as the main accuracy metric and all the mentioned beyond-accuracy metrics (i.e., novelty, diversity, and coverage). The importance of the beyond-accuracy metrics is outlined in Ge, Delgado-Battenfeld, and Jannach (2010). Additionally, this thesis also incorporates a runtime analysis into the results.

### 2.2 Job Recommender Systems

Recommender systems have already been applied to the job domain and the work in this field is a hot research topic. Job recommenders are reciprocal recommenders (Pizzato, Rej, Chung, et al., 2010) since both parties (i.e., job seekers and companies) need to be satisfied to establish a successful recommendation. Recently, two RecSys challenges tackled the problem in the job domain. The RecSys 2016 Challenge<sup>1</sup> was about predicting jobs for users on Xing (Abel, Benczúr, Kohlsdorf, et al., 2016). Conversely, the RecSys Challenge 2017<sup>2</sup> reversed the problem to predicting users for new jobs, again held by Xing (Abel, Deldjoo, Elahi, et al., 2017).

In the RecSys Challenge 2016, a content-based recommendation was the baseline method<sup>3</sup>. Various methods had significant improvements over this method with gradient boosting techniques (as proposed by J. H. Friedman (2002)) being one of the ways to deal with the problem. This rather simple approach also achieved good results with Mishra and Reddy (2016) using a bottom-up approach for this technique. The winning paper models the temporal activity (Xiao, X. Xu, Liang, et al., 2016). Hence, the 2017 baseline<sup>4</sup> was changed to be based on this method and utilizing XGBoost (based on the paper by T. Chen and Guestrin (2016)) instead. The objective of the challenge also changed to candidate recommendations in a cold-start scenario. The winners of this challenge focus heavily on feature engineering (Volkovs, G. W. Yu, and Poutanen, 2017). The best performing approaches of both challenges could conduct an online study for practical evaluation. However, this option is not available to the general public.

This thesis addresses a similar problem but on the Studo Jobs platform instead. It also evaluates the result in an online study. Furthermore, the research focuses on job recommendations for students. This further complicates the user cold-start problem, since students often lack previous work experience in the first place. Another difference is that the recommendations are calculated in real-time on retrieval and thus are constrained by the algorithm runtime.

Another way to tackle job recommender systems is to model it as a matchmaking problem (C. Zhu, H. Zhu, Xiong, et al., 2018; Qin, H. Zhu, T. Xu, et al., 2018). Thus, given a description of a user and a description of a job find the best matches between those two sets. This is, however, not applicable in this case,

---

<sup>1</sup><http://2016.recsyschallenge.com/>

<sup>2</sup><http://2017.recsyschallenge.com/>

<sup>3</sup><https://github.com/recsyschallenge/2016/blob/master/Baselines.md>

<sup>4</sup><https://github.com/recsyschallenge/2017/blob/master/baseline/README.md>



since it requires carefully constructed CVs, which are not available in this form. Additionally, such an approach ignores the information gained via the various interactions that are performed on the platform. Furthermore, recommenders in the job domain can be applied in both directions. First, recommend jobs to users and second recommend users to jobs. If both are applied the result is a bilateral recommender system as proposed by Malinowski, Keim, Wendt, et al. (2006). Again, this not applicable in the setting since the platform does not allow the suggestions of CVs for jobs.

## 2.3 Deep Learning Approaches for NLP

Neural networks are a type of machine learning method, as described in Bishop (2006). They consist of a network of neurons, which resemble the nodes, and connections between the neurons. The neurons have an activation function, which calculates their output connections depending on their input connections and a singular value called the bias. Each connection also has a weight with which is multiplied with the actual value. Typically, the neurons are structured into layers. The input layer process the input data. The network then consists of one or more hidden layers. The result of the network is accessible by the output layer. The model is then learned with the backpropagation algorithm. The goal of the learning phase is to find the right weights for the connections. Given a two-layer neural network (i.e., one hidden and one output layer) and enough neurons, this network structure is able to approximate any function arbitrarily close. A simple type of neural network is the feedforward neural network, also known as a multi-layer perceptron.

Machine learning approaches can be divided into supervised and unsupervised learning. Supervised learning needs labelled data to train on. To validate the performance of the model supervised approaches typically withhold some of the data from the training dataset. The model is then used upon the withheld data (i.e., test dataset) to estimate the predictive power of the model. In unsupervised learning, the whole dataset can be used to fit the model. The model automatically finds hidden structures of the data, which is used as the output.

Deep learning is a sub-topic of machine learning with deep instead of wide neural networks. Although it only emerged recently, there are already many different techniques to apply deep learning. An overview of various deep learning approaches is seen in Deng, D. Yu, et al. (2014). As noted, for natural

## 2 Related Work

language (NLP) processing so-called “embeddings” can be created to decrease the amount of feature engineering required.

Recently, a big area of deep learning focuses on this automatic generation of low dimensional vectors to describe the latent features of items. A very successful paper was published by Mikolov, K. Chen, Corrado, et al. (2013) which is often used as a basis for more complex methods. This approach, often called Word2Vec, is an unsupervised learning method that finds the regularities of words appearing next to each other. Q. Le and Mikolov (2014) expanded the idea of Word2Vec to generate embeddings for text sequences, hereafter referred to as Doc2Vec. For this, the authors created two models the DBOW and PV-DM model. First evaluations showed that PV-DM is the dominant model.

Lau and Baldwin (2016) evaluated Doc2Vec and found out that DBOW performs better than PV-DM. This insight is contrary to the original author’s conclusion. The text embeddings for this thesis are created using the DBOW algorithm of Doc2Vec. If Doc2Vec is mentioned thereafter without specifying the type, it refers to the DBOW implementation by default.

### 2.4 Deep Learning Recommender Systems

In recent years many deep learning recommender systems have been proposed. The emergence of this area can be seen from the 1<sup>st</sup> Workshop on Deep Learning for Recommender Systems (Karatzoglou, Hidasi, Tikk, et al., 2016). The methods for the recommendation can vary widely. A survey of deep learning recommender systems is provided by S. Zhang, L. Yao, and Sun (2017). Recently, many advancements have been made to use word embeddings for recommendations (Covington, Adams, and Sargin, 2016; Barkan and Koenigstein, 2016; Greenstein-Messica, Rokach, and M. Friedman, 2017). Embedding approaches have also been used to generate recommendations in an offline setting (Yanbo Huang, 2016). This thesis expands on this idea and evaluates the generated embeddings in an online setting as well. A similar approach was already done for Xing in Yanbo Huang (2016), which also uses job document embeddings for the content-based recommendation. However, the author neither evaluates it in an online scenario nor evaluates it on beyond-accuracy metrics.

Another application for deep-learning recommender systems is session-based recommender systems, as proposed by Hidasi, Karatzoglou, Baltrunas, et al. (2015). The problem formulation switches the user with a session. Session-based recommendations suffer even more from data sparsity since sessions tend to

have fewer interactions compared to user-based recommendations. Often these recommender systems are modelled with recurrent neural networks (Chatzis, Christodoulou, and Andreou, 2017; Hidasi, Karatzoglou, Baltrunas, et al., 2015; Smirnova and Vasile, 2017). This thesis focuses on modelling the user to the content via textual embeddings. Although anonymous sessions are present in the dataset, the majority of the data is about logged in users. Also, the usage of the deep learning method is not about a direct prediction, but for the conversion of the content of the jobs. Thus, RNNs do not apply in this setting.

## 2.5 Summary

This thesis focuses on content-based filtering with text embeddings in the job domain. For this, it uses the DBOW Implementation of Doc2Vec and combines it with the BLL equation. Thus, the main line of related research is deep learning content-based recommenders for the job domain. This thesis is very similar to previous work done on Xing (Yanbo Huang, 2016). However, the evaluation focuses more on the practical setting by also conducting an online evaluation. Furthermore, this thesis uses the dataset from the Studo Jobs platform.

Parts of this thesis have already been published in Reiter-Haas, Slawicek, and Lacic (2017), Lacic, Kowald, Reiter-Haas, et al. (2018), and Reiter-Haas, Lacic, Duricic, et al. (2019). Reiter-Haas, Slawicek, and Lacic (2017) focus on processing the data, while Lacic, Kowald, Reiter-Haas, et al. (2018) and Reiter-Haas, Lacic, Duricic, et al. (2019) tackle the offline and online evaluation, respectively. This thesis further expands upon this previous research.



## 3 Job Recommendation Algorithms

This chapter describes the different recommender algorithms used for the experiments of this thesis. It first starts with two well-known algorithms, content-based filtering in Section 3.1 and collaborative filtering in Section 3.2. Later it introduces new deep learning algorithms in Section 3.3. Finally, it specifies hybrid recommenders in Section 3.4.

Regardless of the algorithm, the system only recommends public items (i.e., job ads) and excludes the current item from the recommendations. This prevents the algorithm from recommending the current item again, which would otherwise be a likely scenario. The algorithms work with implicit feedback, which provides a natural choice for a dataset with a browsing history (Koren and Bell, 2015). The type of interaction used for the feedback can differ depending on the system. Unless stated otherwise, the algorithms are configured to use all available interactions and treat them equally.

As a fallback mechanism when no recommendations can be generated (i.e., user cold-start), the most popular (MP) approach is used. This approach just recommends items with the highest numbers of interactions. Thus, providing a non-personalized way for recommendations.

### 3.1 Content Based Filtering

Content-based filtering (CBF) uses the content features to generate recommendations as described in Pazzani and Billsus (2007). For job recommendation, the content of job ads is used to generate recommendations for users with a similar profile as shown by Aggarwal (2016a). For this approach, the content needs to be preprocessed. One typical method from information retrieval is to use the term frequency - inverse document frequency (TF-IDF) as described in Rajaraman and Ullman (2011). For recommender systems, this can be modelled by Equation 3.1 (Lops, De Gemmis, and Semeraro, 2011). In this equation,  $t_k$  specifies the  $k$  term and  $d_j$  specifies the  $j$  document. The frequency of the term  $k$

### 3 Job Recommendation Algorithms

in document  $j$  is denoted by  $f_{k,j}$ . The total documents are denoted by  $N$ , while  $n_k$  denotes the number of documents the term  $t_k$  appears in.

$$TF-IDF(tk, dj) = \underbrace{\frac{f_{k,j}}{\max_z f_{z,j}}}_{TF} \cdot \log \underbrace{\frac{N}{n_k}}_{IDF} \quad (3.1)$$

Given this equation, the closeness between two documents can be calculated by a similarity function, like the cosine similarity. To generate recommendations with this equation, one of the documents set to the user profile. Then documents that are similar to the user are considered relevant. For this thesis, the current item the user views is used as his profile. Thus, the user gets a list of recommended items that are similar to the current one.

The default configuration for this algorithm considers the top 25 TF-IDF terms for the recommendation. It uses the job description, title, and teaser from the dataset described in Chapter 4. However, not all content is of equal value. When calculating the overall recommendation score the title gets boosted by a factor of 1.5 and the teaser by a factor of 2. The implementation of the retrieval is further described in Chapter 5.

## 3.2 Collaborative Filtering

Collaborative filtering (CF) takes the history of interactions and finds users that have a similar interaction history (Aggarwal, 2016b). It then recommends the items, which other users have consumed and not by the target user itself. In this thesis, only the job ad views are used for the interaction history.

The algorithm can also be enhanced to take the current item more heavily into account when generating the recommendations. A simple method to make the recommendation more context-aware is to just consider users that also interacted with a target item. This context-aware approach is hereafter referenced as context collaborative filtering (CF<sub>cont.</sub>).

The default configuration for this algorithm considers the top 40 most similar users as candidates and does not consider the context for generating recommendations.

### 3.3 Doc2Vec

Research question 1 is about exploiting the latent feature of the job postings. This thesis explores word embeddings for this cause. The idea of word embeddings is to construct low dimensional vectors consisting of real numbers that capture the regularities of words appearing within texts.

A very influential approach dubbed as Word2Vec was published by Mikolov, Sutskever, K. Chen, et al. (2013). The authors proposed two new methods in particular to generate those vectors. The first is called continuous bag of words (CBOW), which uses the context of a word (a set of nearby words where one word is left out) as input and tries to predict the missing word. The second method, called skip-gram, reverses the approach. Thus, given a word as input its content is predicted. Regardless of the method, the resulting word embeddings model words that have similar concepts close to each other.

This approach was then extended by Q. Le and Mikolov (2014) to come up with vectors for longer text segments (e.g., paragraphs or whole documents) commonly known as Doc2Vec. There are again two methods to generate these embeddings. The first method, distributed memory of paragraph vectors (PV-DM), is similar to CBOW. Additionally, it uses an identifier for the paragraph as input and predicts a single word as output. On the other hand, distributed bag of words of paragraph vectors (PV-DBOW), just takes the paragraph id as input and predicts a set of words as output. Figure 3.1 shows how the job identifier can predict several words with the PV-DBOW model.

Doc2Vec has already been applied to content-based job recommender systems. Yanbo Huang (2016) successfully applied this algorithm to career social networking site Xing<sup>1</sup> in an offline setting. Thus, an offline evaluation was done as described in 6.1. This thesis goes further and explores the performance of this approach in 6.2 and applies it to the more specialized field of student recommendations. For this thesis, the PV-DBOW method is used and the vectors are trained on the job ad description.

The default configuration for this algorithm uses the cosine similarity, while the embeddings are trained on a single epoch a learning rate of 0.025. The window size, which configures the sequence length for the input, is set to 20. The training uses a regularization technique by a negative sampling of 10 items. The resulting vectors have a dimensionality of 100. The training of the vectors considers only the job description.

---

<sup>1</sup><https://www.xing.com/>

### 3 Job Recommendation Algorithms

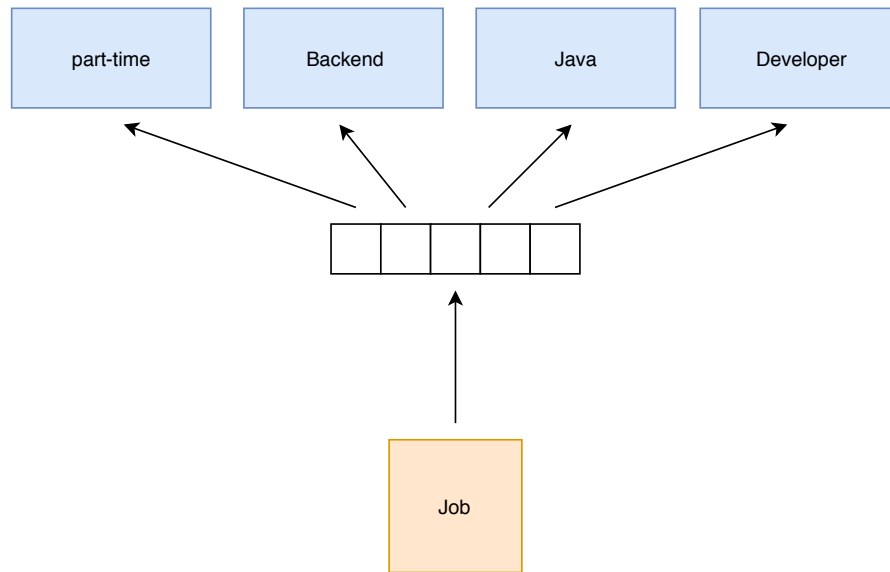


Figure 3.1: Representation of the PV-DBOW model. Given a job, the context of the job is predicted. In this case, the predicted context is part-time, Backend, Java, and Developer.

These vectors build one aspect of the deep learning approach. The other aspect to consider to retrieve relevant documents for a user. This thesis uses three ways to achieve this. Subsections 3.3.1 explains the most simple retrieval method. Subsection 3.3.2 describes an intuitive way to model the user history for more advanced retrieval. Finally, Subsection 3.3.3 focuses on a more sophisticated method that considers the interaction times as well. For the similarity function used in all approaches, the cosine similarity is used.

#### 3.3.1 Last Vector

In its most simple form only the current job ad, which is the last interacted one, is considered and thus the vector of the job ad is taken as it is. By utilizing a similarity function related documents can be retrieved. This approach is denoted as LAST strategy. However, this method provides a non-personalized way to retrieve documents. Given a last interacted job, every user gets the same recommendations. Thus, it might be useful to consider the user history as well for generating the reference vector, which the other two retrieval method cover.



### 3.3.2 Average Vectors

An intuitive way to construct a reference vector, that considers the history of job interactions, is to just merge the vectors by averaging them (denoted as AVG strategy). This is similar to how word vectors can be averaged to form sentences as described by Q. Le and Mikolov (2014). This forms a new target vector to work with. The recommendations can be generated by simply using a similarity function on this vector. For such a model, the dimensions that are dominant in a lot of the interactions stay dominant. Contrary, if the interactions depict contradicting behaviour the dimensions cancel each other out.

### 3.3.3 Base-Level Learning

The availability of specific items in human memory is sensitive to the frequency and recency of its use (Petrov, 2006). This means that frequent or recent experiences are easier to recall. The base-level learning (BLL) equation from ACT-R architecture (Anderson, Bothell, Byrne, et al., 2004) reflects this tendency towards frequency and recency. The equation is part of the declarative memory module that promotes long-term retrieval. The module provides a model from cognitive sciences, which captures both aspects of the before mentioned methods (the recency of LAST, as well as the frequency of AVG). The construction for the BLL for a given item  $j$  is described by Equation 3.2.  $TS_{ref}$  refers to the timestamp when the recommendation is requested.  $TS_{j,i}$  specifies the timestamp for the  $i^{\text{th}}$  interaction (of  $n$  total interactions) on the item  $j$ . The parameter  $d$  denotes the magnitude of time decay.

$$BLL_j = \ln\left(\sum_{i=1}^n (TS_{ref} - TS_{j,i})^{-d}\right) \quad (3.2)$$

Thus, it calculates the time difference for each interaction. The time decay is then applied by the parameter  $d$ . Since  $d$  is in the exponent the decay happens much more rapidly in the beginning. This gives more weight to recent interactions. Whereas, for old interactions, the difference is of little significance. Finally, the terms are summed up and the value is regularized by the log function. This prevents the dominance of frequent interactions on the same item. Given the result for each item, the reference embedding is then calculated by weighting the individual embeddings  $vec_j$  as shown in Equation 3.3.

### 3 Job Recommendation Algorithms

$$vec = \sum_{j=1}^m vec_j * BLL_j \quad (3.3)$$

The approach is used in a similar way as described in Kowald, Pujari, and Lex (2017). Given a user's interaction history, the BLL equation can use the item embeddings and come up with a vector that describes the user preferences. This preference vector can then be used to find items that are similar to those preferences. This approach effectively combines the interactions of the user with the content of the job ads. Per default, the time parameter  $d$  is set to 0.5.

#### 3.3.4 User2Vec

This thesis does not explore a baseline approach for demographic recommendations but applies it in a similar fashion as the content-based recommender with deep learning. The approach is denoted as User2Vec. This approach is not explored in greater detail since it is only used for a side experiment.

Similar to documents, user data can also be converted to embeddings. While one approach might be to convert the user content with the Doc2Vec algorithm (Section 3.3). It might pose two possible problems. First, a user might not have any content at all, which is the case for anonymous users in the system. Thus no content can be used directly for this case. Second, if the content does not form a meaningful sequence of words. This can be non-textual content, as well as just unordered lists of keywords. In this case, other methods like autoencoders might be more suitable.

Regardless of content, the users always have sequences of items they interacted with. These sequences can then be used with the Doc2Vec approach. A downside of this approach is that user interactions are very volatile and thus it is hard to come up with a stable model.

## 3.4 Hybrid Recommendations

Each presented algorithm has advantages as well as disadvantages. For instance, collaborative filtering is known to provide good results, but cannot be applied for new users without interaction data. Thus, a combination of multiple algorithms might lead to a better overall result. There are several ways at how to produce hybrid recommendations as detailed in Burke (2007). This thesis

utilizes two of those approaches. Firstly, mixed hybrid recommendations are described in Subsection 3.4.1. Secondly, weighted hybrids are described in Subsection 3.4.2.

### 3.4.1 Mixed Hybrids

The mixed hybrid uses a round-robin system of the algorithms. Thus, it calculates the recommendations for each algorithm and then takes the first recommended item of the first algorithm, the first of the second algorithm and so on. When an item is already present in the resulting recommendations, it is not inserted again. Instead, either the item or the algorithm can be skipped. When skipping the item the next item of the algorithm result is taken instead. Conversely, when the algorithm is skipped it just proceeds with the next algorithm instead.

This process is continued until the specified amount of recommendations is generated. If an algorithm does not have any recommendations left in its results, it is skipped entirely.

### 3.4.2 Weighted Hybrids

The weighted hybrid uses a score for each result of the individual algorithms. A simple way to come up with such a score is to apply the inverse rank as a score. Thus for results of length  $n$ , the first item of the results gets  $n$  as a score, the second  $n - 1$  and so on. The score of each algorithm is then weighted by some predefined weights. The weights are simply multiplied by the score as seen by Equation 3.4, where  $A$  is the set of all the individual algorithms. The algorithm then chooses the items with the highest total score.

$$s_i = \sum_{a \in A} w_a * s_a \quad (3.4)$$



## 4 Studo Jobs Platform

This chapter describes the Studo Jobs platform and the dataset used for the experiments. First, it provides a detailed overview of the Studo mobile application (Section 4.1), where the job platform is integrated into. Next, the job platform itself is described in Section 4.2, where the evaluations are carried out. The job platform consists of two scenarios for showing recommendations, which are detailed in 4.3. Finally, the dataset is explained in Section 4.4. It consists of user, job and interaction data.

### 4.1 Studo App

The Studo mobile application was created early in 2016 and has the main functionality to provide easier access for university relevant services<sup>1</sup>. As of December 2019 the app serves over 100,000 students of 42 universities in Austria and is the biggest app of its kind in this country. The user base of the app is very active as there are 5 million applications starts per month, with many students using the app multiple times per day.

The app consists of several easily accessible tabs as can be seen in Figure 4.1. The app has a news feed that provides students with information relevant to their studies. Figure 4.1a shows a course overview and management system that is directly integrated into the app. A calendar automatically shows them events based on the courses they have enrolled in. The Studo Chat allows students of similar courses or studies to ask questions and exchange information with each other. The mail client receives mails on the go and makes it easy to send mails as well. The app also shows the grades of past courses and allows access to the curriculum of the students. Additionally, it provides students with information on menus for nearby lunch options. External services are also included in the app. The cooperation with Iamstudent<sup>2</sup> allows access to a wide variety of vouchers eligible for students. Finally, a job platform can be accessed from

---

<sup>1</sup><https://studo.com/>

<sup>2</sup><https://www.iamstudent.at/>

## 4 Studo Jobs Platform

within the app (Figure 4.1b). While the job platform can also be accessed from the browser, the app is the main way students interact with it.

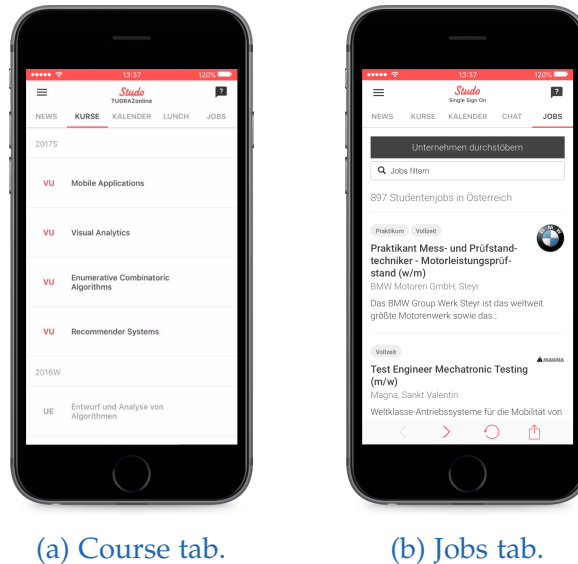


Figure 4.1: Screenshots of the course (a) and jobs tab (b) in the Studo App. The displayed tabs depend on the university of the student.

## 4.2 Studo Jobs

The Studo Jobs platform<sup>3</sup> was launched in Summer 2016 in the Austrian federal state Styria first. It provides companies with a way to advertise their job vacancies via job postings (also referred to as job ads) to students. A job ad contains the textual description of the job with some structure. Figure 4.2 shows an example of a job ad on the platform. One year later in the Summer of 2017, it was rolled out for the whole of Austria (11<sup>th</sup> August 2017).

On the 18<sup>th</sup> of September 2017 the first recommender, which was a content-based filtering algorithm, went live as can be seen in Figure 4.3a. This was followed by the first deep learning approach by the end of 2017. In the second quarter of 2018, the recommendations were also used for the job overview page (i.e., Home page) of the job platform with a collaborative filtering approach, thus expanding its use case. As of February 2019, all job lists on the platform are either generated by the recommendation engine or is a result of a search query.

<sup>3</sup>previously found on <https://studo.co/jobs/>

The Studo Jobs platform can be accessed via mobile app and via the web page, most users access it via the app (around 70% as of January 2019). Due to the limited screen space of mobile phones, it is not possible to show the recommended jobs in a sidebar as is the case on the web page. Thus, the recommendations are shown after the job ad. This also limits the number of job ads that can be shown simultaneously. Due to these limitations, only three job ads are being recommended per page.

A UI A/B test was conducted to determine whether showing the recommendations directly after the teaser would lead to higher usage of the recommendations, as well as raise the awareness that they even exist. This could also lead to more active users (i.e., interacting with more job ads). However, after some negative feedback from some customers, which did not like that users were tempted to view the next job ad before they finished reading theirs, the test was prematurely cancelled. This shows some of the limitations of conducting the experiments on a real platform. Furthermore, it shows that optimizing the platform for the customers is more than just pushing up the numbers.

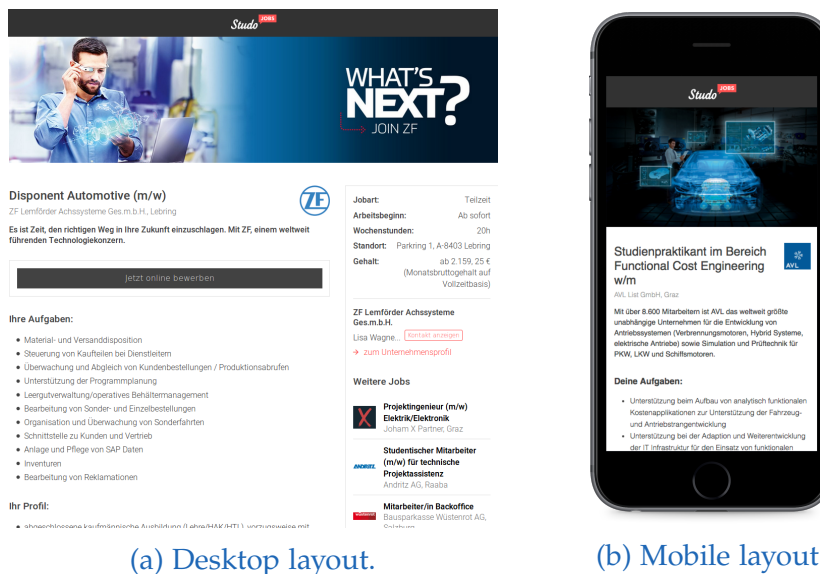


Figure 4.2: An example job ad, which consists of a textual description. (a) shows the desktop-optimized layout, while (b) shows the app-optimized layout.

Many job platforms provide the opportunity to buy premium job ads, that gain more visibility. The Studo Jobs platform has decided against such a practice since it is designed as a data-driven platform. Thus, the students should always get the best suiting job ads for them. Furthermore, the winners of the RecSys 2017 challenge also opted for a simpler model, where the premium job ads and

## 4 Studo Jobs Platform

users were not distinguished from the rest (Volkovs, G. W. Yu, and Poutanen, 2017). It seems that the added complexity works against the little benefits such a business strategy would incur.

A user poll of Studo users with 934 participants, showed that most users (98.4%) are willing to search for a job on a mobile phone (Horvath, 2018). Moreover, almost half would also apply per mobile phone (44.9%). This trend was already predicted by Böhm (2013). Furthermore, the study suggests that mobile users expect to be able to apply via mobile phone.

Since November 2019, the job platform is superseded by the Talto career platform<sup>4</sup>. This platform changed the focus from jobs to career-related content in general. Thus, it puts more emphasis on the companies and also provides articles. Nevertheless, this thesis focuses on its predecessor and therefore on job recommendation for students.

### 4.3 Recommendation Scenarios

The platform currently has two main recommendation scenarios. Both scenarios are shown on different parts on the platform. Furthermore, the context in which they are shown is different. One serves as an overview of available job ads, while the other serves navigational purposes. Thus, it makes sense to evaluate and tune the scenarios individually. The Infobox scenario is described in Subsection 4.3.1, while the Home scenario is described in Subsection 4.3.2.

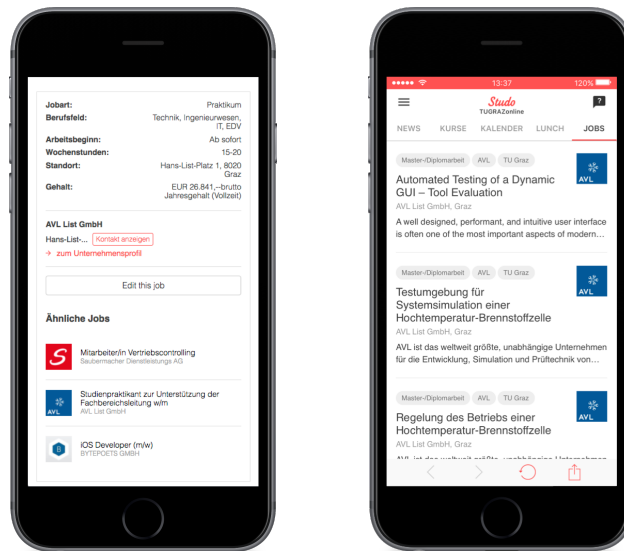
#### 4.3.1 Infobox Scenario

The first recommendation scenario is shown when viewing a job ad. The user gets additional jobs recommended, which are shown in the job Infobox. Depending on the screen size the recommendations get displayed differently. On mobile devices, the recommendations are shown after the job ad (as seen in Figure 4.3a). When viewing on the desktop, the job ads are shown on the sidebar. These recommendations improve the flow on the platform since it is easier to get to another job ad if one does not quite fit. Due to space limitations, the number of recommended job ads is quite low (i.e., 3 job ads). This scenario uses CBF as baselines as described in Chapter 3 Section 3.1.

---

<sup>4</sup><https://talto.com/>





(a) Infobox scenario.

(b) Home scenario.

Figure 4.3: The two recommendation scenarios on Studo Jobs. The Infobox recommendations (a) are shown when viewing a job ad. It contains a list of only 3 job ads. The Home recommendation scenario (b) is shown when viewing the main hub of the job portal. It contains 5 recommendations of the 25 displayed items.

### 4.3.2 Home Scenario

The second recommendation scenario was added later to the platform. When the user is on the Home view, a list of job ads gets recommended. In this scenario, a longer list can be recommended (i.e., 25) to the user. Of those, the first 5 are recommended (Figure 4.3b). This scenario also receives a lot more interactions when compared to the Infobox scenario. This scenario uses CF as baselines as described in Chapter 3 Section 3.2.

## 4.4 Dataset

The data that is used for generating the recommendations consists of three data structures. First, there are the job ads, which are used for content-based and vector-based algorithms (explained in Subsection 4.4.1). Second, there is the user-specific content data. This data is referred to as CVs and is further described in Subsection 4.4.2. Note that there are anonymous users that do not have this explicit content. The final data structure, the interaction, links

## 4 Studo Jobs Platform

the other two together. This datatype is thoroughly described in Subsection 4.4.3. Finally, Subsection 4.4.4 explains the preprocessing and enrichment of job ads.

A snapshot of the complete platform stats is shown in Table 4.1. As can be seen, the data is quite sparse. Moreover, the number of views is dominant over the other interaction types.

Type	Amount
Job Ads	5835
Users	87,906
CVs	18,589
Views	401,267
Applications	8,529
Contact Details	17,415
Shares	2,138
Feedbacks	507,397
Recommendations used	6371
Sparsity	98.95%

Table 4.1: Overall dataset statistics (as of 19<sup>th</sup> of May 2018). Views are the dominant interaction type. The ratio of interactions to possible user and job ad combination is very sparse.

As a remark, the outlined structures of the dataset are just how the information is imported into the recommender system. However, the internal representations in the database differ from those structures. For instance, each generated interaction also adapts the corresponding user profile and job ads. Thus, each user profile contains all personally consumed job identifiers. Conversely, each job ad contains the identifiers of all users who interacted with it. Furthermore, the job ads have counters for each interaction type. This is done to speed up the retrieval process of the system.

### 4.4.1 Job Ads

All job ads follow the same structure and no unstructured or semi-structured content is permitted for the platform (e.g., PDFs). Although the job ads themselves are structured, the content itself is text-based and thus unstructured.

There are 3 textual content fields for the job, namely title, teaser, and description. The description provides the most details of those three. While the format is unstructured, the general information within is typically very similar, due

to the requirements the document has to fulfill. Normally, it contains the responsibilities and requirements for the job ad. Often, it also contains job benefits. Another particularity is that negated sentences are used very sparingly. This phenomenon is likely a result of overqualified employees (i.e., having too many skills) being less of an issue than employees lacking the required skills to perform the job in the first place.

A job ad contains several fields that are detailed below. It contains several textual fields (i.e., *Title*, *Teaser*, *Content*, *ContentHtml*) that describe the job ad. It also contains many fields for the metadata about the job ad (i.e., *Categories*, *Tags*, *Language*, *BeginNow*, *Effort*, *Salary*, *CompanyName*). There are fields for the job location with varying granularity (i.e., *LocationCity*, *LocationFederalState*, *LocationCountry*) and fields for the different contact information (i.e., *ApplicationPhone*, *ApplicationOnlineUrl*, *ApplicationMail*). Finally, it also contains internal information for the system (i.e., *ID*, *UrlPath*, *Status*, *DateCreated*, *DateModified*). An example job ad is shown by Listing 4.1.

**ID** specifies a unique identifier for the job ad.

**UrlPath** specifies the public URL to access the job ad.

**Status** can be "PRIVATE", "PUBLIC", "EXPIRED" and specifies the visibility of the job ad and whether the job has been public at some point. This is relevant since expired job ads are valid data sources, while private jobs might not be filled out completely.

**DateCreated** is the ISO date representation as string datatype when the job ad has been created.

**DateModified** specifies the last modification time, again as ISO representation.

**Title** specifies the title of the job ad.

**Teaser** of the job ad. A short text that shall encourage users to read the full job ad. Often contains information about the company.

**ContentHtml** specifies the description of the job ad as HTML representation.

**Content** specifies the description of the job ad as text representation.

**Categories** specify a list of employment types that suit the job ad.

**Tags** specify a list of tags that are related to the job ad. The tags are either referencing a job discipline or a job industry.

**Language** of the job ad. On the platform, jobs can either be in German, which is the default option, or in English. The language is automatically detected from the text.

**BeginNow** specifies whether the starting time for the position is now.

**Effort** specifies the hours per week for the job. Can be "MINI" (0-10h), "HALF" (10-35h), "FULL" (> 35h) or free text.

**Salary** as free text, which allows any format.

**ApplicationPhone** specifies the application's contact phone number.

**ApplicationOnlineUrl** specifies the URL for an online application.

**ApplicationMail** specifies the application contact email address.

**CompanyName** specifies the name of the company.

**LocationCity** specifies the city of the job vacancy.

**LocationFederalState** specifies the federal state of the job vacancy.

**LocationCountry** specifies the country of the job vacancy. The jobs are typically in Austria, but some are from neighbouring countries like Germany or Italy.

Listing 4.1: An example of an job ad as JSON.

```
{
  "id": "EJSZ4...",
  "urlPath": "2017/06/verkaufsmitarbeiterin/",
  "status": "PUBLIC",
  "dateCreated": "2017-06-21T11:57:01.000+02",
  "dateModified": "2017-06-21T11:57:01.000+02",
  "content": "Als VerkaufsmitarbeiterIn ...",
  "contentHtml": "<h3>Als VerkaufsmitarbeiterIn ...",
  "title": "VerkaufsmitarbeiterIn Markthalle",
  "categories": ["Teilzeit"],
  "tags": ["Uni Graz", "TU Graz"],
  "language": "de",
  "teaser": "IKEA verkauft alles fuers Zuhause...",
  "beginNow": true,
  "effort": 0,
  "salary": "\euro 1.546 / Monat (Basis Vollzeit) ",
  "applicationPhone": "",
  "applicationMail": "",
  "applicationOnlineUrl": "https://ww8.ikea.com/...",
  "companyName": "IKEA Moebelhaus OHG",
  "locationCity": "Graz",
  "locationFederalState": "Styria",
  "locationCountry": "Austria",
  "viewCounter": 227,
  "applicationButtonClicks": 0
}
```

### 4.4.2 CVs

In the system, users can state explicit information about themselves for their user profiles. The user profiles are also called CVs (although in a very simplistic form). These are used for the CV recommendations. Furthermore, the platform uses the data to pre-configure the filtering options on the platform. The user profiles are generated in the Studo app only. Users have to opt-in to allow this data to be saved.

A CV contains several fields that are detailed below. Most of those fields are filled out by the user (i.e., *Universities*, *Studies*, *ProfessionalField*, *ExpectedFinalDegreeYear*, *LocationFlexibility*, *Effort*). Two fields track the engagement of the users with their CVs (i.e., *UpdateCount*, *ViewCount*). Finally, the CV is linked to the user with a unique identifier (i.e., *UserID*). An example CV is shown in Listing 4.2.

**UserID** specifies a unique identifier of the user who filled out the CV. Thus, each user can have at most one CV.

**Universities** specify the readable identifiers of the universities of the user (e.g., "grazTU" or "grazUniKF").

**Studies** specify the names of the university studies a user is signed up for.

**ProfessionalField** specifies the selected professional fields. The values are chosen from a predefined list of fields.

**ExpectedFinalDegreeYear** specifies the expected final year, which a user has specified. The format described as regex is `'>?[0-9]{4}'` (e.g., "2019" or "> 2018").

**LocationFlexibility** can be empty, "CITY", "STATE" or "GLOBAL". The empty string means the value has not been set by the user. This specifies the range the user is willing to travel for a job.

**Effort** specifies the work effort in hours per week a user wants to work. Can be "MINI" (0-10h), "HALF" (10-35h), "FULL" (> 35h).

**UpdateCount** specifies the number of times the user updated the CV.

**ViewCount** specifies the number of times the user opened the CV. It is always bigger or equal to the number of the *UpdateCount* field since users have to view their CVs to update them.

## 4 Studo Jobs Platform

Listing 4.2: An example of an CV as JSON

```
{
  "userId": "SQH29...",
  "universities": ["grazTU", "grazUniKF"],
  "studies": [],
  "professionalField": ["ASSISTANCE", "FINANCE", ...],
  "expectedFinalDegreeYear": "2018",
  "locationFlexibility": "GLOBAL",
  "effort": ["MINI", "HALF", "FULL"],
  "updateCount": 5,
  "viewCount": 16
}
```

### 4.4.3 Interactions

All interactions have the same structure. All the interaction data is collected implicitly. Thus, the user cannot provide explicit feedback about the recommendation yet. However, the meaning and value of each of those interactions differ. The interaction data is used for collaborative filtering. It is also implicitly used for the most popular algorithm, but the data is tracked indirectly on the job ad. The interaction data consists of different types of interactions (e.g., views, applications, and opening contact details).

Interactions in the system can be modelled as a quadruple of user, item, timestamp and interaction type. As such, the interaction contains two relational fields (i.e., *UserId*, *JobAdId*), as well as the type and the time as other fields. For data processing purposes a unique identifier is assigned to each interaction. Furthermore, two other fields provide additional information on the interaction (i.e., *Authenticated*, *Platform*). Finally, interactions that happen as a result of a recommendation can be linked to the recommendation as well by the *RecommenderId*. Each field is further detailed below. An example of an interaction is shown in Listing 4.3.

**ID** specifies a unique identifier for the interaction. It can be created at random.

**UserId** specifies the identifier of the user, who initiated the interaction.

**Authenticated** specifies whether it was an authenticated user or an anonymous user. App users are typically authenticated, while web users tend to access the platform anonymously.

**Time** specifies the time and date as ISO representation when the interaction occurred.

**Platform** specifies the platform where the interaction happened. It can be "STUDO\_APP" or "WEB". Note that WEB does not distinguish between mobile and desktop.

**RecommenderId** specifies the ID of the recommendation that leads to the interaction, which is generated by the recommender system. It is left empty if the interaction happened without the guidance of the recommender.

**JobAdId** specifies the identifier of the job ad on which the interaction was performed.

**Type** specifies the type of interaction. Can be "AD\_VIEW" (when clicking the job ad), "APPLICATION" (when clicking the apply button while on job ad, does not guarantee that the user does send the application for real), "SHOW\_CONTACT\_DETAILS" (when a user expands the contact details while on job ad, this is separate from the application button), "COMPANY\_VIEW" (when a user opens the company information site from the job ad).

In addition to data for generating the recommendations, the system also needs to collect data for the online evaluation. Thus, the system also saves all the parameters for each generated recommendation. Note that in the interaction data there is the field called *RecommenderId*. This field is set when the interaction was generated by a previous recommendation. For job views, it means that the user clicked on a recommender job ad. For all the other interactions it means that they were performed on the viewed recommended job ad. The system then tracks these interactions as feedback for the recommender. This feedback is for the online evaluation in Chapter 6).

Listing 4.3: An example of an Interaction as JSON

```
{
  "id" : "MP9Z4..." ,
  "userId" : "SQH29..." ,
  "authenticated" : false ,
  "time" : "2017-07-12T12:55:34.165+0000" ,
  "platform" : "WEB" ,
  "recommenderId" : "WDF4F..." ,
  "jobAdId" : "EJSZ4..." ,
  "type": "AD_VIEW"
}
```

### 4.4.4 Data Processing and Enrichment

In preparation for the master thesis, a related experiment has been conducted in order to automatically enrich the job ads. The goal of this experiment is to predict labels based on job descriptions. In particular, we are interested in predicting professional fields. For this purpose, multiple classifiers were trained and then evaluated on Accuracy, F1-Measure, and AUC. The results showed that support vector machines can predict the labels very well and performing even better when optimized with stochastic gradient descent. It evaluates the algorithms on four labels, namely:

- Business
- Technology
- Catering
- Software

The evaluation compares the results of 10 different classifier models. Three of these classifiers use decision trees (i.e., Random Forest, CART, and AdaBoost). Three models use neural networks (i.e., MLP, CNN, and M-CNN). Two methods are based on probabilities (i.e., Naive Bayes, and Logistic Regression). And finally, two support vector machines are used (i.e., Linear SVM, and SVM-SGD). These extracted professional fields are shown as metadata on the job ads and are represented by the field *tags*. Additionally, they are used for a qualitative analysis of the embedding space.

The support vector machine with stochastic gradient descent is the best performing classifier in terms of Accuracy and F1-measure. This classifier has been incorporated into the platform as a service to automatically infer the professional fields from the job ad. The findings of this experiment were presented at the RS-BDA workshop on the I-KNOW conference 2017 (Reiter-Haas, Slawicek, and Lacic, 2017).

For the tagging experiment, a different dataset was used in order to train the model. This is because at the time of the experiment not enough data was provided by the Studo Jobs platform. Thus, for the dataset other job platforms were crawled. The crawled dataset consists of four fields. The ID uniquely identifies the job ad. From the content of the job ad, only the text was parsed out of the HTML. The job title was also crawled but not directly used in the experiment. Finally, each job ad consists of a set of ground truth labels for the professional fields. Out of these ground truth labels, only four were used. The reason for this is that at the time of the experiment, only those four labels were present at the Studo Jobs platform.



## 5 Implementation Details

Part of the thesis is the implementation of the algorithms in a practical setting for the evaluation. Especially how the recommender system was set up on the Studo Jobs platform for the online evaluation. The chapter consists of three parts. Section 5.1 gives an overview of the recommender architecture in general. Section 5.2 details the integration of the recommender into the Studo Jobs platform. Finally, Section 5.3 explains the training of embeddings in more detail.

### 5.1 Architecture

The recommender architecture of the Studo Jobs platform uses the ScaR (Scalable Recommendation-as-a-service) framework<sup>1</sup>. The framework is described in detail by Lacic, Traub, Kowald, et al. (2015). It makes use of the microservice architecture as proposed by Lewis and Fowler (2014). In this architecture, the program is split up into individually working components. Since all the components function independently from each other, the whole system also functions independently. Nevertheless, several components need to be configured to work with the job platform. The system is remotely connected to the job platform and communicates via HTTP requests.

At the basis of the ScaR framework is a data management service that provides an interface to the database. Out of the box, it supports Apache Solr<sup>2</sup> as a data source. Solr is built atop of the Apache Lucene search engine. Lucene supports various functions for the retrieval of documents. For instance, the scoring function can be adapted by specifying boosting weights. This functionality is used for content-based filtering to specify the importance of various textual fields for the recommendation. The engine service builds the core of the framework. This service is responsible for generating recommendations. The exact algorithms and parameters for the generation can be configured via

---

<sup>1</sup><http://scar.know-center.tugraz.at/>

<sup>2</sup><https://lucene.apache.org/solr/>

## 5 Implementation Details

a repository service. Custom-tailored services build upon the basis. A data importer handles the data schema and is the pivotal point for the integration detailed in Section 5.2. The recommendation provider processes the recommendations requests and responses. Finally, there is also a service to conduct online and offline evaluations.

The ScaR framework also contains a deep learning service. This service first trains on the items in the database. It then uses the computed model to create the embeddings for existing as well as new items in the system.

The microservices are deployed via Docker<sup>3</sup> containers on a Docker stack, which are configured via a Docker compose file. The servers are set up as Docker swarm where the Docker services are running on. The recommender system runs on the production and a staging environment. The staging server serves for testing and developing purposes. It allows testing the functionality of updates of the recommender itself, but it also provides recommendations for local development of the job platform. This is needed since running the recommender system in a local environment is unfeasible due to the high amount of resources required.

### 5.2 Integration

For the full integration of the recommender system and running A/B test in production, three steps need to be taken. First, the data needs to be fed into the recommender system, which is described in Subsection 5.2.1. Given this first step, the system can already generate recommendations but not evaluate them. The next step is creating a feedback loop, as detailed in Subsection 5.2.2. This feedback loop enables the evaluation of the recommendation performance but does not allow conducting A/B tests. Finally, for an A/B test, the user base needs to be split accordingly, which is described in Subsection 5.2.3. Additionally, the recommender needs to provide responses in real-time for a smooth user experience, which is explained in Subsection 5.2.4.

#### 5.2.1 Data Ingestion

For the integration, the Studo Jobs platform has to import its data into the recommender system. In principle, this can easily be achieved via the data

---

<sup>3</sup><https://www.docker.com/>

importer. Regardless, the synchronization of the two systems is very important but difficult to implement correctly. Unsynchronized interaction data adversely affects the recommendation quality since the latest information cannot be used for the generation. The synchronization is even more important for the state of job ads. If a job ad expires, this update needs to be sent immediately. Otherwise, the recommender continues to provide recommendations for a job, which is no longer available to the user. This leads to unwanted "404 - not found" pages.

While important, the synchronization of the job platform with the recommender system cannot be ensured at all times. But given enough time after the last update, the whole system should become consistent, which is called eventual consistency. This is a typical trade-off for distributed systems (Vogels, 2009). Putting it differently, this workflow sacrifices consistency, at least to some extent, for availability and partition-tolerance. It is stated by the CAP theorem, that only two of those three properties can be achieved in a distributed system (Gilbert and Lynch, 2002). Furthermore, the synchronization of a distributed needs to deal with network issues like timeouts. Thus, it is important to consider whether the data import request delivers a valid response or retry otherwise.

### 5.2.2 Feedback Loop

For the evaluation of recommendations in production, a feedback loop has been established. In ScaR, each recommendation request is logged with a unique identifier. This identifier gets returned alongside the generated recommendations. There is one adaption needed to close the loop. When delivering recommendations for the user, this identifier is built into the user events. Thus, each event that happens as a result of a recommendation links to its recommendation identifier. Furthermore, all subsequent events on the same item get assigned the same recommendation identifier.

A recommendation request requires several parameters. It consists at least of a *RecommenderProfile* and a *UserID*. An *ItemId* is only required for some profiles. The amount of recommendations is specified with *MaxResults*. All parameters are detailed below.

**UserID** specifies a unique identifier of a user.

**ItemID** specifies a unique identifier of an item.

**MaxResults** specifies the maximum number of recommendations that shall be generated.

**RecommenderProfile** specifies the filters the recommendation by certain criteria (e.g., companies, job cities).

## 5 Implementation Details

The response of the recommender system is simple. Of course, the most important content is the *Recommendations* of jobs. But, it also contains a *RecommID* for the feedback loop and some *DebugInformation* for practical purposes. The details of the complete response are described below.

**RecommID** represents a unique identifier of the produced recommendations.

This information is required to build the feedback loop.

**Recommendations** represents a list of IDs of recommended items. The items are ordered by relevance.

**DebugInformation** contains the HTTP status code, Error ID, and a custom message.

The recommender system stores the information for both the request and the response for later evaluation. Additionally, it stores internal information like the composition of hybrid recommendations. To complete the feedback loop, the interactions, which are sent to the recommender systems also include the *RecommID*. This information is linked to the previously stored recommendation information. Thus, the available information provides the whole picture of a recommendation and the resulting interactions.

### 5.2.3 A/B Test Workflow

To conduct A/B tests for online evaluations, the workflow has been extended. Figure 5.1 details the typical workflow for an A/B test on a web page. The same workflow is used for A/B tests on recommendations. When a recommendation is requested, the system first checks whether the user has already been assigned an A/B test variant. If it has a variant, it retrieves its settings and creates the recommendation accordingly. If no variant has been assigned the system determines into which variant the user shall be put in to achieve the desired split (typically equal splits). Then, the recommendation is created accordingly and the chosen variant is stored. Typically, the variant just determines the recommender algorithm with a particular configuration (i.e., recommendation profile) for generating the recommendations. This process ensures that each user is assigned to exactly one group for the complete duration of an A/B test. When no A/B test is running the system just requests a specific recommendation profile.

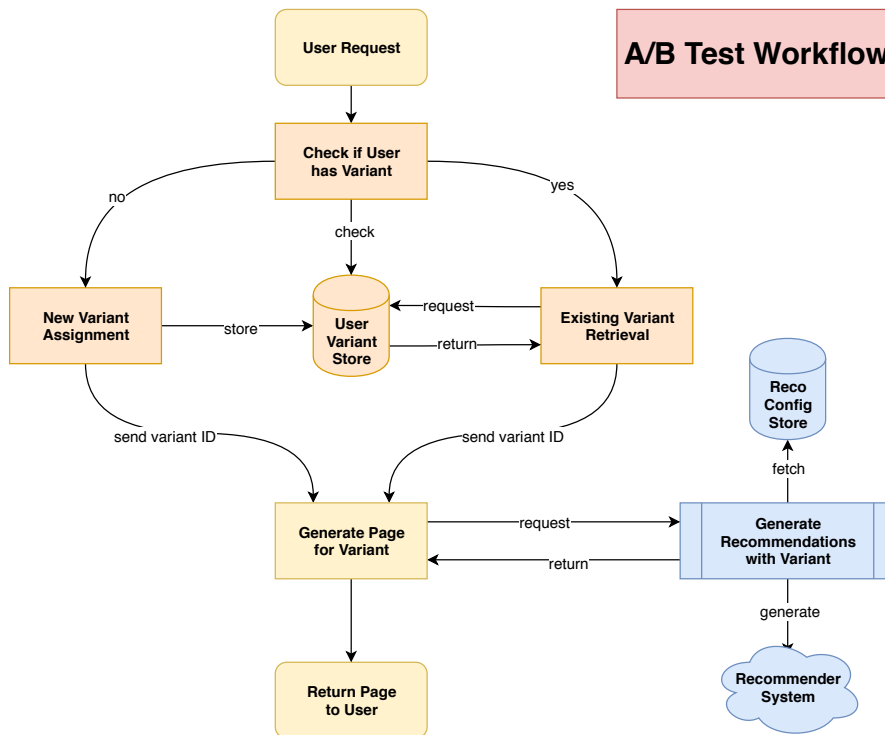


Figure 5.1: Diagram describing the assignment workflow for A/B tests. When requesting a web page with a running A/B test, this workflow determines the variant the user receives. This process ensures that for the whole duration of a test, each user always gets the same variant. Yellow indicates general web page request handling. Orange handles the A/B variant assignment. Blue is recommender specific.

### 5.2.4 Realtime Recommendations

In order to ensure the responsiveness of the platform, the recommendations have to adhere to realtime constraints. Ideally, the recommendation response time should be below 100 milliseconds for good user experience (Eksombatchai, Jindal, J. Z. Liu, et al., 2018). For this reason, the recommender system makes use of the inverted index provided by Lucene. The recommender implementation is configured to store all the information in a specific way to allow quick retrieval. For instance, interactions store the user identifier in the items and vice versa. This allows the retrieval of users by only considering the item data. The retrieval of the embeddings is more complex. It makes use of a custom build Solr plugin using the Lucene payload feature (the same way as in Reiter-Haas, Lacic, Duricic, et al. (2019)). Payloads store arrays containing arbitrary information and associate tokens for retrieval. In particular, the tokens for the embeddings store the position of the vector representation. Storing positional

## 5 Implementation Details

information allows for fast similarity calculations. While various similarity functions are supported, this thesis uses specifically the cosine similarity for retrieval.

### 5.3 Embedding Training

This section details the information for the training of the embeddings. The embeddings are trained using the Doc2Vec algorithm. In particular, the DBOW model is used. The Doc2Vec algorithm is written in DeepLearning4j<sup>4</sup>, which has the algorithm already implemented and just needs to be configured. This section is further split into the tuning of the algorithm in Subsection 5.3.1 and the qualitative analysis of training results in Subsection 5.3.2.

#### 5.3.1 Parameter Tuning

For parameter tuning, we measure the job application rate for different settings. The tuning of parameters considers window size (i.e., 5, 10, 15, and 20), negative samples (i.e., 0, 5, 10, and 15), epochs (i.e., 1, 2, 3, 5, and 5), and dimensions (i.e., 100, 150, 200, 250, and 300). The results of this tuning are visualized in Figure 5.2.

No clear winning strategy could be derived since the algorithm provides good results regardless of parameter choice. We use a window size of 20, which is the largest amount used in tuning. Thus, the algorithm considers long textual sequences as input. The learning rate is set to 0.025, which is a typical value for the algorithm. To avoid possible over-fitting, the algorithm is trained on 1 epoch only. This also speeds up the training time. To keep the model simple, the resulting embeddings are of size 100. However, we also conducted evaluations with embedding dimension sizes of 200 and 300. As regularization, the training uses 10 negative samples.

#### 5.3.2 Embedding Analysis

The embeddings are visualized in two-dimensional space using two techniques. First, an approach by Maaten and Hinton (2008) called t-distributed stochastic neighbor embedding (t-SNE), which preserve the distance from the higher

---

<sup>4</sup><https://deeplearning4j.org/>

## 5.3 Embedding Training

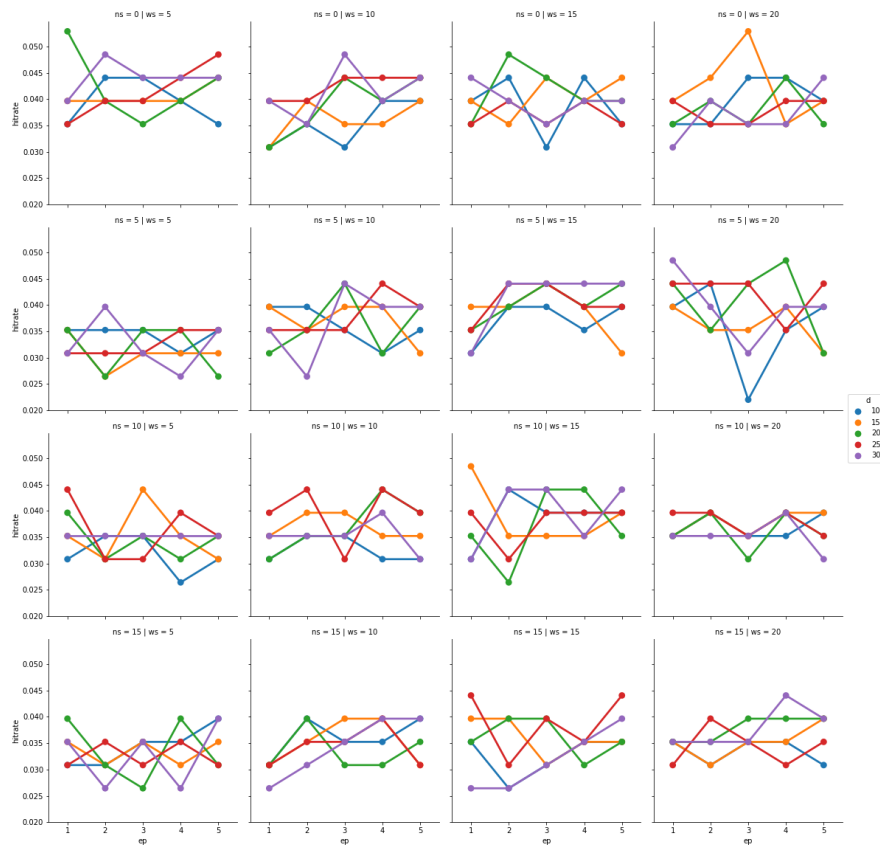


Figure 5.2: Results of parameter tuning. No correlation between the different tuning parameters can be observed.

dimensional space. Thus, points that were far apart are still far apart, while close points stay close. Figure 5.3 shows the trained embeddings, where the points are colored according to their tags. It was trained with a typical perplexity of 30, which specifies the number of nearest neighbours for the learning process. The other approach by McInnes, Healy, and Melville (2018) is called uniform manifold approximation and projection (UMAP). It exploits the topological structure and geometry within the data. The authors claim that it preserves more of the global structure compared to t-SNE. Figure 5.4 shows the results of the training, with 15 neighbours considered and the euclidean metric. The color again represents the tags.

With these techniques, a qualitative analysis of the embeddings can be performed. It is apparent from both visualizations that there is no global clustering pattern regarding tags, while several local clusters are found. The results of this analysis are in line with the expectations. When jobs are quite similar you

## 5 Implementation Details

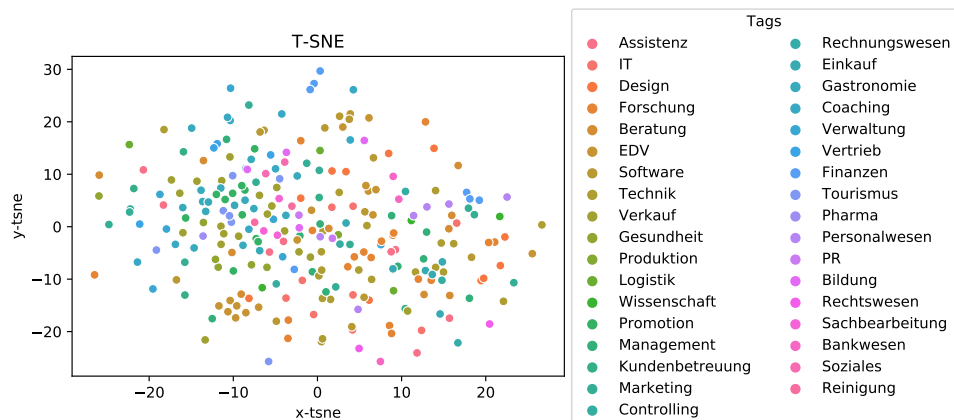


Figure 5.3: Visualization of embedding space using t-SNE. Colors indicate the tags. The jobs are spread out rather evenly across the two-dimensional space.

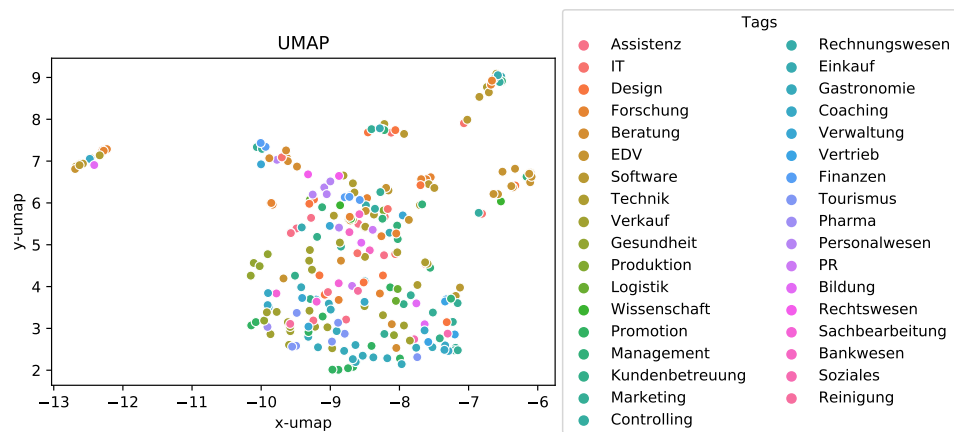


Figure 5.4: Visualization of embedding space using UMAP. Colors indicate the tags. There are several irregularities in the distribution of points. Local clusters tend to be dominated by a single tag.

expect them to also have the same tag. However, there are also similarities between jobs of different tags. When comparing the two visualizations, it is apparent that t-SNE is spread out rather evenly, while UMAP has large gaps between clusters. Interestingly, close clusters in UMAP tend to be dominated by mainly one tag. This indicates that similar jobs with the same tag are typically more similar than similar jobs of different tags.



## 6 Evaluation Protocol

This chapter explains the procedure for the evaluation of the recommender system. There are several ways to evaluate recommender systems, namely offline evaluations, online evaluations, and user studies (Shani and Gunawardana, 2011). In this thesis, the recommender system is evaluated using offline as well as online evaluations. The setup for the offline evaluation is described in Section 6.1, while Section 6.2 describes the procedure for the online evaluation.

The metrics to evaluate recommender systems can be split into accuracy based metrics and beyond-accuracy based metrics. Accuracy metrics measure how precise the recommendations are in terms of recommending suitable items. This is done by measuring whether users actually use the recommendation in a positive manner (e.g., click on the recommended item). Beyond-accuracy metrics measure other factors that are of interest, like how diverse the generated recommendations are.

The applied metrics differ depending on the type of evaluation performed. For the online evaluation, a feedback loop has been established and the metrics can directly be computed from information available. For offline evaluations, the simulation based on the user's interaction history has to be performed first.

Recommender systems are typically evaluated based on explicit feedback like reviews or ratings. Due to the given dataset only containing interaction data, the evaluation is only conducted on implicit feedback (as proposed for Netflix by Koren and Bell (2015)). This is true for both procedures, online and offline. The problem at hand is formulated as follows: given a target user, find the top- $k$  most relevant jobs from a set of available jobs.

### 6.1 Offline Protocol

An offline evaluation uses data from the past and then splits it into a training and a test set. The algorithms then simulate real interactions given the training set and measure the results in comparison to the test set. Since part of this

## 6 Evaluation Protocol

master thesis is already published in Lacic, Kowald, Reiter-Haas, et al. (2018), the evaluation protocol is similar to this publication. In addition to measuring the accuracy of the recommendations, we explore the impact on other measures as well.

For the evaluation, the dataset was generated by a method similar to Kowald, Lacic, and Trattner (2014). Only users with enough (i.e., a predefined variable) interactions are considered in the evaluation. Then the last  $k$  job postings of the user history are used for the test set, while the others are used for the training set. For the reported results in this thesis, only users with at least 4 interactions on different job postings are included. The last 3 of those job postings are added to the test set, while the remaining (i.e., at least 1) items are put in the training set. Thus, it simulates a previous state of the user's history. The recommender system then tries to predict the 3 items in the test set for each user. The recommendations for the evaluation are configured to return a list of 3 items as well, as is the case for the Infobox scenario described in Chapter 4.

In Lacic, Kowald, Reiter-Haas, et al. (2018) the recommender system's performance was predicted by several measures that are also used for this thesis. The  $DCG@k$  (Equation 6.1) is a measure of accuracy, which also takes the position of the recommended items into account. The  $rel_i$  signifies how relevant an item is at position  $i$ . Typically it is just the indicator function of whether the item belongs to the set of relevant items (i.e., 1 if it is relevant and 0 otherwise). The parameter  $k$  specifies how many items are being recommended at most. The metric is usually normalized by dividing with the  $iDCG@k$ , which is the highest possible (i.e., ideal) value the  $DCG@k$  can achieve. The resulting metric is called the  $nDCG@k$ . Since  $nDCG$  is an accuracy based measure, it tries to predict how accurate the recommendations were when comparing it to the test set.

$$DCG@k = \sum_{i=1}^k \frac{2^{rel_i} - 1}{\log_2(i + 1)} \quad (6.1)$$

$$nDCG@k = \frac{DCG@k}{iDCG@k} \quad (6.2)$$

Also, other accuracy based measures from information retrieval, like Accuracy and F1-measure, apply for the problem at hand. However, these measures do not take the position into account. Accuracy (Equation 6.3) just measures the percentage of correctly retrieved samples. In particular, it specifies a ratio of correct relevant (i.e.,  $tp$ ) and correct irrelevant (i.e.,  $tn$ ) documents, over the whole dataset ( $fp$  and  $fn$  denote incorrectly considered as relevant or

irrelevant, respectively). Typically, only a small fraction of items are considered relevant for a particular user, which leads to a skewed distribution in the dataset. Thus, one could achieve good results in terms of Accuracy by only predicting these towards the skewed side (i.e., predicting no relevancy). In the case of recommender systems, providing an empty recommendation list might achieve high results as most items are likely irrelevant. However, this is not the desired outcome. If one side is more important than the other, this can be achieved by utilizing either precision or recall. Precision (Equation 6.4) only considers relevant documents that have been predicted. Thus, it is used when you rather prefer a small set containing only relevant documents. Recall (Equation 6.5), on the other hand, considers all relevant documents. It is used when providing a full list of relevant documents is important. Since both cases can easily be optimized by either predicting only a few relevant (in case of precision) or all documents (in case of recall), these measures are usually not used without another measure. This is where F1-measure (Equation 6.6) comes into play. It is the harmonic mean of the two other measures. Thus, both metrics have to be reasonably high for good results. This means that the results have good precision without sacrificing the recall and vice versa.

$$accuracy = \frac{tp + tn}{tp + fp + tn + fn} \quad (6.3)$$

$$precision = \frac{tp}{tp + fp} \quad (6.4)$$

$$recall = \frac{tp}{tp + fn} \quad (6.5)$$

$$F_1 = \frac{2 * precision * recall}{precision + recall} \quad (6.6)$$

Additionally, two other measures are used for the offline evaluation. The first is the mean reciprocal rank (MRR). The second is the mean average precision (MAP). Both equations also consider the rank of the results and serve a similar purpose as nDCG. The equation for the MRR is given by Equation 6.7, and the equation for MAP is given by Equation 6.8.

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \quad (6.7)$$

## 6 Evaluation Protocol

$$MAP = \frac{\sum_{q=1}^Q AveP(q)}{Q} \quad (6.8)$$

Using accuracy based measures is only one way to evaluate a recommender system. However, there are also other important factors to consider. Novelty, as described in Zhou, Kuscsik, J.-G. Liu, et al. (2010), is a measure that considers whether the recommended items are novel, which means that they were not often recommended before. Thus, recommending popular items very often might lead to high accuracy, but very low Novelty. This is undesired as novel recommendations have been shown to build trust in the recommender system (Pu, L. Chen, and Hu, 2011). The Novelty@k, as given in Equation 6.9, computes the fraction of the popularity of the recommended item to the most popular item. This is done for all  $k$  items that are recommended. The log function is used for regularization, while the addition 1 avoids the computation of  $\log(0)$ .

Depending on the definition, the novelty is the inverse of the calculated sum, which is the definition used from now on. Thus, a list of only the most popular item (repeating within the recommendation list) would result in a novelty of 0. Recommending the most unpopular item(s) would yet result in the maximum novelty, which would be close to 1.

$$Novelty@k = 1 - \frac{1}{k} \sum_{i \in k} \frac{\log_2(pop_i + 1)}{\log_2(pop_{MAX} + 1)} \quad (6.9)$$

An important consideration when recommending a list of items at once is how similar those items are. Although recommending very similar items might result in high accuracy, it might lead to dissatisfaction of the users. Smyth and McClave (2001) propose a diversity metric for recommender systems and argue for the advantages of diverse recommendations in certain scenarios. Equation 6.10 defines Diversity@k, which uses a distance measure to compute how dissimilar the recommended items are to each other. This is achieved by applying the dissimilarity function  $d(i, j)$  on the content of the job postings.

$$Diversity@k = \frac{1}{k \cdot (k - 1)} \sum_{i \in R} \sum_{j \in u^k, j \neq i} d(i, j) \quad (6.10)$$

Serendipity measures how unusual or surprising recommendations are (Y. C. Zhang, Séaghdha, Quercia, et al., 2012). For this, it computes the similarity of a recommendation with the expected contents. This expectation is calculated

by the items the user has consumed in the past. If it is dissimilar then the serendipity is high. Very high serendipity might come at the cost of accuracy and thus would have a negative impact. However, a high serendipity recommender might recommend something the user likes, but did not explore in the past. Thus, it might steer the user into a new dimension and boost diversity. The equation for this measure can be seen in Equation 6.11.

$$\text{Serendipity}@k = \frac{1}{|R_k| * |H_s|} \sum_{i \in R_k} \sum_{j \in H_s} d(i, j) \quad (6.11)$$

Computing measures per recommendation does not lead to a global statement of the recommender system's performance. Hence, the individual results need to be aggregated for an overall estimation. Equation 6.12 averages the results for all user recommendations to calculate the global performance of the metric. It is applied to all previously mentioned metrics and is the value used to report the results.

$$\text{GLOBAL} = \frac{1}{|U|} \sum_{u \in U} \text{METRIC} \quad (6.12)$$

For the three previously mentioned beyond-accuracy metrics, no perfect values exist. Nevertheless, you can still evaluate and optimize towards certain target values. In Lacic, Kowald, Reiter-Haas, et al. (2018), this was used on the novelty measure. First, the novelty of consumed (i.e., applications) recommendations is calculated. This adapted novelty measure is then used as a target value for optimizing the recommendations. Thus, you want recommendations that are as close as possible to this target value, since it might boost the number of applications.

The results are also evaluated in terms of user coverage and recommendation runtime. User coverage specifies the percentage of users for which recommendations can be generated. Sometimes it is not possible to generate any recommendations for a user with a certain algorithm. This is typically the case for cold-start users (i.e., users without any interactions). Non-personalized approaches should achieve a value of 1 since they are not user-dependent. Note, that only users with a minimum amount of interactions are considered for the simulation. Thus, a user without this minimum amount of interaction is not considered and some algorithms might not work in that case.

The time it takes to generate a recommendation (i.e., runtime) is also interesting. This measure is especially important when generating recommendations in

## 6 Evaluation Protocol

real-time. It is measured in milliseconds and the runtime metric only considers the calculation time, but does not account for external factors like the internet speed of a user. Hence, these external factors might play a more important role in practice. Nonetheless, it gives the first estimation of the computational complexity of the algorithms.

As already indicated, an offline evaluation might not catch all the aspects of a real-world scenario. Thus, the findings have to be validated in an online system. The protocol for these evaluations is covered in Section 6.2.

### 6.2 Online Protocol

The online evaluation is done to validate the results of the offline evaluation. In the online evaluation, different configurations are being tested in a real-life scenario. Thus, this kind of evaluation captures all the aspects of the platform. This leaves little to no speculation on whether the results are reasonable (provided that there is a statistical significance). However, the results might be specific to the platform they are applied to.

The method chosen for the online evaluation is to conduct an A/B test. In an A/B test, the user base gets split up into two or more groups for the whole duration of the test. All the tests conducted for this master thesis were split in two groups where 50% of the user base belongs to each group. The splitting of users into groups is done in order to easily compare the given approaches with each other. The equal splits are done to increase the likelihood of statistical significance when a low amount of data is used. Since more available data for each approach increases the likelihood of statistical significance for the results.

To measure the performance of the algorithm, the click-through rate is being utilized (Equation 6.13). This metric measures the ratio between the impression against some kind of action that was performed. In particular, the action specifies whether any action happened on a recommended item. Typically, this means that at least one click was performed since this is the usual starting point for other interactions. Unfortunately, no impressions are included in the dataset. Thus, it is assumed that each generated recommendation is also shown to the user. This means that the CTR specifies the percentage of recommended items that lead to an interaction.

$$CTR = \frac{\#actions}{\#impressions} \quad (6.13)$$

For the calculation of CTR, we use the feedback loop established in Chapter 5. It ensures that actions, which resulted due to a recommendation, can be linked to the corresponding recommendation. Given this setup, the metric can easily be calculated by considering all the recommendations in question and their linked interactions.

Finally, it is important to measure how likely the hypothesis of the result is true or not. This is done by a hypothesis test. The utilized test statistic for this is the Person's chi-squared test statistic  $\chi^2$  (Pearson, 1900). If the results are not in the critical area of the chi-squared distribution, they can be said to be statistically significant. The typical p-value of 0.05 was chosen for this. In Equation 6.14,  $O_i$  specifies the observations, while  $E_i$  specifies the expectations, which is the mean for the null hypothesis.

$$\chi^2 = \sum_{i=1}^n \frac{(O_i - E_i)^2}{E_i} \quad (6.14)$$

Multiple A/B tests were conducted using the previously established method. Due to the chosen approach (i.e., two 50% splits), only one A/B test can be performed per recommendation scenario simultaneously. The duration for each A/B test was approximately between two weeks and one month each. The exact duration for each test is stated in Chapter 7.

For runtime, the student's t-test statistic is used to calculate the statistical significance (Student, 1908). In Equation 6.14,  $x$  specifies the samples,  $\mu_0$  specifies the mean,  $s$  specifies the standard deviation, and  $n$  specifies the number of samples.

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}} \quad (6.15)$$

In theory, an online evaluation should replicate the results of the offline evaluations. However, the results cannot be compared directly since they utilize different metrics. The comparison of CTR to accuracy based measures seems reasonable, but there is a factor that CTR cannot account for and that is the length of the session. Since the navigation through the job platform does not reach an endpoint, the algorithms can never achieve 100% CTR. This factor becomes smaller as the sessions get longer.





# 7 Results

This chapter provides the findings of the experiments. The evaluations are conducted with the setup described in Chapter 6. First, the offline evaluation results are detailed in Section 7.1. Then, in Section 7.2, the online evaluation results are being covered. Finally, in Section 7.3, the two types of evaluation are compared against each other, and the findings are summarized.

## 7.1 Offline Evaluation

The results of the offline evaluations consider three different kinds of measures on multiple algorithms. The first part of the offline evaluation compares four baseline algorithms with a vector-based approach. The baseline algorithms consist of Most Popular (MP), Content-based Filtering (CBF), Collaborative Filtering (CF), and Context CF ( $CF_{cont}$ ). The vector-based approach uses the LAST strategy for aggregation. Each kind of the three different measures is evaluated in their corresponding subsection. First, the accuracy based measures in Subsection 7.1.1. These measures are generally used to determine the expected quality of the recommendation. Second, the beyond-accuracy measures in Subsection 7.1.2. While the beyond-accuracy measures do not have a direct goal to achieve, they provide valuable insights into the underlying workings of the algorithms. Third, the Subsection 7.1.3 very shortly covers the computational performance of the algorithms. The evaluation is performed with the protocol defined in Section 6.1. The results of this part of the offline evaluation are displayed in Table 7.1 but are also visualized in their corresponding subsections.

Next, Subsection 7.1.4 considers the temporal effects of job interactions. Hence, it incorporates the aspects of frequency and recency into the evaluation. It summarizes the previous work of Lacic, Kowald, Reiter-Haas, et al. (2018), which has been published previously in the IFUP 2018: Workshop on multi-dimensional information fusion for user modeling and personalization. Finally, Subsection 7.1.5 summarizes the findings of the offline evaluation.

## 7 Results

Approach	CBF	CF	CF <sub>cont</sub>	MP	LAST
nDCG	0.0077	0.0284	0.0276	0.0168	0.0074
MRR	0.0049	0.0182	0.0178	0.0106	0.0047
MAP	0.0052	0.0191	0.0186	0.0107	0.0051
F1	0.0073	0.0261	0.0250	0.0158	0.0069
Precision	0.0074	0.0261	0.0250	0.0158	0.0069
Recall	0.0073	0.0261	0.0250	0.0158	0.0069
Diversity	0.4757	0.6568	0.6526	0.6197	0.5592
Novelty	0.5960	0.4067	0.4122	0.3283	0.5924
Serendipity	0.6379	0.6787	0.6774	0.6449	0.6793
User Coverage	0.9029	0.9972	0.9968	1.00000	0.9999
Runtime (ms)	45.1279	41.4760	53.4375	12.9440	50.5185

Table 7.1: Results of the offline evaluation of CBF, CF, CF<sub>cont</sub>, MP, and LAST. All results were evaluated when recommending 3 items. First are six accuracy metrics, followed by four beyond-accuracy, and finally runtime. All accuracy metrics report similar results across the five approaches.

### 7.1.1 Accuracy

Figure 7.1 visualizes various accuracy based measures (i.e., nDCG, MRR, MAP, F1, Precision, and Recall). As the results indicate, all approaches are stable across the different metrics (i.e., the order and magnitude between the approaches are relational to the metrics). Thus, we consider all metrics together instead of analysing each metric individually.

Both CF approaches perform best by far. Interestingly, including the context (i.e., CF<sub>cont</sub>) slightly worsen the performance. We suspect that this effect could be a result of the small dataset as the candidate filtering further narrows the small people of similar users. MP also performs well on accuracy. Hence, it already provides a good baseline in a cold-start scenario. We make use of this fact in the online evaluation. CBF struggles to generate accurate recommendations and performs the worst in this regard. The performance of LAST is similar to CBF.

### 7.1.2 Beyond-Accuracy

We employ four beyond-accuracy measures for the evaluation (i.e., Diversity, Novelty, Serendipity, and User Coverage). Unsurprisingly, Figure 7.2 shows that

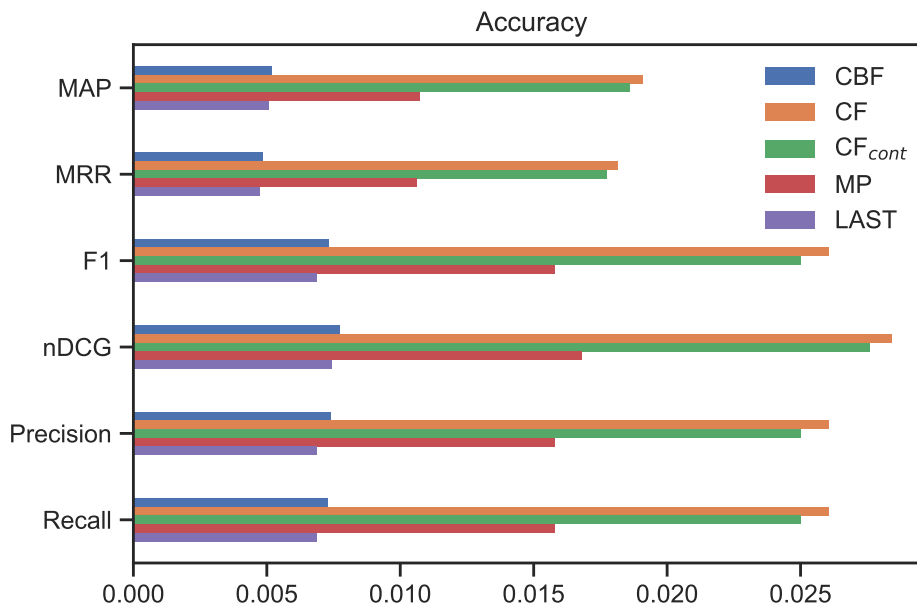


Figure 7.1: Results of the accuracy measures. CF performs best. While both content-based approaches perform significantly worse.

the approaches perform very differently on the various metrics as they measure very different aspects of the recommendation.

In terms of user coverage, all approaches but CBF achieve almost perfect results. CBF cannot provide recommendations for jobs that are dissimilar to all other jobs. Hence, users who interact with those jobs would not receive any recommendations. This effect vanishes when using LAST since the similarity function almost certainly finds something to recommend. Considering both CF approaches, they can almost cover all the users, which the offline evaluation considers, as well. However, this is only true for users, which have the minimum amount of interactions required to be included in the evaluation (i.e., at least 4). Finally, only MP is able to recommend to all users since the recommendations are the same for all of them (i.e., non-personalized).

Given this and the fact that it also provides reasonably high accuracy, we use MP for the setup of the online evaluation. There, we use it as a fallback in the case that not enough recommendations can be generated. This ensures that a fixed-size list of jobs is always generated for the job platform.

Both MP and CF lead to highly diverse results, while CBF performs poorly in this regard. This effect is expected since content-based recommendations

## 7 Results

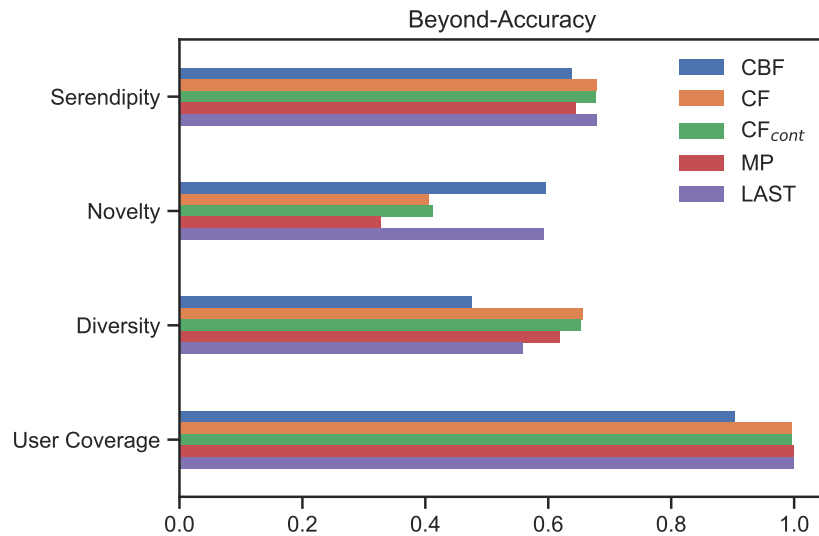


Figure 7.2: Result of the beyond-accuracy measures. The biggest difference can be observed in novelty. MP performs worst, while both content-based approaches have a very high novelty.

consider similar items for recommendation and thus the recommendations are also similar to each other. The LAST approach performs a bit better in this regard since it performs the operations not on the content itself, but the embeddings (i.e., low-dimensional representations of the content) instead. Since CF is oblivious about the content for the generation of recommendations, a high diversity is expected. This is true for both CF algorithms. Since  $CF_{cont}$  is more restrictive on the jobs considered for the recommendation, the results are likely a little less diverse. The diversity of MP depends heavily on the dataset and is rather high in this case.

The novelty measure differs greatly between the approaches. Both, CBF and LAST, have a very high novelty since they do not consider interaction data for generating recommendations. Thus, new items, which are novel per definition, can be recommended immediately. Unsurprisingly, MP has the lowest novelty since it recommends the same items to all users. Both CF approaches lie somewhere in the middle. Interestingly, the more restrictive  $CF_{cont}$  has slightly higher novelty than its counterpart. Thus, its restriction lessens the effect that popularity has on the algorithm.

The serendipity of all approaches is very similar, with CBF and MP being slightly less surprising. Apparently, the slightly lower serendipity of CBF is a result of both the metric and the approach using the content and a similarity function

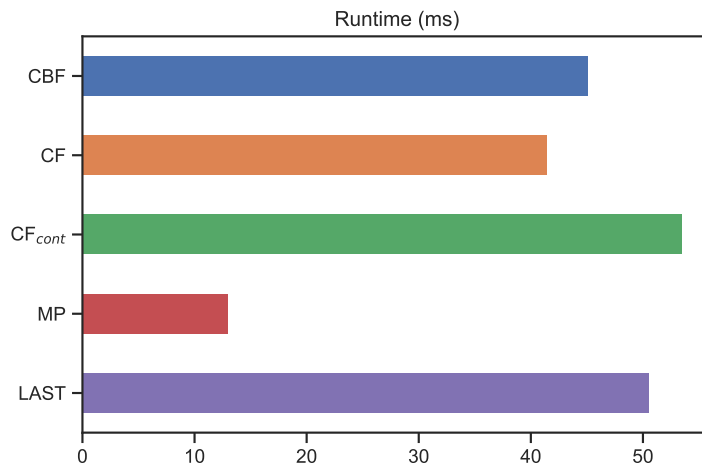


Figure 7.3: Comparison of the runtime performance. MP is the quickest to calculate. The other methods are all pretty equal. All of them can be used for real-time recommendations.

for the calculation. Counter-intuitively, MP also leads to highly surprising recommendations. This can be attributed to the fact that MP does neither consider the content nor the user history.

### 7.1.3 Runtime

Figure 7.3 shows the average time in milliseconds to calculate the results per recommendation request. Unsurprisingly, the MP algorithm is by far the quickest. The other four lie relatively close to each other. All algorithms compute the results in about  $50ms$ , which makes all of them suitable for the task at hand.

There are some interesting findings, though. The CF is quicker than the CBF approach. Given the accuracy of this approach, this makes it not only more accurate but also faster. The LAST approach is only slightly slower than the CBF approach. Finally, the CF<sub>cont</sub> is the slowest of them all. This is interesting as this rather simple algorithm needs more time than the LAST approach, but this can be explained by the additional filtering step. Note that all of the results have to be taken with a grain of salt. The results cannot consider the varying calculation time, depending on the implementation and underlying architecture.

## 7 Results

### 7.1.4 Considering Frequency and Recency

The previous experiments only evaluated the LAST strategy of the vector-based approaches. LAST only considers the most recent job interaction. Here, we also consider the frequency of job interactions by utilizing the AVG approach. Moreover, we also evaluate the incorporation of the BLL equation into recommendations.

This part of the evaluation has already been published on the WSDM 2018 conference (Lacic, Kowald, Reiter-Haas, et al., 2018). Here, we present a summary of this publication. The work mainly focuses on the effects of frequency and recency of job interactions. Moreover, it also explores the effects of dimensionality on the results. The evaluation uses three baseline approaches (i.e., MP, CBF, and CF) and compares them against multiple vector-based approaches. The vector-based approaches are trained on 100, 200, and 300 dimensions. As already stated, we consider all three aggregation strategies (i.e., LAST, AVG, and BLL). Additionally, a mixed hybrid model combining the BLL and CF model is considered. The evaluation protocol also differs slightly. It considers recommending 3 and 6 jobs. The results are evaluated on nDCG, novelty, and diversity. Additionally, an adapted novelty measure (i.e., Novelty\*) is evaluated. This measure considers how close the novelty of the recommendations is to the novelty value where most applications happen (i.e., the target novelty  $N_A$  of 0.58).

Table 7.2 shows the results of the publication. We found both MP and CF are far from the target novelty since their novelty is very low. CBF, on the other hand, even overshoots the target novelty. Hence, the slightly lower novelty of LAST is desired. Using AVG, which considers frequency only, decreases accuracy but increases diversity compared to LAST. Actually, it leads to the lowest accuracy and highest diversity. It also further increases the already very high novelty of LAST, which is undesired. These findings suggest a trade-off between frequency and recency is needed.

The BLL approach balances the effects of both extremes. It leads to just slightly less accuracy than LAST and just slightly less diversity than AVG. Similarly to LAST, it outperforms CBF on both those metrics. The novelty metric stays comparable to LAST (actually BLL with 100 dimensions is closest to the target novelty).

The effects of embedding dimension on the performance are very small. The diversity increases slightly with higher dimensions. Interestingly, a higher dimension typically leads to a decrease in the nDCG metric. The effects on

## 7.1 Offline Evaluation

Approach		k = 3				k = 6				
		Novelty*	Novelty	Diversity	nDCG	Novelty*	Novelty	Diversity	nDCG	
MP		.5849	.1649	.7261	.0395	.6057	.1857	.7156	.0722	
CBF		.8124	.7676	.4536	.0122	.7965	.7835	.4854	.0156	
CF		.7718	.3518	.6736	.0889	.7860	.3660	.6814	.1292	
Doc2Vec	LAST	d=100	.8331	.7469	.4845	.0170	.8161	.7639	.5486	.0217
		d=200	.8411	.7389	.5091	.0182	.8212	.7588	.5854	.0219
		d=300	.8448	.7352	.5163	.0177	.8206	.7594	.5953	.0220
	AVG	d=100	.8275	.7525	.6929	.0107	.8144	.7656	.7239	.0154
		d=200	.7930	.7870	.7455	.0099	.8050	.7750	.7830	.0135
		d=300	.7715	.8085	.7439	.0091	.8003	.7797	.7796	.0133
	BLL	d=100	.8500	.7300	.5974	.0156	.8322	.7478	.6486	.0198
		d=200	.8284	.7516	.6408	.0146	.8275	.7525	.7006	.0188
		d=300	.8191	.7609	.6388	.0144	.8222	.7578	.7015	.0186
CF + BLL		.8731	.4531	.6820	.0721	.9578	.5378	.6890	.0900	

Table 7.2: This table provides the evaluation results of Lacic, Kowald, Reiter-Haas, et al. (2018) and has been previously published in the IFUP 2018: Workshop on multi-dimensional information fusion for user modeling and personalization. It shows that a recommendation approach that uses the BLL equation provides a good balance between accuracy and diversity while achieving the best performance with respect to the target novelty (i.e., the Novelty\* measure).

novelty depend on whether the focus is on frequency or recency. For recency (i.e., LAST), the novelty decreases with higher dimensions. Whereas for frequency (i.e., AVG), this effect is reversed and the novelty increases with the dimension size.

Finally, we use a mixed hybrid by combining BLL with CF in a round-robin fashion. The idea of this trade-off is to use the advantages of both approaches while counteracting the disadvantages. The result of this combination is a much higher accuracy than using only a vector-based approach. The nDCG is comparable to CF alone and thus the combination also outperforms the LAST and MP approach. Combining algorithms typically boosts the diversity of recommendations. Unsurprisingly, the combined diversity is higher than either CF or BLL alone and is only surpassed by MP and AVG. Furthermore, the hybrid model achieves the highest regarding the adapted novelty measure. Thus, we conclude that the combined model provides a good balance between accuracy, diversity, and novelty.

## 7 Results

### 7.1.5 Offline Findings

In summary, both CF and MP have high accuracy and diversity, but a low novelty. The results of CBF are opposite, low accuracy and diversity, but a high novelty. Finally, the LAST performs similar to CBF but improves the diversity and user coverage.

Using the BLL approach provides a good balance between the recency aspect of LAST and the frequency aspect of AVG. CF provides a strong baseline technique for generating recommendations. Finally, a hybrid combination provides a well-rounded model of the different aspects of recommendations.

From this evaluation, we conclude that the best recommendation technique uses a combination of various aspects and approaches. In particular, a model that performs well on all metrics incorporates three types of information. First is the latent information in the job content, which we model with embeddings. Second is the temporal information of the job interactions, which we model with the BLL equation. Last is the similarity information of users, for which we use CF. These findings are put to the test in various online evaluations.

## 7.2 Online Evaluation

The online evaluation first covers the results of all conducted A/B tests on two recommendation scenarios. Subsection 7.2.1 considers the experiments with the Infobox scenario, while Subsection 7.2.2 deals with the Home scenario. Both sections provide findings of various A/B tests and are structured into paragraphs. Each paragraph evaluates one particular A/B test. The name of the test is denoted at the beginning. We visualize the results in figures that show the daily CTR on the left and boxplots of the runtime on the right.

The focus of those experiments lies on statistically significant results (i.e., a p-value lower than 0.05). Nonetheless, we shortly report the results of several other experiments as well. The evaluations use the protocol defined in Section 6.2. Five of these evaluations have already been published on the RecSys 2019 conference (Reiter-Haas, Lacic, Duricic, et al., 2019). Those experiments are marked by a  $\star$  after the test name. The online evaluation is then concluded by presenting the overall findings in Subsection 7.2.3.



### 7.2.1 Infobox Tests

We perform eight experiments within the Infobox scenario. We start by conducting two preliminary experiments on parameter choice for content-based recommendations. First, we evaluate the [Effects of job teasers](#) in CBF. Second, we deal with the [Curse of dimensionality](#) in LAST. Hence, we consider the effects of higher dimensional embeddings on the recommendations. Moreover, we evaluate whether this increase in computational cost is justified. Then, we focus on tests for the vector-based approaches. Thus, we consider the [Impact of embeddings](#) on recommendation performance. The BLL equation is used to evaluate the [Influence of frequency and recency](#) on the results. Experimenting on the time decay parameter shows the [Merit of recency](#) for this scenario. Regarding collaborative filtering, we test whether [Considering context](#) improves the results. This evaluation is performed to choose the correct approach for the following hybrid experiments. To evaluate the performance of embeddings in a hybrid setting, we are [Combining embeddings](#) with collaborative filtering. Finally, we conduct a similar experiment by [Combining frequency and recency](#). This experiment concludes the evaluation for Infobox recommendations.

**Effects of job teasers.** For content-based recommendations, it is important to select the right features to reach a satisfying result. In this preliminary test, we evaluate whether using a teaser text (as described in Chapter 4) is necessary for a good recommendation performance. The test was 30 days long, and 43,653 recommendation requests from 12,543 distinct users were received. The inclusion of the teaser results in a statistically significant increase of 13% in CTR. In particular, considering the teaser results in a CTR of 0.0272, whereas no teaser results in only in a CTR of 0.0240. One detail to note is that the teaser texts are often very similar within the job ads of the same company. Thus, this might result in a lower diversity of the recommendation when a company has multiple jobs online.

**Curse of dimensionality.** In this second test for parameter configuration, we consider the effects of dimensionality. To optimize the embeddings each dimension should encode a meaningful (latent) property. Thus, more dimensions might adversely affect the results. In this experiment, two popular dimension sizes were compared, namely 100 and 300. We conducted a 32 days long test, where 10,018 distinct users performed 45,810 recommendation requests. The results show that the bigger dimension embeddings perform worse in CTR (i.e., the CTR of 100 dimensions is 16% higher than for 300 dimensions). In

## 7 Results

particular, the CTR is 0.0274 for 100 dimensions and 0.0237 for 300 dimensions. This suggests that job ads can already be encoded very well within a low dimensional vector space. Moreover, it suggests that the number of features needed to accurately describe a job ad are rather low and additional dimensions have an adversary effect. Regarding the runtime, the difference is very minimal (i.e., 154ms for 100 and 159ms for 300 dimensions).

**Impact of embeddings\***. An expectation of this master thesis is that using a low dimensional representation of the content can improve content-based recommendations. Therefore, this test compares CBF with LAST. Over the 32 days testing period, 31,968 recommendation requests were performed by 8,576 distinct users. This experiment is the initial test on whether embeddings improve the recommendation performance.

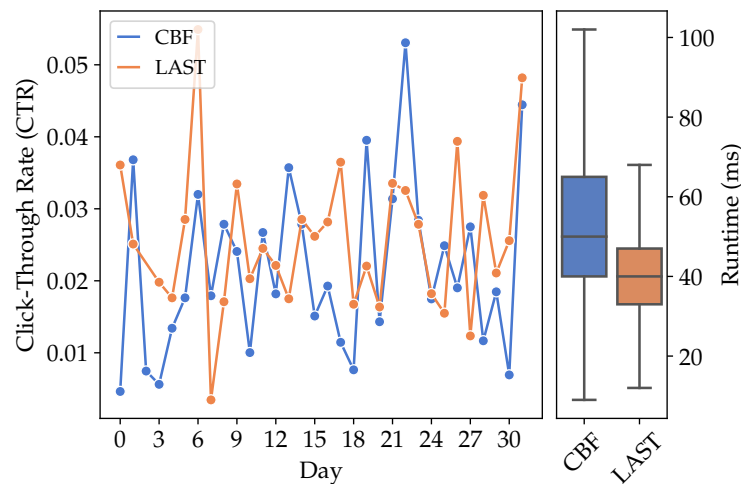


Figure 7.4: Results of Impact of embeddings, which compares the CBF baseline to LAST. The CTR fluctuates greatly but overall LAST outperforms CBF.

Figure 7.4 shows that the performances of both algorithm fluctuate greatly over testing period. Nonetheless, LAST leads to an overall increase of 18% in CTR. Surprisingly, LAST also has a 24% lower runtime. Both results are statistically significant. In particular, LAST has a CTR of 0.0229 and a 39ms runtime, while CBF has a CTR of 0.0194 and a 51ms runtime. This confirms that indeed a vector-based approach is capable of to improve content-based recommendations. This experiment has previously been published on the RecSys 2019 conference (Reiter-Haas, Lacic, Duricic, et al., 2019).

**Influence of frequency and recency\***. The incorporation of the BLL equation applies the temporal aspects of the user history to the embeddings. This experiment builds upon the results of the [Impact of embeddings](#) and compares BLL (with a time decay parameter of 0.5) with LAST. Over the 15 days testing period, 18,464 recommendation requests were performed by 4,715 distinct users.

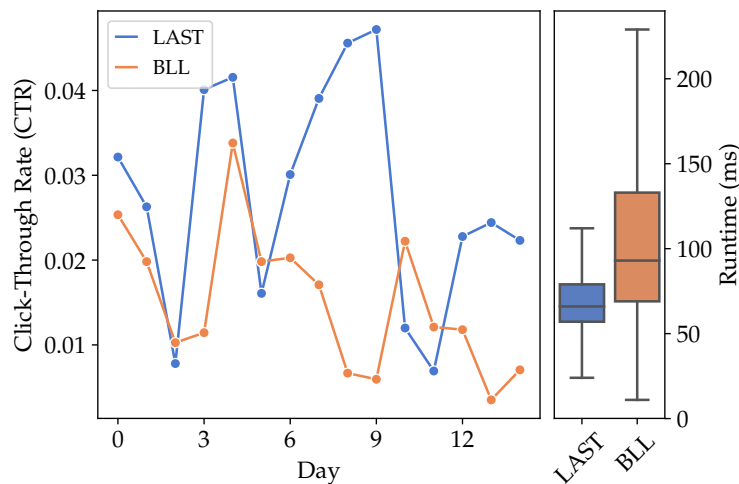


Figure 7.5: Results of Influence of frequency and recency, which compares LAST to BLL. The BLL algorithm could not further improve upon the performance of LAST.

Figure 7.5 shows that BLL could not improve the results, but has the opposite effect. As such, BLL performs worse on both CTR and runtime. In fact, the results are very significant for both measures, as the p-value is below 0.0005. In particular, LAST has a 75% greater CTR and 29% less runtime compared to BLL. The specific results of LAST are a CTR of 0.0249 and a runtime of 67ms. While the CTR is comparable to values of [Impact of embeddings](#), the runtime is a lot higher in this case. This can be attributed to external factors, like server load. This is also the reason A/B tests are performed in the first place, to avoid such irregularities in the results. Considering the results of BLL shows that it only has a very low CTR of 0.0142 and a high runtime of 94ms. These findings contradict the results of the offline evaluation and need further exploration in this regard. This experiment has previously been published on the RecSys 2019 conference (Reiter-Haas, Lacic, Duricic, et al., 2019).

**Merit of recency\***. We suspect that the results of the [Influence of frequency and recency](#) can be attributed to frequency aspects of the BLL equation. Thus, in this experiment, we investigate this parameter in further detail. We use apply the

## 7 Results

parameter to simulate a shorter (i.e.,  $d = 0.6$ ) and longer (i.e.,  $d = 0.4$ ) memory retention. Our hypothesis is that a focus on recency (i.e., the higher time decay parameter) performs better in the given scenario. The test is conducted over a 15 days testing period and received 11,992 recommendation request from 3,375 distinct users.

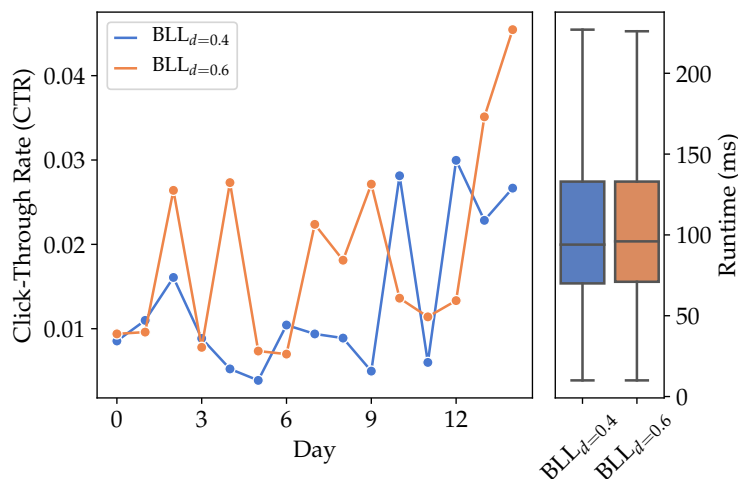


Figure 7.6: Results of Merit of recency, which compares the time decay parameters of 0.4 and 0.6 for BLL. A higher time decay parameter leads to better results. Thus, a focus on recency is better for the Infobox scenario.

Figure 7.6 supports this hypothesis.  $BLL_{d=0.6}$  has a 36% higher CTR than  $BLL_{d=0.4}$ , which is a significant increase. In particular,  $BLL_{d=0.6}$  has a CTR of 0.0174, while  $BLL_{d=0.4}$  has a CTR of 0.0128. Regarding runtime, no statistically significant differences could be found as both configurations perform similarly in speed (i.e.,  $BLL_{d=0.6}$  has a runtime of 97ms, and  $BLL_{d=0.4}$  has a runtime of 95ms). This time, the performances are comparable to the [Influence of frequency and recency](#).  $BLL_{d=0.6}$  performs better, while  $BLL_{d=0.4}$  performs worse on CTR compared to BLL with a time decay of 0.5. Moreover, the runtime performance is also very close. This experiment has previously been published on the RecSys 2019 conference (Reiter-Haas, Lacic, Duricic, et al., 2019).

**Considering context.** This experiment investigates whether considering context (i.e., the current item) is beneficial for collaborative filtering. Thus, it compares CF with  $CF_{Cont.}$ . The 29 days long test received 26,286 recommendation requests from 6,182 different users. We also evaluate several hybrid combinations, which we combine with collaborative filtering, later on.

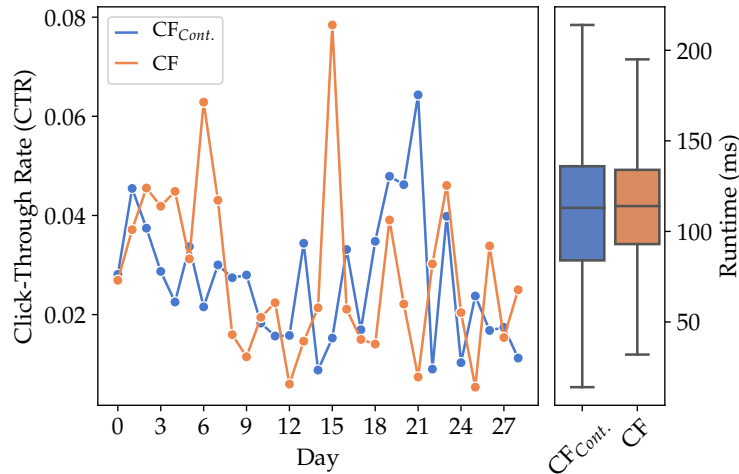


Figure 7.7: Results of Considering context, which compares the context including  $CF_{cont}$  to the simpler CF. Considering the current item for CF does not improve the results.

Unfortunately, no clear (i.e., statistical significant) winner of this test could be found. Figure 7.7 shows that both approaches perform very similar in terms of CTR and runtime. CF has a slightly higher CTR (i.e., an 8% increase) compared to  $CF_{Cont.}$ . But CF also has a slightly higher runtime of  $114ms$  compared to  $111ms$  of  $CF_{Cont.}$ . The lower runtime of  $CF_{Cont.}$  is counteracted by a higher standard variation. In particular, CF has a CTR of 0.0265, while  $CF_{Cont.}$  has a CTR of 0.0244. For runtime, we find that  $CF_{Cont.}$  has a greater variation. We attribute this variation to the differences in the number of users considered for the filtering step. We conclude that the additional complexity that  $CF_{Cont.}$  introduces is not worth the effort, and we stick to CF hereafter.

**Combining embeddings.** We repeat the experiment of the [Impact of embeddings](#) but in a hybrid setting. As such, we combine both CBF and LAST with CF in a round-robin fashion. We denote those approaches as  $HYB_{CBF}$  and  $HYB_{LAST}$ , respectively. Over 31 days, there were 41,937 recommendation requests from 9,434 distinct users.

Figure 7.8 indicates that both combinations have almost identical performance in CTR. Hence, no statistical significant result was found for CTR. As such,  $HYB_{LAST}$  performs only 5% better than  $HYB_{CBF}$  for CTR. Both approaches have a high CTR, as  $HYB_{LAST}$  has 0.0271, and  $HYB_{CBF}$  has 0.0259. We suspect that the performance of CF dominates this combination and leads to the similarity of the results. The runtime is  $125ms$  for  $HYB_{LAST}$  and  $133ms$  for  $HYB_{CBF}$ .

## 7 Results

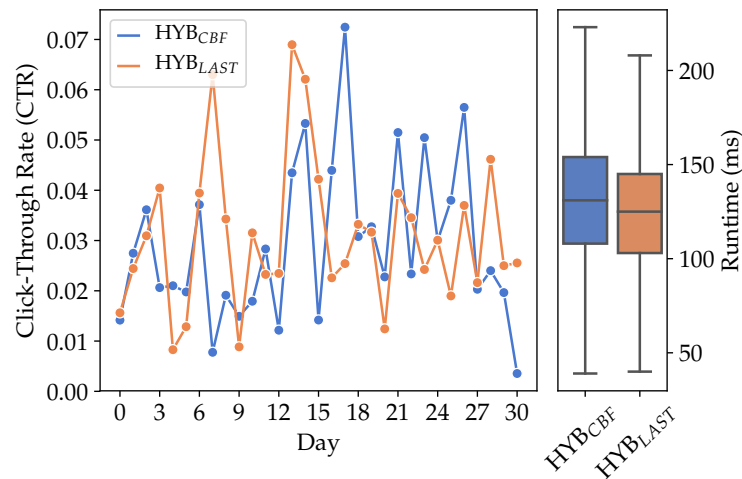


Figure 7.8: Results of Combining embeddings, which compares CBF and LAST, both in a hybrid combination with CF. Both combinations closely resemble each other. Thus, no significant difference is found. A dominance of CF could explain the phenomenon.

**Combining frequency and recency.** The results of the [Influence of frequency and recency](#) indicate that BLL is not well suited for the Infobox scenario. We test this hypothesis again in a hybrid setting, which we set up similar to [Combining embeddings](#). Thus, we compare LAST and BLL in hybrid combination with CF. We denote the two approaches as  $\text{HYB}_{\text{LAST}}$  and  $\text{HYB}_{\text{BLL}}$ , respectively. The test took 20 days and handled 18,824 recommendation requests from 4,951 distinct users.

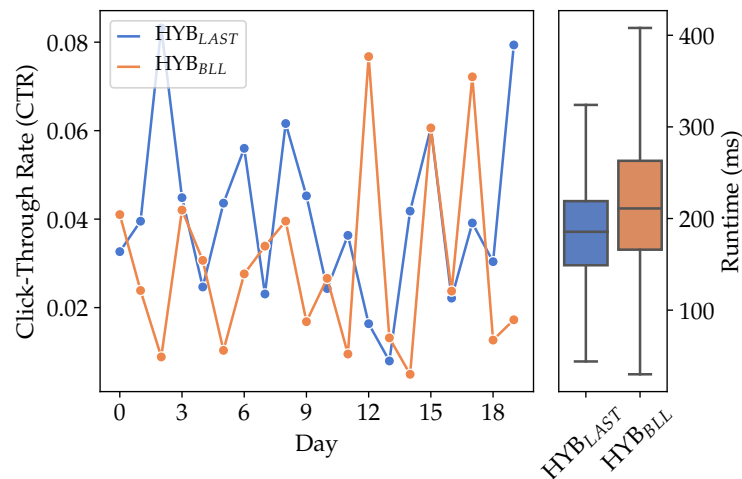


Figure 7.9: Results of Combining frequency and recency which compares LAST and BLL, both in a hybrid combination with CF. Even in a hybrid setting BLL with a time decay of 0.5 does not seem suited for the Infobox scenario.

Figure 7.9 again confirms our hypothesis, as  $\text{HYB}_{\text{LAST}}$  performs 35% better on CTR, which is a statically significant increase. In particular, the results are a CTR of 0.0364 for  $\text{HYB}_{\text{LAST}}$  and 0.0270 for  $\text{HYB}_{\text{BLL}}$ .  $\text{HYB}_{\text{LAST}}$  has runtime of 184ms, and  $\text{HYB}_{\text{BLL}}$  has 220ms. Comparing the results to testing results of [Combining embeddings](#), we see that in this case, CF was not able to leverage the performance to a similar level. Thus, we conclude that BLL, with its incorporation of frequency, is indeed unsuited for the task.

## 7.2.2 Home Tests

For the Home scenario, we conduct three experiments. Similar to the Infobox scenario, we evaluate the [Influence of frequency and recency](#) on the recommendation performance and effects of [Combining frequency and recency](#) with collaborative filtering. Thus, we also compare their results against the Infobox scenario. Additionally, in a small experiment, we evaluate the effects of [Considering demographics](#) on the recommendation performance.

**Influence of frequency and recency\***. In the [Influence of frequency and recency](#) of the Infobox scenario in Subsection 7.2.1, we report that BLL performs low. We suspect that this poor performance is a particularity of the Infobox scenario. Thus, we conduct another test with BLL but for the Home scenario. The test scenario is similar to the previous one. However, this time we compare against another baseline (i.e., CF which is the default for the Home scenario). During the 25 days of the test, 26,334 recommendation requests were performed by 9,620 distinct users.

Figure 7.10, indeed, shows that BLL outperforms CF as it leads to a 16% higher CTR and 14% lower runtime. Both differences are statistically significant and confirm our expectations. In particular, BLL has a CTR of 0.0671 and runtime of 114ms, while CF has a CTR of 0.0580 and a runtime of 132ms. Note that in this scenario, the CTR for both is significantly higher compared to the Infobox scenario. This is a result of the more prominent placement of the recommendations for this scenario. Unlike the results of the Home scenario, here, BLL poses a strong technique for generating recommendations, as suggested by the offline evaluation. This indicates that the offline evaluation tends to replicate the results of the Home scenario more closely. This is no surprise as most interactions happen on the Home page. Interestingly, for the Home scenario, BLL works far better than predicted by the offline evaluation. Contrary to the offline evaluation, where CF had a much higher nDCG compared to BLL,

## 7 Results

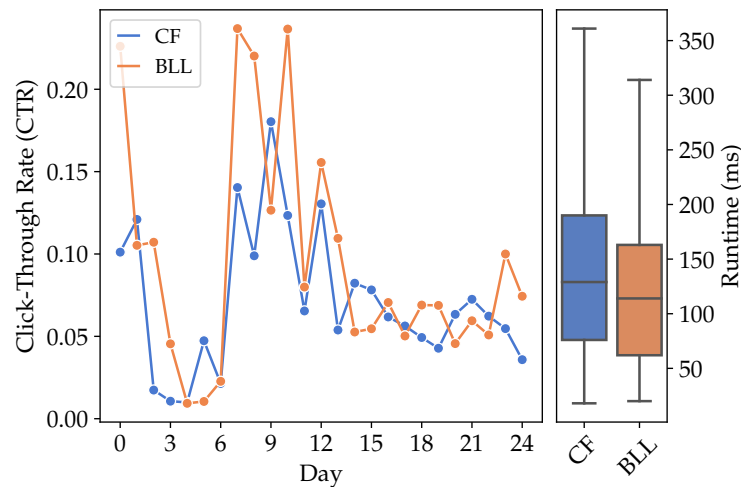


Figure 7.10: Results of Influence of frequency and recency, which compares the CF baseline to BLL. In the Home scenario, BLL performs even better than CF.

for this test, the CTR of BLL is higher. This experiment has previously been published on the RecSys 2019 conference (Reiter-Haas, Lacic, Duricic, et al., 2019).

**Combining frequency and recency\***. We know from the [Influence of frequency and recency](#) that BLL is very suitable for the scenario. Furthermore, in the offline evaluations, a combination of BLL and CF provides the overall best results. Here, we investigate whether this combination, which we denote as  $\text{HYB}_{\text{BLL}}$ , still improves upon CF. We conduct the test over a 19 days period, where the recommender received 24,907 recommendation requests from 9,313 distinct users.

Figure 7.11 confirms our intuition. The combination of BLL and CF (i.e.,  $\text{HYB}_{\text{BLL}}$ ) leads to a 33% increase in CTR but comes at the cost of a 38% increase in runtime. In particular,  $\text{HYB}_{\text{BLL}}$  has CTR of 0.0471 and a 172ms runtime, whereas, CF has CTR of 0.0354 and a 106ms runtime. The results on both metrics are very significant (i.e., have a p-value less than 0.0005). Hence, the difference in CTR is much greater this time compared to the [Influence of frequency and recency](#). We conclude that the combination of BLL and CF is indeed well suited for the task. However, you have to account for the higher computational effort required for such a combination. This experiment has previously been published on the RecSys 2019 conference (Reiter-Haas, Lacic, Duricic, et al., 2019).



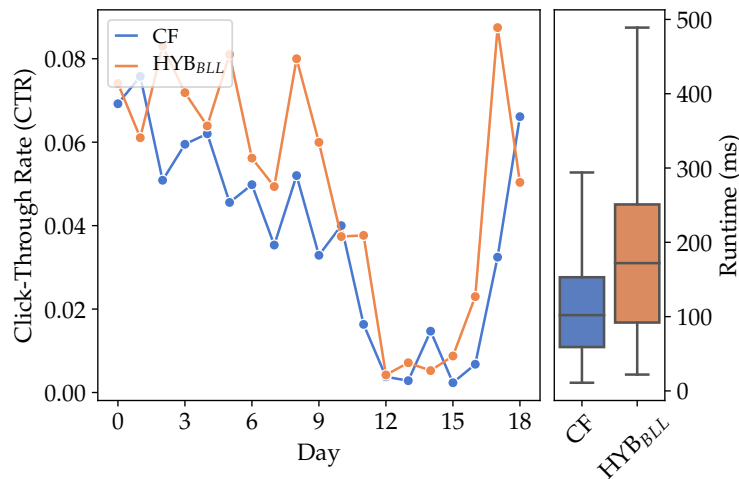


Figure 7.11: Results of Combining frequency and recency, which compares the CF baseline to a hybrid combination of BLL with CF. The results show that the hybrid combination with BLL improve upon the CF baseline.

**Considering demographics.** Until now, the user CV data (i.e., demographic data) is not considered for the recommendations. Hence, we conduct one final experiment with a limited scope. In this experiment, we utilize the data of the user CV and use the User2Vec algorithm for generating recommendations. However, the algorithm can only generate recommendations for users that have content data in their CVs. When splitting the user base for the A/B test, we put half of the users with CV in the User2Vec group, and the other half is joined by the users without CV data. Over 13 days, 6,252 recommendations requests were handled. The experiments show that the recommendations for the users in the User2Vec group perform better (i.e., a 63% increase in CTR). In particular, User2Vec has a very high CTR of 0.1178, while CF also has a competitive CTR of 0.0722. The result indicates that there is indeed potential to consider the user CV as well for the recommendations. However, there is a big consideration of the experimental setup. The setup does not account that the choice in splitting the user base leads to this difference in recommendation performance. Hence, users that take the effort to fill out their CVs could generally be more engaged with the platform. Thus, User2Vec could even have a negative impact on the performance compared to CF. We, therefore, conclude the approach shows potential, but further investigation is indeed needed.

## 7 Results

### 7.2.3 Online Findings

The findings of the online evaluations show that vector-based approaches are suitable for both scenarios (i.e., Infobox and Home). However, the temporal aspects vary depending on the recommendation scenario in question. The results of Infobox scenario (Subsection 7.2.1) indicate that a focus on recency works best. Whereas, the results of the Home scenario (Subsection 7.2.2) leans more towards a focus on frequency. Additionally, for both scenarios, a combination with CF leads to top performances. We applied these findings to the platform and rolled the two winning combinations (i.e.,  $\text{HYB}_{LAST}$  for the Infobox and  $\text{HYB}_{BLL}$  for the Home scenario) out on their respective scenarios.

We also performed several tests (i.e., [Effects of job teasers](#), [Curse of dimensionality](#), and [Considering context](#)) regarding parameter choice on the Infobox scenario. While there were some statistically significant differences, there are no particular interesting findings in this regard. This suggests that these do not matter a lot in a practical setting compared to the choice of algorithms. We, therefore, conclude that it is typically better to just use the simpler model (e.g. use embeddings of 100 instead of higher ones).

Finally, we also shortly explored the potential of the user data for generating recommendations. The results of [Considering demographics](#) are indeed promising but not very conclusive. While a further investigation is out of scope for this thesis, we suggest considering demographic data as another source of information for recommendations. Moreover, the findings suggest that considering many different types of information is a key factor for good recommendation performance.

## 7.3 Comparison and Summarization

Here, we compare the findings of the offline evaluation in Section 7.1 and the online evaluation in Section 7.2. The results indicate many similarities between the two types of evaluation. The findings of both suggest that a combination of a vector-based approach with CF works best. Both evaluations also suggest that the temporal information provided by BLL greatly influences the results. However, the findings disagree on whether the focus should lie on the frequency or recency aspect for a high recommendation performance. For the offline evaluation, the balance of frequency and recency in BLL suggests overall good performance. Contrary, for the online evaluation, the results of BLL depend greatly on the chosen scenario. For the Infobox scenario, the

recency aspect of vector-based approaches dominates. Whereas, for the Home scenario, using BLL with its frequency aspect even outperforms CF, which is not predicted by the offline evaluation.

Comparing the differences in the evaluation procedures also leads to another interesting finding. For measuring the accuracy of the recommendations, we use mainly nDCG, which acts as a surrogate for the other accuracy measures, for the offline and CTR for the online evaluation. While the results of the measures are typically comparable across the evaluations, they measure very different aspects. The nDCG considers the problem completely in isolation of other factors as it only considers relevancy. Thus, just returning only relevant items leads to the perfect value of 1. The other factors, which we measure in the beyond-accuracy metrics, are not included. As stated previously, a low novelty and diversity might lead to a low user satisfaction (Kenthapadi, B. Le, and Venkataraman, 2017; Smyth and McClave, 2001). The CTR measure is completely different in this regard. There, a perfect value is often not feasible as users likely cannot interact with all recommendations. Moreover, CTR also indirectly captures beyond-accuracy factors. As such, a too low diversity that negatively affects user satisfaction is also reflected in the CTR. We observe that the proposed combination of the offline evaluation, which provides a good balance between novelty, diversity, and accuracy, also performs well in the online evaluation. This indicates that beyond-accuracy measures are of great importance for evaluation purposes.

We summarize our findings in three key takeaways. First, the results seem to suggest that using multiple different aspects of information provides the best results. In our case, we use temporal, content, and similarity aspects. In particular, we model these three aspects as following: (i) we convert the content data into embeddings, (ii) we include the temporal data with the BLL equation, and (iii) we create a hybrid combination with CF that considers the user similarity. Second, we find that an offline evaluation already provides a good estimate for the real performance of recommendations but cannot easily consider the differences in the recommendation scenario. Thus, these estimations need to be put to the test in an online evaluation. Third, we find that algorithms that perform well on beyond-accuracy metrics in an offline evaluation also perform well in reality. Hence, we argue that beyond-accuracy plays a vital role in evaluating the performance of recommendations and captures important aspects that accuracy based measures do not account for.

Conclusively, for the Studo Jobs platform, we propose to use a hybrid combination of LAST and CF for the Infobox scenario and a combination of BLL and CF for the Home scenario.



## 8 Conclusion

This thesis tackled the problem of providing job recommendations to students. For this purpose, we answered two research questions. The first research question (i.e., *RQ1*) tackles the problem of how to improve job recommendations with latent features of job postings. While the second research question (i.e., *RQ2*) explores the configuration and combination of algorithms that provide the best result in student job recommendations. We answered those questions by conducting offline and online evaluations, respectively.

Regarding *RQ1*, the results show that deep learning can be an effective enhancement to the recommendation use case. This thesis successfully applies a deep learning method for the job domain. The applied deep learning approach outperforms the traditional content-based methods (e.g., the TF-IDF model). Additionally, the inclusion of time decay (i.e., by using the BLL equation) when modeling latent features further improves the results according to accuracy. Moreover, considering the frequency of job interactions improves the diversity of the generated recommendations. Regarding novelty, the results show that content-based approaches provide highly novel recommendations in general. Furthermore, the results suggest that a too high novelty leads to user dissatisfaction. This novelty can be balanced out by combining the content-based approach with, for instance, collaborative filtering (CF). This, in the end, results in having highly accurate and diverse results, as the offline evaluation suggests.

Building upon these results, to answer *RQ2* we conducted an online evaluation on two recommendation scenarios. First, the Infobox scenario where recommendations are shown near additional job information, and second, the Home scenario where recommendations are shown on the Home view. For both scenarios, the results could improve upon the baseline by using the embeddings for the recommendations. However, the results also show that the importance of considering the frequency of job interactions depends on the recommendation use case. For the Infobox recommendations, there is much more weight on recent items. Thus, a similar item recommender already performs well in this scenario. However, for the Home recommendations, the results could be further improved by incorporating a balance of frequency and recency, as is

## 8 Conclusion

suggested by the results for  $RQ_1$ . Since there are many more interactions for the index scenario than are for the sidebar scenario, these results agree with the offline evaluation. Although, considering the offline evaluation alone leads to the appearance that the BLL equation is superior in general. This shows the importance of conducting experiments in an online setting, to alleviate factors simulations do not account for. Additionally, the CF approach can be used to further boost the result for both scenarios. This also suggests that CF is well suited for providing recommendations in the job domain.

Finally, the findings of this master thesis have been applied to the Studo Jobs platform. Both, the Infobox and the Home, scenarios employ a combination of a vector-based approach with CF. In addition to the evaluation, several students reported to us that they have easily found their job on the platform, which further supports the benefits of applying the recommender system for this use case.

### 8.1 Reflections

There are several considerations for this thesis. These arise mainly due to the evaluation being carried out on a real platform. The main consideration is that the dataset that was used in this thesis is a proprietary dataset of Moshbit and thus cannot be released to the public, which limits reproducibility.

**Testing environment.** The advantage of being able to carry out the experiments on a real platform comes at the expense of possible changes to user behaviour on the platform. These changes emerge due to several factors. First, the platform received many alterations over the whole testing period. For this reason, there was a lot of emphasis on keeping the environment controlled whenever an online evaluation was running. Thus, major updates were deployed only between the different tests, while only minor and critical updates happened during testing periods. While this ensures that the A/B test is consistent in itself, the tests cannot be easily compared with one another. Other factors to consider are seasonal changes and trends of user behaviour. However, this factor needs to be accounted for in a real application by repeating the experiments with the same or a similar setup after some time has passed.

**Different testing periods.** Another related problem is the difference in testing periods. This problem can also be attributed to the practical setup. As stated, the environment was being kept controlled regarding product changes and releases of updates have been timed accordingly. However, other factors affect the timing of releases like critical bugs or the day of the week, which were accounted for by adjusting the testing period.

**Small dataset.** Another consideration is regarding the choice of the dataset. This thesis focused on only one rather small dataset, which limits the generality of the statements. However, the publication on offline evaluations (which we also report in Lacic, Kowald, Reiter-Haas, et al. (2018)) were additionally carried on the larger dataset from the RecSys 2017 challenge. This supports the evaluation done for *RQ1*. The evaluation for *RQ2* cannot easily be conducted in another real application and thus has been chosen to be specific for Studo Jobs.

**Evaluation metrics.** Finally, while this thesis covered the evaluation from a wide variety of different angles, there is a wide variety of improvements that could be included as well. For instance, considerations include newer evaluation metrics and more experiments on parameter tuning. These are out of scope for the current work due to the broad range that it already covers. However, many future extensions to this thesis are being outlined in Section 8.2 that should help to further improve the recommendation quality.

## 8.2 Future Work

In this master thesis, a wide range of experiments were conducted over a long period to answer the two research questions. However, some of the results generated new questions that can be explored. One such thing is how the behaviour of the various types of users differ from each other. For instance, whether users that fill out a CV are generally more engaged with the platform or how the behaviour of app and web users differ. The interaction data provide a vast source of information that should be explored in more detail. Moreover, the results also indicate a potential for including the demographic data of the user CV, which needs to be further explored.

While this master thesis explored the conversion of text into low dimensional embeddings via paragraph vectors, there are many more ways in which similar

## 8 Conclusion

results could be achieved. The use of autoencoders is another very popular method to construct embeddings. Unlike paragraph vectors, which use sequential data, autoencoders can encode arbitrary data. Recently, network embeddings have gained much momentum. For instance, one approach tackles the problem of encoding heterogeneous entities into the same vector space (C.-J. Wang, T.-H. Wang, Yang, et al., 2017). This allows for heterogeneous retrieval of items by connecting concepts shared between them. Originally, the authors used the approach for text to item retrieval, as is the case for the search functionality. However, it could also be used for recommender systems. In our case, this would mean, given a user profile retrieve relevant jobs for that user. The concepts could be modelled by the skills required for the job.

While this thesis focused on job recommendation, other entities can be recommended in this domain as well. Instead of recommending jobs, companies themselves could be recommended. This would target the business case of employer branding. Since employees are hard to find, companies need to establish themselves as an attractive employment option. Furthermore, the process could also be reversed and companies could get recommendations for potential job candidates. This could either be done to recommend a candidate that matches a job or a company.

Besides the recommendation, the manual search functionality is another often used feature of the Studo Jobs that users use to find the right jobs. The alternatives, which are filtering and browsing of job postings, are being phased out slowly as the number of jobs on the platform grows. Since search and recommendations are very similar conceptually, it might make sense to combine those two options to enable a personalized search. This personalized search could be achieved in multiple ways. One way would be to just re-rank the search results based on the recommendation. For instance, if a job would have been recommended and is also in the search result, then rank this job higher than the other results.

Another area that should be explored is how gender influences the system and whether gender bias is prevalent within the recommendations. While gender is not considered for the recommendations themselves, the algorithm could learn a bias that exists within the data. This is due to humans not being unbiased, which is the source of the data. If such a bias exists, then appropriate steps should be taken to eliminate it from the recommendations. Moreover, gender bias is just one example of many different kinds of biases, like racial biases, that could exist in the system.

Finally, this thesis explored several beyond-accuracy metrics for recommender



## 8.2 Future Work

systems, but it interprets these metrics separately from the accuracy metrics. Vargas and Castells (2011) proposed metrics that integrate the rank and relevance into novelty and serendipity metrics. Thus, their metrics allow for a more holistic view of the system and should be used in the future.

As a final remark, the Studo Jobs platform was superseded by the Talto career platform. Thus, all future work needs to be carried out on Talto instead.



# Bibliography

- Abel, Fabian (2015). "We know where you should work next summer: job recommendations." In: *Proceedings of the 9th ACM Conference on Recommender Systems*. ACM, pp. 230–230 (cit. on p. 2).
- Abel, Fabian, András Benczúr, Daniel Kohlsdorf, Martha Larson, and Róbert Pálovics (2016). "Recsys challenge 2016: Job recommendations." In: *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, pp. 425–426 (cit. on pp. 2, 8).
- Abel, Fabian, Yashar Deldjoo, Mehdi Elahi, and Daniel Kohlsdorf (2017). "Recsys challenge 2017: Offline and online evaluation." In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, pp. 372–373 (cit. on pp. 2, 8).
- Aggarwal, Charu C (2016a). "Content-based recommender systems." In: *Recommender systems*. Springer, pp. 139–166 (cit. on p. 13).
- Aggarwal, Charu C (2016b). "Neighborhood-based collaborative filtering." In: *Recommender Systems*. Springer, pp. 29–70 (cit. on pp. 6, 14).
- Anderson, John R et al. (2004). "An integrated theory of the mind." In: *Psychological review* 111.4, p. 1036 (cit. on pp. 3, 17).
- Barkan, Oren and Noam Koenigstein (2016). "Item2vec: neural item embedding for collaborative filtering." In: *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, pp. 1–6 (cit. on p. 10).
- Bennett, James, Stan Lanning, et al. (2007). "The netflix prize." In: *Proceedings of KDD cup and workshop*. Vol. 2007. New York, NY, USA., p. 35 (cit. on p. 1).
- Bishop, Christopher M (2006). *Pattern recognition and machine learning*. Springer Science+ Business Media (cit. on p. 9).
- Böhm, Stephan (2013). "Behavior and expectations of mobile job seekers: an industry study focusing on job boards." In: *Proceedings of the 2013 annual conference on Computers and people research*. ACM, pp. 105–110 (cit. on pp. 1, 24).
- Burke, Robin (2002). "Hybrid recommender systems: Survey and experiments." In: *User modeling and user-adapted interaction* 12.4, pp. 331–370 (cit. on p. 6).
- Burke, Robin (2007). "Hybrid web recommender systems." In: *The adaptive web*. Springer, pp. 377–408 (cit. on p. 18).

## Bibliography

- Chandramouli, Badrish, Justin J Levandoski, Ahmed Eldawy, and Mohamed F Mokbel (2011). "StreamRec: a real-time recommender system." In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, pp. 1243–1246 (cit. on p. 7).
- Chatzis, Sotirios P, Panayiotis Christodoulou, and Andreas S Andreou (2017). "Recurrent latent variable networks for session-based recommendation." In: *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*. ACM, pp. 38–45 (cit. on p. 11).
- Chen, Chen, Hongzhi Yin, Junjie Yao, and Bin Cui (2013). "Terec: A temporal recommender system over tweet stream." In: *Proceedings of the VLDB Endowment* 6.12, pp. 1254–1257 (cit. on p. 7).
- Chen, Tianqi and Carlos Guestrin (2016). "Xgboost: A scalable tree boosting system." In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, pp. 785–794 (cit. on p. 8).
- Covington, Paul, Jay Adams, and Emre Sargin (2016). "Deep neural networks for youtube recommendations." In: *Proceedings of the 10th ACM conference on recommender systems*. ACM, pp. 191–198 (cit. on p. 10).
- Deng, Li, Dong Yu, et al. (2014). "Deep learning: methods and applications." In: *Foundations and Trends® in Signal Processing* 7.3–4, pp. 197–387 (cit. on p. 9).
- Eksombatchai, Chantat et al. (2018). "Pixie: A system for recommending 3+ billion items to 200+ million users in real-time." In: *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, pp. 1775–1784 (cit. on p. 37).
- Friedman, Jerome H (2002). "Stochastic gradient boosting." In: *Computational Statistics & Data Analysis* 38.4, pp. 367–378 (cit. on p. 8).
- Ge, Mouzhi, Carla Delgado-Battenfeld, and Dietmar Jannach (2010). "Beyond accuracy: evaluating recommender systems by coverage and serendipity." In: *Proceedings of the fourth ACM conference on Recommender systems*. ACM, pp. 257–260 (cit. on p. 7).
- Gilbert, Seth and Nancy Lynch (2002). "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services." In: *Acm Sigact News* 33.2, pp. 51–59 (cit. on p. 35).
- Greenstein-Messica, Asnat, Lior Rokach, and Michael Friedman (2017). "Session-based recommendations using item embedding." In: *Proceedings of the 22nd International Conference on Intelligent User Interfaces*. ACM, pp. 629–633 (cit. on p. 10).
- Hidasi, Balázs, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk (2015). "Session-based recommendations with recurrent neural networks." In: *arXiv preprint arXiv:1511.06939* (cit. on pp. 10, 11).
- Horvath, Stefanie (2018). "Keep It Lean." MA thesis (cit. on p. 24).

- Huang, Yanbo (2016). "Exploiting Embedding in Content-Based Recommender systems." MA thesis (cit. on pp. 10, 11, 15).
- Huang, Yanxiang, Bin Cui, Wenyu Zhang, Jie Jiang, and Ying Xu (2015). "Tencent: Real-time stream recommendation in practice." In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, pp. 227–238 (cit. on p. 7).
- Jones, Janelle, John Schmitt, et al. (2014). "A college degree is no guarantee." In: *Center for Economic and Policy Research (CEPR)* (cit. on p. 1).
- Karatzoglou, Alexandros et al. (2016). "RecSys' 16 Workshop on Deep Learning for Recommender Systems (DLRS)." In: *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, pp. 415–416 (cit. on p. 10).
- Kenthapadi, Krishnaram, Benjamin Le, and Ganesh Venkataraman (2017). "Personalized job recommendation system at linkedin: Practical challenges and lessons learned." In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, pp. 346–347 (cit. on pp. 2, 67).
- Koren, Yehuda and Robert Bell (2015). "Advances in collaborative filtering." In: *Recommender systems handbook*. Springer, pp. 77–118 (cit. on pp. 7, 13, 41).
- Kowald, Dominik, Emanuel Lacic, and Christoph Trattner (2014). "Tagrec: Towards a standardized tag recommender benchmarking framework." In: *Proceedings of the 25th ACM conference on Hypertext and social media*. ACM, pp. 305–307 (cit. on p. 42).
- Kowald, Dominik, Subhash Chandra Pujari, and Elisabeth Lex (2017). "Temporal effects on hashtag reuse in twitter: A cognitive-inspired hashtag recommendation approach." In: *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, pp. 1401–1410 (cit. on pp. 7, 18).
- Lacic, Emanuel, Dominik Kowald, Markus Reiter-Haas, Valentin Slawicek, and Elisabeth Lex (2018). "Beyond Accuracy Optimization: On the Value of Item Embeddings for Student Job Recommendations." In: *International Workshop on Multi-dimensional Information Fusion for User Modeling and Personalization (IFUP'2018) co-located with the 11th ACM International Conference on Web Search and Data Mining (WSDM'2018)* (cit. on pp. 4, 11, 42, 45, 49, 54, 55, 71).
- Lacic, Emanuel, Matthias Traub, Dominik Kowald, and Elisabeth Lex (2015). "Scar: Towards a real-time recommender framework following the microservices architecture." In: *Proc. of LSRS 15* (cit. on p. 33).
- Lau, Jey Han and Timothy Baldwin (2016). "An empirical evaluation of doc2vec with practical insights into document embedding generation." In: *arXiv preprint arXiv:1607.05368* (cit. on p. 10).
- Le, Quoc and Tomas Mikolov (2014). "Distributed representations of sentences and documents." In: *Proc. of ICML'14* (cit. on pp. 3, 10, 15, 17).

## Bibliography

- Lee, Yeon-Chang, Jiwon Hong, Sang-Wook Kim, Sheng Gao, and Ji-Yong Hwang (2015). "On recommending job openings." In: *Proceedings of the 26th ACM Conference on Hypertext & Social Media*. ACM, pp. 331–332 (cit. on p. 2).
- Lewis, James and Martin Fowler (2014). "Microservices: a definition of this new architectural term." In: *MartinFowler.com* 25. URL: <https://martinfowler.com/articles/microservices.html> (visited on 05/17/2018) (cit. on p. 33).
- Liu, Rui, Wenge Rong, Yuanxin Ouyang, and Zhang Xiong (2017). "A hierarchical similarity based job recommendation service framework for university students." In: *Frontiers of Computer Science* 11.5, pp. 912–922 (cit. on pp. 1, 2).
- Lops, Pasquale, Marco De Gemmis, and Giovanni Semeraro (2011). "Content-based recommender systems: State of the art and trends." In: *Recommender systems handbook*. Springer, pp. 73–105 (cit. on p. 13).
- Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.Nov, pp. 2579–2605 (cit. on p. 38).
- Malinowski, Jochen, Tobias Keim, Oliver Wendt, and Tim Weitzel (2006). "Matching people and jobs: A bilateral recommendation approach." In: *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*. Vol. 6. IEEE, pp. 137c–137c (cit. on p. 9).
- Max Roser, Hannah Ritchie and Esteban Ortiz-Ospina (2019). "World Population Growth." In: *Our World in Data*. <https://ourworldindata.org/world-population-growth> (cit. on p. 1).
- McInnes, Leland, John Healy, and James Melville (2018). "Umap: Uniform manifold approximation and projection for dimension reduction." In: *arXiv preprint arXiv:1802.03426* (cit. on p. 39).
- Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). "Efficient estimation of word representations in vector space." In: *arXiv preprint arXiv:1301.3781* (cit. on p. 10).
- Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean (2013). "Distributed representations of words and phrases and their compositionality." In: *Advances in neural information processing systems*, pp. 3111–3119 (cit. on p. 15).
- Mishra, Sonu K and Manoj Reddy (2016). "A bottom-up approach to job recommendation system." In: *Proceedings of the Recommender Systems Challenge*. ACM, p. 4 (cit. on pp. 2, 8).
- Mitchell, Brian (1998). *International historical statistics: Europe 1750-1993*. Springer (cit. on p. 1).
- Parra, Denis and Shaghayegh Sahebi (2013). "Recommender systems: Sources of knowledge and evaluation metrics." In: *Advanced techniques in web intelligence-2*. Springer, pp. 149–175 (cit. on p. 7).

- Pazzani, Michael J and Daniel Billsus (2007). "Content-based recommendation systems." In: *The adaptive web*. Springer, pp. 325–341 (cit. on pp. 5, 13).
- Pearson, Karl (1900). "X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling." In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50.302, pp. 157–175 (cit. on p. 47).
- Petrov, Alexander A (2006). "Computationally efficient approximation of the base-level learning equation in ACT-R." In: *Proceedings of the seventh international conference on cognitive modeling*. Citeseer, pp. 391–392 (cit. on p. 17).
- Pizzato, Luiz et al. (2010). "Reciprocal recommenders." In: *ITWP 2010*, p. 53 (cit. on p. 8).
- Pu, Pearl, Li Chen, and Rong Hu (2011). "A user-centric evaluation framework for recommender systems." In: *Proceedings of the fifth ACM conference on Recommender systems*. ACM, pp. 157–164 (cit. on p. 44).
- Qin, Chuan et al. (2018). "Enhancing person-job fit for talent recruitment: An ability-aware neural network approach." In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, pp. 25–34 (cit. on pp. 2, 8).
- Rajaraman, Anand and Jeffrey David Ullman (2011). *Mining of massive datasets*. Cambridge University Press (cit. on p. 13).
- Reiter-Haas, Markus, Emanuel Lacic, Tomislav Duricic, Valentin Slawicek, and Elisabeth Lex (2019). "Should we Embed? A Study on the Online Performance of Utilizing Embeddings for Real-Time Job Recommendations." In: *Proceedings of the 13th ACM Conference on Recommender Systems*. ACM, pp. 496–500 (cit. on pp. 4, 11, 37, 56, 58–60, 64).
- Reiter-Haas, Markus, Valentin Slawicek, and Emanuel Lacic (2017). "Studo Jobs: Enriching Data With Predicted Job Labels." In: *Workshop on Recommender Systems and Social Network Analysis (RS-SNA'2017) co-located with i-KNOW'2017* (cit. on pp. 4, 11, 32).
- Ricci, Francesco, Lior Rokach, and Bracha Shapira (2011). "Introduction to recommender systems handbook." In: *Recommender systems handbook*. Springer, pp. 11–14 (cit. on p. 5).
- Robotham, David (2012). "Student part-time employment: characteristics and consequences." In: *Education+ Training* 54.1, pp. 65–75 (cit. on p. 1).
- Shani, Guy and Asela Gunawardana (2011). "Evaluating recommendation systems." In: *Recommender systems handbook*. Springer, pp. 257–297 (cit. on p. 41).
- Smirnova, Elena and Flavian Vasile (2017). "Contextual sequence modeling for recommendation with recurrent neural networks." In: *Proceedings of the 2nd*

## Bibliography

- Workshop on Deep Learning for Recommender Systems*. ACM, pp. 2–9 (cit. on p. 11).
- Smyth, Barry and Paul McClave (2001). “Similarity vs. diversity.” In: *International conference on case-based reasoning*. Springer, pp. 347–361 (cit. on pp. 44, 67).
- Student (1908). “The probable error of a mean.” In: *Biometrika*, pp. 1–25 (cit. on p. 47).
- Vargas, Saúl and Pablo Castells (2011). “Rank and relevance in novelty and diversity metrics for recommender systems.” In: *Proceedings of the fifth ACM conference on Recommender systems*. ACM, pp. 109–116 (cit. on p. 73).
- Vogels, Werner (2009). “Eventually consistent.” In: *Communications of the ACM* 52.1, pp. 40–44 (cit. on p. 35).
- Volkovs, Maksims, Guang Wei Yu, and Tomi Poutanen (2017). “Content-based neighbor models for cold start in recommender systems.” In: *Proceedings of the Recommender Systems Challenge 2017*. ACM, p. 7 (cit. on pp. 8, 24).
- Wang, Chuan-Ju, Ting-Hsiang Wang, Hsiu-Wei Yang, Bo-Sin Chang, and Ming-Feng Tsai (2017). “ICE: Item Concept Embedding via Textual Information.” In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, pp. 85–94 (cit. on p. 72).
- Xiao, Wenming, Xiao Xu, Kang Liang, Junkang Mao, and Jun Wang (2016). “Job recommendation with Hawkes process: an effective solution for RecSys Challenge 2016.” In: *Proceedings of the Recommender Systems Challenge*. ACM, p. 11 (cit. on p. 8).
- Zhang, Shuai, Lina Yao, and Aixin Sun (2017). “Deep learning based recommender system: A survey and new perspectives.” In: *arXiv preprint arXiv:1707.07435* (cit. on p. 10).
- Zhang, Yuan Cao, Diarmuid Ó Séaghdha, Daniele Quercia, and Tamas Jambor (2012). “Auralist: introducing serendipity into music recommendation.” In: *Proceedings of the fifth ACM international conference on Web search and data mining*. ACM, pp. 13–22 (cit. on p. 44).
- Zheng, Yong (2017). “Multi-stakeholder recommendation: Applications and challenges.” In: *arXiv preprint arXiv:1707.08913* (cit. on p. 2).
- Zhou, Tao et al. (2010). “Solving the apparent diversity-accuracy dilemma of recommender systems.” In: *Proceedings of the National Academy of Sciences* 107.10, pp. 4511–4515 (cit. on p. 44).
- Zhu, Chen et al. (2018). “Person-Job Fit: Adapting the Right Talent for the Right Job with Joint Representation Learning.” In: *ACM Transactions on Management Information Systems (TMIS)* 9.3, p. 12 (cit. on pp. 2, 8).