Emanuel Lacić, M.Sc.

Real-time Recommender Systems in Multi-Domain Settings

DISSERTATION

for the attainment of the degree of

Doctor of Engineering Sciences (Dr. techn.)

submitted to

Graz University of Technology

Supervisor:
Assoc.-Prof. Dr. Elisabeth Lex

Graz, 25.05.2022

EIDESSTATTLICHE ERKLÄRUNG

AFFIDAVIT

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit/Diplomarbeit/Dissertation identisch.

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/diploma thesis/doctoral dissertation.

3.5.2022

Datum / Date

Unterschrift / Signature

Abstract

Recommender systems have become an essential tool in supporting users to find relevant content in an overloaded information space. Be it for business, government, or education, the utilization of recommender systems has become popular across many application domains. It remains however an open challenge for the research community to simultaneously deal with multiple domain-specific requirements and information sources. In addition to that, the advent of the modern Web has posed the need for handling frequent and diverse data updates with high scalability and real-time processing, and thus, has brought new challenges that recommender systems need to deal with. It is therefore the aim of this thesis to investigate how real-time recommendations can be provided within a multi-domain setting.

For this, four separate research problems are tackled. As the usage of various information sources is nowadays a popular choice for many application domains, the first research question investigates the impact of recommendation quality with respect to combining different sources of data. An empirical study presented in this thesis shows, for example, that sources like social or location data are especially beneficial for tackling the cold-start problem. The second research question builds upon these insights and addresses the problem of customization, scalability, and real-time performance while supporting heterogeneous information sources across multiple domains. This yields a methodological contribution of design principles and aspects that such systems need to support, as well as two recommendation frameworks that present the underlying architecture suited for a multi-domain setting. The third research question focuses on how to balance the trade-off between accuracy and runtime performance within a real-time setting. The performed scalability experiments demonstrate how to achieve the desired recommendation accuracy while guaranteeing the same runtime performance under different load peaks. Moreover, it is shown how to further speed up the generation of recommendations by, for

example, adapting the popularly utilized Collaborative Filtering algorithm to find neighbors in a greedy fashion or utilizing search engine technology with latent item embeddings. The fourth research question of this thesis tackles the problem of improving real-time recommendations beyond accuracy. This results in an online user study that shows how offline experiments that focus on improving only accuracy provide a limited perspective on the true recommendation utility. Besides, this thesis presents a methodological contribution in form of a novel approach that utilizes neural autoencoders to improve the recommendation quality beyond accuracy for anonymous session users.

Finally, the research performed in this thesis opens up a number of possible research strands for future work on algorithms and systems specifically tailored for real-time recommendations that are applied in a multi-domain setting. These, for instance, include tackling different biases, incorporating privacy mechanisms, approximating online performance as well as providing session-aware real-time recommendations.

Zusammenfassung

Empfehlungssysteme gelten als ein wesentliches Instrument, um Benutzer bei der Suche nach relevanten Inhalten zu unterstützen. Deren Einsatz hat sich deswegen in vielen Anwendungsbereichen, wie zum Beispiel im Handel, für den Staat oder im Bildungswesen, durchgesetzt. Eine Problemstellung für die *Research Community* ist jedoch, wie man in einer Multistakeholder-Umgebung gleichzeitig die verschiedenen bereichsspezifischen Anforderungen und Informationsquellen berücksichtigen kann. Darüber hinaus hat das modernen Web neue Herausforderungen für Empfehlungssysteme mit sich gebracht, wie die Notwendigkeit, häufige und vielfältige Datenupdates hoch skalierbar und in Echtzeit handzuhaben. Das Ziel dieser Arbeit ist es, zu untersuchen, wie Empfehlungen in einer Multistakeholder-Umgebung in Echtzeit bereitgestellt werden können.

Es werden vier Forschungsfragen untersucht. Da die Nutzung verschiedener Informationsquellen heutzutage in vielen Anwendungsbereichen eine beliebte Wahl ist, untersucht die erste Forschungsfrage den Einfluss der Empfehlungsqualität in Bezug auf das Kombinieren von mehreren solchen Datenquellen. In einer empirischen Studie wird, zum Beispiel, festgestellt, dass Quellen, die soziale oder ortsbezogene Daten beinhalten, besonders vorteilhaft für die Bewältigung des *Cold-Start* Problems sind. Die zweite Forschungsfrage baut auf diesen Erkenntnissen auf und befasst sich mit dem Problem der anwendungsorientierten Unterstützung heterogener Informationsquellen. Der Fokus hier liegt insbesondere in der Anpassung, Skalierbarkeit und Echtzeitleistung innerhalb einer Multistakeholder-Umgebung. Daraus ergibt sich ein methodischer Beitrag zu den Prinzipien und Aspekten, die bei der Gestaltung solcher Systeme unterstützt werden müssen, sowie zwei Frameworks für Empfehlungssysteme, die die zugrunde liegende Architektur für eine Multistakeholder-Umgebung darstellen. Die dritte Forschungsfrage untersucht, wie der *trade-off* zwischen algorithmischer Genauigkeit und der Laufzeitleistung in einer Echtzeitumgebung aus-

geglichen werden kann. Die durchgeführten Experimente bezüglich Skalierbarkeit zeigen, wie man die gewünschte Empfehlungsgenauigkeit erreichen kann, während eine unveränderte Laufzeitleistung unter verschiedenen *load peaks* garantiert wird. Weiters wird gezeigt, wie man die Generierung von Empfehlungen weiter beschleunigen kann, indem man zum Beispiel den häufig genutzten *Collaborative Filtering* Algorithmus anpasst, um Nachbarn in einer *greedy* Art und Weise zu finden, oder indem man Suchmaschinentechnologie mit latenten Vektoren nutzt. Die vierte Forschungsfrage dieser Arbeit befasst sich mit dem Problem der Verbesserung von Echtzeitempfehlungen, die über typischerweise verwendete Genauigkeitsmaße hinaus geht. Das Ergebnis ist eine Onlinestudie, die zeigt, dass Offline-Experimente, die sich nur auf die Verbesserung der Genauigkeit konzentrieren, eine limitierte Perspektive auf die echte Effektivität von Empfehlungssystemen bietet. Weiters wird ein methodischer Beitrag präsentiert, der einen neuartigen Ansatz zur Nutzung von neuronalen *Autoencodern* vorstellt, um die Empfehlungsqualität für anonyme Session-Benutzer zu verbessern.

Abschließend eröffnet die in dieser Arbeit durchgeführte Forschung eine Reihe möglicher Forschungsrichtungen für zukünftige Arbeiten, die sich an Algorithmen und Systeme fokussieren, die speziell auf Echtzeit-Empfehlungen zugeschnitten sind und in einer Multistakeholder-Umgebung eingesetzt werden. Dazu gehören zum Beispiel der Umgang mit verschiedenen Daten *Biases*, die Einbeziehung von Mechanismen zum Schutz privater Daten, Online-Performance vorab zu schätzen, sowie die Bereitstellung von Echtzeit-Empfehlungen anhand mehrerer Sessions, die für bekannte Benutzer verfügbar sind.

Acknowledgements

This dissertation would not be possible without a number of people who provided me their support and advice throughout the whole realization process of this thesis.

First and foremost, I would like to thank my supervisor, the head of the Social Computing research group at the Institute of Interactive Systems and Data Science and dean of study for Computational Social Systems at Graz University of Technology - Assoc.-Prof. Dr. Elisabeth Lex. Thank you for constantly pushing me to do my best and giving me all the critical feedback, regardless of how hard it was for me to sometimes hear it. And thank you for just being there, not only to help me with my research but also for always going well beyond. I want you to know that without your tremendous support, all of this would not have been possible.

To Prof. Dr. Stefanie Lindstaedt, head of Institute for Interactive Systems and Data Science and CEO & Scientific Director of Know-Center - thank you for making it possible to work on my research and provide me with so many opportunities to apply it directly in an industry setting at the Know-Center. Prof. Dr. Christoph Trattner, my advisor during the first two years of this thesis, thank you for introducing me to the world of Recommender Systems and teaching me how to write papers and conduct scientific experiments.

To all my current and former colleagues, thank you for all the discussions and valuable feedback that helped me finish my thesis. Franjo Bratić and the whole IT department — instead of a thank you, I would rather say sorry for all the headaches I may have caused you. A special thanks goes to Dr. Dominik Kowald, Dr. Sebastian Dennerlein and Dieter Theiler which accompanied me on my journey since day 1. Guys, if nothing else, never forget Barcelona. Also, one more shoutout to Markus

Reiter-Haas, it was a pleasure doing research with you and helping you finish your Master's degree.

One huge *thank you* goes to my Zagreb crew at the Know-Center: Mario "Kum" Lovrić, Leon "Kumić" Fadljević and Tomislav "Požuri" Đuričić. All those car rides between Zagreb and Graz flew just by with you. From all people, however, I have to mention and give my thanks to Tomislav Đuričić twice. Not because you pushed me to learn how to play basketball without fouling (at least you tried). But because you, out of all people here, supported me the most — especially in times I really needed it. This now brings me to Borivoj Radmanović, Antonella Karajica and Admir Brkić — thank you for being the kind of friends I didn't want to have, but the ones I deserved. Joking, of course, you're awesome!

Last but not least, a special thanks to my whole family. Now you can finally stop asking me when I'm going to finish my PhD. Seriously though, to my grandparents, all my aunts and uncles as well as all my cousins, thank you for being there for me. Iris and Hrvoje Pindrić, I appreciated the public nagging and pushing me to get it over with — it actually did help. A special thanks goes to my dear brother Mario Lacić, my dearest sister Lucija (still) Lacić as well as soon-to-be brother-in-law Gabrijel Tutić. Your unwavering belief in me was a constant moral support. Finally, I would not only love, but I owe it deeply to thank my parents, Dubravka and Miroslav Lacić. Thank you for raising me and teaching me the right values of life. Thank you for your unconditional love and constant support. Thank you for making all of this possible and not even blinking an eye whenever I came and asked you for help. I dedicate this thesis to you. Without you believing in me, it would have not been possible otherwise.

"You can have data without information, but you cannot have information without data." (Daniel Keys Moran)

Contents

1 Introduction **6**
1.1 Structure of this thesis 7
1.2 Research Questions 8
1.3 Scientific Contributions 20

2 Related Work **22**
2.1 Recommender Algorithms 22
2.2 Neural Recommenders 23
2.3 Information Sources 24
2.4 Scalable Recommender Systems 25
2.5 Real-time Recommendations 26
2.6 Evaluating Recommendations 26

3 Publications **28**
3.1 Main Publications 28
3.2 Additional Publications 34

4 Conclusions and Outlook **122**
4.1 Results and Contribution 122
4.2 Limitations 125
4.3 Topics for Future Research 127

List of Figures

1.1 Research questions 9
1.2 Traditional recommender network 10
1.3 Heterogeneous recommender network 11
1.4 Notions of a domain 13
1.5 Multi-domain design aspects 14
1.6 System architecture of ScaR 16

List of Tables

3.1 Overview of the 6 main publications as well as the 12 additional publications made for answering all four research questions of this thesis. 29

Chapter 1

Introduction

With the advent of the Web, the majority of our activities like shopping, socializing or working are being performed online [Lewis, 2021]. During the past decade, the amount, complexity and heterogeneity of information available on the Internet has downright skyrocketed and presents a huge hurdle for the human processing capabilities. For example, the e-commerce website Amazon.com provides hundreds of millions of products [Smith and Linden, 2017] of which an average user is highly unlikely to be able to process them all. This naturally also holds for applications in other domains like Airbnb, YouTube, Spotify, or Netflix. It has become abundantly clear, particularly since the hype of the social Web and platforms such as Twitter, Facebook, and Reddit, that we now live in an age of information overload [Matthes et al., 2020, Lewis, 2021].

This led to a tremendous increase in popularity of utilizing recommender systems where nowadays, they are acknowledged as an essential feature to help users discover relevant content [McNee et al., 2006]. At Neflix for example, up to 75% of what users watch are recommendations[1] and this is estimated to save the company more than $1 billion per year [Gomez-Uribe and Hunt, 2015a]. As a consequence, current research focuses on improving the prediction task in order to deliver recommendations that are more likely to align with people's preferences. Thus, many well known methods are available, such as the ones that use the textual content of recommendable items [Balabanović and Shoham, 1997] or the historical preference of a user in form of interactions with the online system [Sarwar et al., 2001, Koren et al., 2009].

[1] `http://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers`

Most of the earlier work on recommender systems has focused on adapting existing algorithms for a specific application domain like movies at Netflix, music at Spotify or products at Amazon. But in recent years, integrating data across different domains [Im and Hars, 2007, Cantador et al., 2015] or markets [Roitero et al., 2020, Bonab et al., 2021] is starting to gain more momentum in the research community. A common approach in such settings is to simplify the prediction task and view it as a problem with only two dimension. That is, given a user profile and a target item, the task at hand is to predict if there exists a link between them and if yes, to see if that particular item should be recommended. The current challenge here lies in fully utilizing data from auxiliary sources. This especially holds when the aim is to support and provide recommendations to multiple domains at the same time.

With the arrival of the Big Data era, recommender systems must now deal with the dynamic nature of the Web, requiring them to analyze large amounts of data, handle streams of new data, and support a range of data formats (i.e., volume, velocity, and variety which define the Big Data problem [Russom et al., 2011]). In these situations, traditional recommender systems typically examine the data offline and update the underlying model at regular intervals. However, a user's choice depends on many factors, which are susceptible to change anytime. For example, users who physically visit a shopping mall can trigger frequent indoor location updates via a smartphone application while moving through the mall [Lacic et al., 2015b]. An offline model update approach that, for example, lasts hours may miss the user's recent location context and fail to give the appropriate recommendations to meet the user's real-time need. As such, being able to consider a user's real-time interests is gaining traction and is not only in high demand within the research community but also with recommender systems' practitioners [Huang et al., 2015, Jannach et al., 2017, Al-Ghossein et al., 2021, Kersbergen and Schelter, 2021]. [Lacic, 2016]

Thus, the aim of this thesis is to show how real-time recommendations can be provided within a multi-domain setting.

1.1 Structure of this thesis

This thesis is structured as follows: the current chapter continues with Section 1.2, which defines the four research questions, the overall problem, approach, and findings

related to each of the questions. The accompanied Section 1.3 then summarizes the scientific contributions of this dissertation's research. Chapter 2 presents an overview of the state-of-the-art that is related to this dissertation. Chapter 3 lists the publications related to this thesis and presents their corresponding contributions. This chapter is split in two parts, the main contributions that are also included in this thesis and additional publications, which are further strengthening the outcome of this thesis but are only referenced. Finally, Chapter 4 concludes this thesis and points to relevant research directions for future work.

1.2 Research Questions

This section describes the four research questions that are investigated in this thesis. As visualized in Figure 1.1, this thesis first investigates how combining data sources impacts the recommendation quality (RQ1). Second, the problem of tailoring recommendations for a multi-domain environment is tackled (RQ2). Third, the real-time performance of recommender algorithms is analyzed when applied in a multi-domain environment (RQ3). Finally, this thesis studies improvements on the quality of recommendations which go beyond the notion of standard accuracy measures (RQ4). Summing up, in this thesis, the following research questions are tackled:

RQ1: How does combining different data sources and recommender approaches impact the robustness of recommendations?

Problem. Personalized recommender systems commonly calculate their recommendations using only one information source, namely interaction data. This can be, in case of an e-commerce system for example, implicit user feedback like already clicked or purchased products [Guo et al., 2011, Zhang and Pennacchiotti, 2013, Trattner et al., 2014]. As depicted in Figure 1.2, solving the recommendation problem in such a traditional setting would mean to predict connections in a bipartite graph. Although there are many examples that perform reasonably good when handling the prediction task in such way, nowadays, many platforms have the opportunity to collect additional information about their users. As seen in Figure 1.3, where an example of a social marketplace is depicted, this can come directly from different data

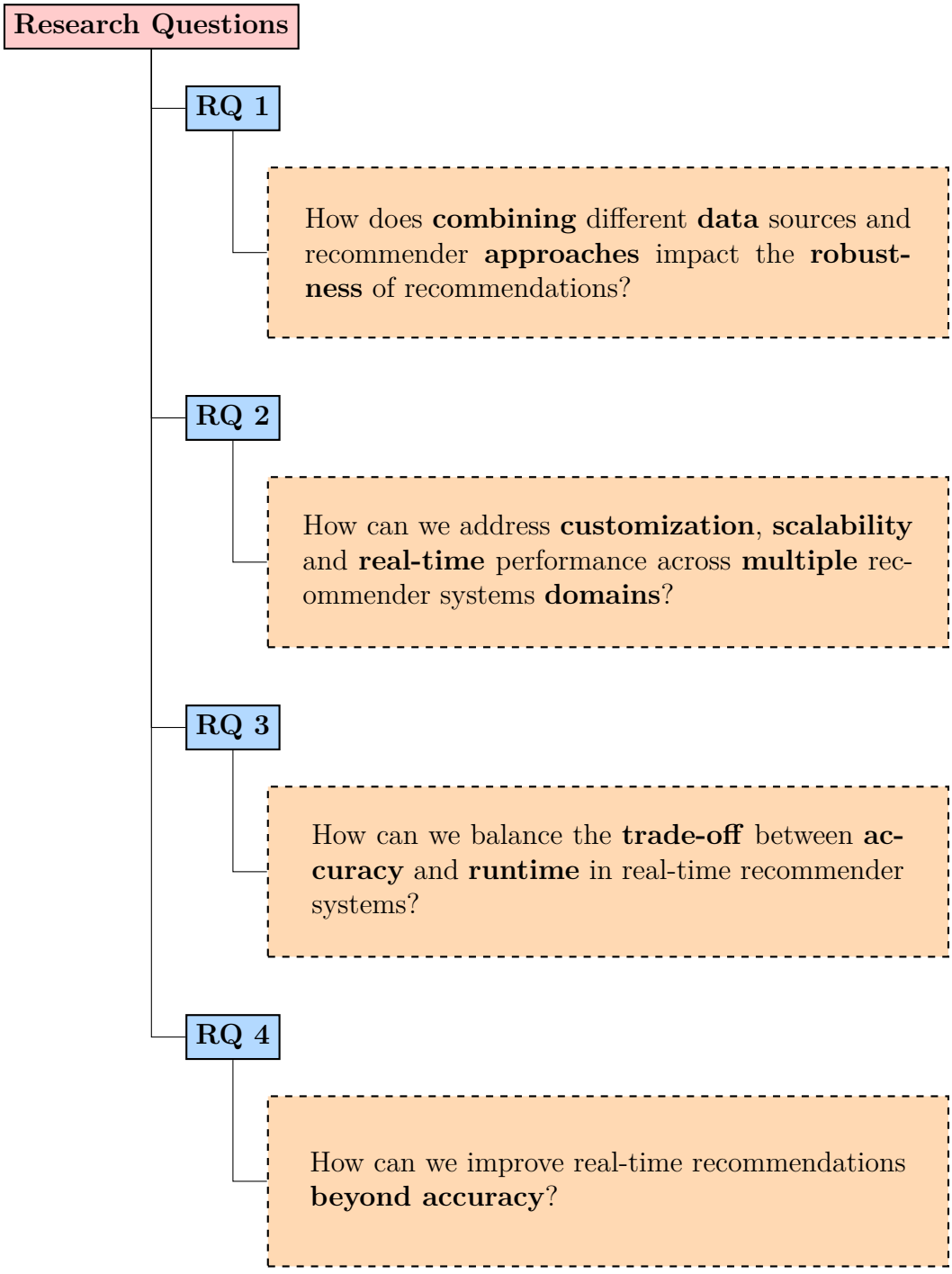


Figure 1.1: The four research questions addressed in this thesis.

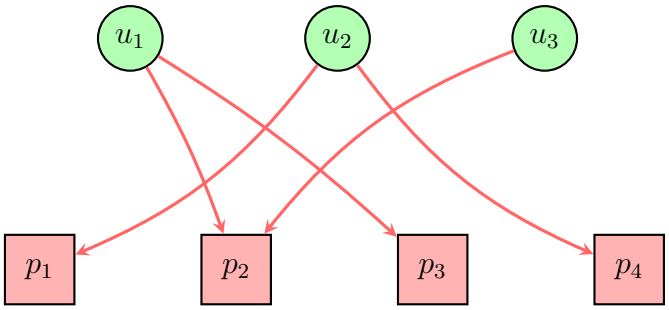


Figure 1.2: Traditional recommender systems consider only a bipartite network which represents user interactions. For instance, a user (green) in an e-commerce online system would buy products (red) and this data would be used to recommend additional products to purchase.

sources like social and location-based data (e.g., by supporting a social login mechanism [Kontaxis et al., 2012]) or be inferred from the existing data (e.g., creating a location network out of check-in data). Even though previous research has shown that additional information can be useful in the wide field of recommender systems (e.g., [Feng and Wang, 2012, Liu et al., 2017b, Berndsen et al., 2020, Patro et al., 2020]), it remains an open problem to understand the impact on the robustness of recommendations when combining different information sources.

Approach. The first research question of this thesis aims to understand how different information sources can help in learning a user's preference and consequently create more robust recommendations. To that end, in [Lacic et al., 2015a] we studied the efficacy of utilizing social, location and marketplace data features to recommend products and categories to users in an online social marketplace. In addition to that, we investigated on how to efficiently combine data sources in order to further increase the recommendation quality. This study was extended in [Lacic et al., 2014c] and [Lacic et al., 2015b], where we looked at the impact of additional data sources for users that are new to the system. In [Duricic et al., 2018] we explored how explicit trust connections between users can be leveraged to provide recommendations when no other data is available about a user. In [Lacic et al., 2014b], we studied social tags and how to use them by considering their temporal patterns. Finally, in [Reiter-Haas et al., 2017] we showed how domains which do not rely on additional data sources can be enriched with data that can serve as an information source for the recommender.

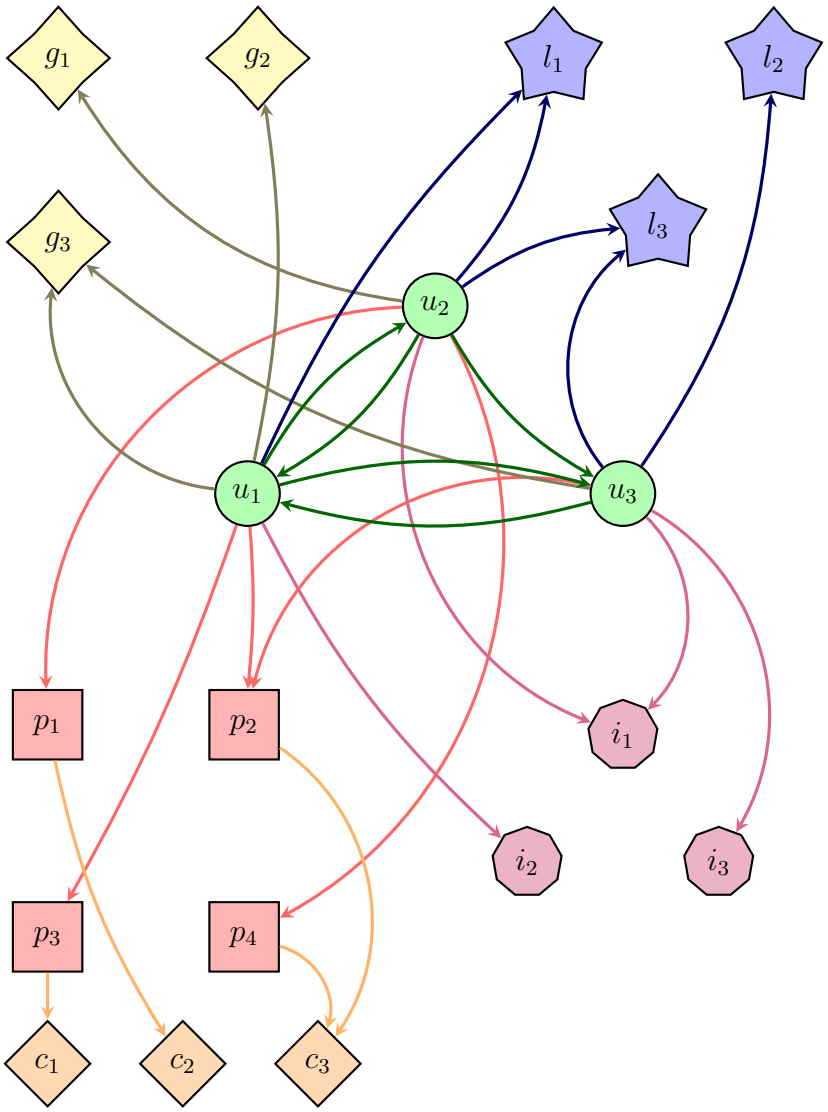


Figure 1.3: Heterogeneous network representing an online social marketplace that contains information commonly found in popular online systems like Facebook or Amazon. A common action for users is to purchase or sell a product p_i (red) that belong to a specific category c_i (orange). Users also frequently belong to a social group g_i (yellow) or have a specific interest i_i (purple) denoted in their profile. Mostly however, a user u_i (green) interacts with other users via social interactions such as likes or comments. In addition to that, users may notify their community via a check-in when they arrive at a specific location of their interest l_i (blue).

Findings and Contributions. We first introduced content and network-based similarity features that can be applied for different data sources. By focusing on the popular non-probabilistic user-based Collaborative Filtering [Schafer et al., 2007] algorithm, we further showed how the use of such features in the recommendation process helps to improve the recommendation performance. Here, for example, utilizing the network structure derived from social interactions (e.g., likes) was shown to be particularly beneficial in improving the recommendation quality. Furthermore, our experiments proved our initial assumption that combining user similarity features of the different data sources in a hybrid manner can lead to more robust recommendations with respect to accuracy, diversity, and user coverage. [Lacic et al., 2015a]

We also demonstrated how the recommendation quality could improve if an additional data source is provided (e.g., [Reiter-Haas et al., 2017, Duricic et al., 2018]). In case of social data, we showed that the overall quality greatly depends on the number of social profiles available in the system. We also extended this with a cold-start simulation of a new system (e.g., a similar case like it was with the beginning of Spotify) in which we assumed that all new users provide social data. In this case, the delivered prediction quality can already be significantly increased in the early stages of the new system. [Lacic et al., 2014c]

This was further confirmed in a similar experiment where we tackled extreme cold-start users, but with location data. We showed that when location data is available (e.g., generated from indoor positioning systems in shopping malls), we can greatly improve the first recommendation to completely new users in the system by either using the raw location traces or inferring a location network out of them. [Lacic et al., 2015b]

Finally, we also considered social tagging systems, which are another popular data source. We contributed with a novel algorithm called Collaborative Item Ranking Using Tag and Time Information (CIRTT) and showed how the recommendation performance can be improved by ranking candidate item sets using the information of frequency and recency of tag use. [Lacic et al., 2014b]

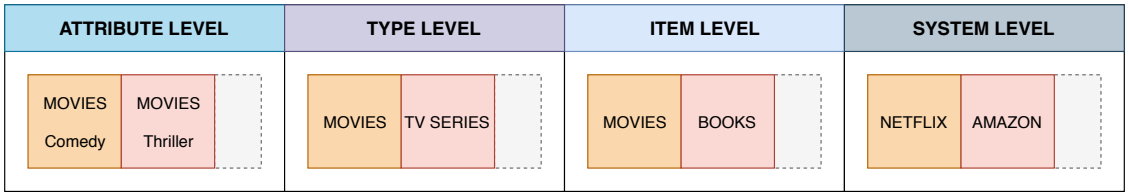


Figure 1.4: Notions of a domain according to [Cantador et al., 2015]. The attribute level contains the same type of items (e.g., movies) but has different values on certain attributes (e.g., genre of the movie). The type level notion has similar types of items (e.g., movies and TV series) which share some of their attributes. Individual domains on the item level have different types of items (e.g., movies and books). On a system level, items (e.g., movies or books) belong to distinct systems, which are considered as different domains (e.g., Netflix and Amazon). [Cantador et al., 2015, Lacic et al., 2017]

RQ2: How can we address customization, scalability and real-time performance across multiple recommender systems domains?

Problem. Most of the literature about recommender systems focuses on developing novel approaches that improve the accuracy for items that belong to a single domain (e.g., movies, music, news, etc.). But focusing on multiple domains (e.g., [Roitero et al., 2020, Bonab et al., 2021]) is gaining more traction in the research community. The work of [Cantador et al., 2015] was the first attempt to define the concept of a domain in the setting of recommender systems. As seen in Figure 1.4, the authors distinguish between four different domain notations. First, there is the attribute level, where items are of the same type (e.g., movie genres). Second, the type level, where items are of similar types but do not necessarily share all attributes (e.g., movies and TV series). At the item level, items are not of the same type and differ in most or all attributes (e.g., movies and books). Finally, at the system level, items and users belong to completely different systems (e.g., Netflix and Amazon). [Lacic et al., 2017]

Most work that adopt this notion focus solely on utilizing the data between domains in order to improve the recommendation performance [Gao et al., 2013, Loni et al., 2014, Sahebi and Walker, 2014, Elkahky et al., 2015, Sahebi and Brusilovsky, 2015, Bonab et al., 2021]. There is however still a lack of work on how to actually simultaneously support recommendations in a multi-domain environment.

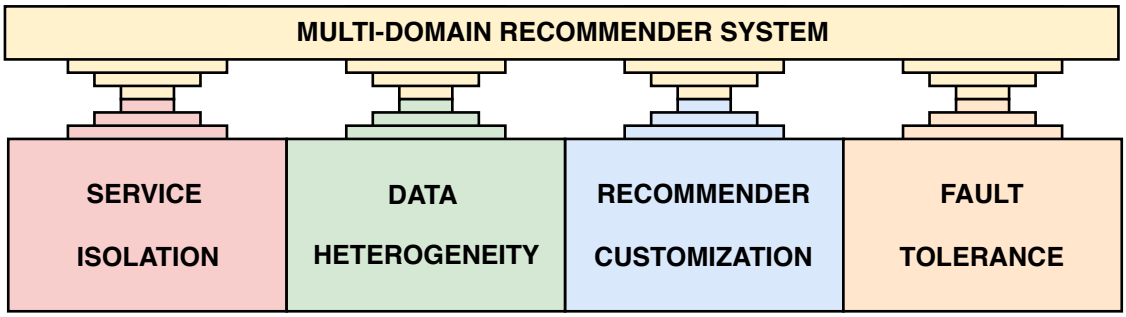


Figure 1.5: Building a recommender system for a multi-domain environment needs to address the aspect of (i) providing service isolation, (ii) supporting data heterogeneity, (iii) allowing recommender customization, and (iv) ensuring fault tolerance.

Approach. The aim of the second research question is to build upon the work of [Cantador et al., 2015] and extend the scope of multi-domain recommender systems. In [Lacic et al., 2017] we have categorized and introduced different design aspects that should be addressed when providing recommendations. To show the applicability of these design aspects, we first presented a framework for providing real-time recommendations for online marketplaces [Lacic et al., 2014c] and then extended it to account for scalability and a multi-domain environment [Lacic et al., 2015c]. Finally, in [Traub et al., 2015, Lacic et al., 2016, Kowald et al., 2018, Lacic et al., 2018a, Lacic et al., 2018b] we showed how such an approach can be utilized to customize a recommender system across multiple domains.

Findings and Contributions. As depicted in Figure 1.5, we categorized a multi-domain recommender system into four different aspects. First, sharing hardware resources while ensuring performance isolation and fault tolerance is required to support different requirements in terms of recommendation request load[2]. Such functionality helps, for instance, in handling unanticipated load peaks a domain may occasionally encounter. Second, a multi-domain recommender system inherently needs to be able to handle a diverse set of data sources and simultaneously support an easy integration of new data types (e.g., by altering the underlying schema). Such a need can be depicted by Listing 1.1, where two domains have a requirement of storing user-item interactions. But as differentiated in [Adomavicius and Tuzhilin,

[2]The number of simultaneous recommendation requests that the system needs to process in a timely manner.

```
{
  "item": "5a35c4d3−a5dc−36d4−6a8b−f543da297a7c",
  "users_listened": [5506, 68107, 1000002]
  "users_listened_count": 3
  "domain": "LastFM"
},
{
  "user": "56861",
  "item": 3473,
  "rating": 2.0
  "domain": "MovieLens"
}
```

Listing 1.1: Example of different user-item interactions that need to be stored and utilized by recommender algorithms in an multi-domain envorinment. The recommender system in this case needs to support the implicit listening interactions from the music domain (i.e., LastFM), as well as explicit movie ratings found in the movie domain (i.e., MovieLens).

2008], a recommender algorithm employed by one domain may consider different types of user interactions than the one used in the other domain. That is, the underlying system would need to support every kind of interaction (e.g., implicit, explicit or inferred ones) that could form such a two-dimensional view. Finally, a multi-domain recommender system should support the customization of domain-specific parameters related to the employed algorithm and unambiguously know which source of information should be used to calculate the recommendations. [Lacic et al., 2017]

To show that these design principles can be applied in real systems, we first contributed with SocRecM, a Java-based recommender framework that is aimed to build a scalable social recommender engine for online marketplaces. With SocRecM we showed how search engine technology can be leveraged to build a recommender system that supports large quantities of diverse data structures. [Lacic et al., 2014c]

We then improved on the runtime performance in a multi-domain setting and contributed with ScaR (i.e., Scalable Recommendations-as-a-service), an extension that is aimed for a multi-domain setting. To show its applicability as depicted in Figure 1.6, we customized and presented ScaR for the domains of Tourism, Music, Movies, Venues, Social Care, E-Commerce, Technology Enhanced Learning and Job

Matchmaking. Furthermore, we found that the domain-specific customization of a recommender algorithm is of particular importance, as it also fosters the reproducibility [Ekstrand et al., 2011] of experimental evaluations conducted within an application domain. [Lacic et al., 2015c]

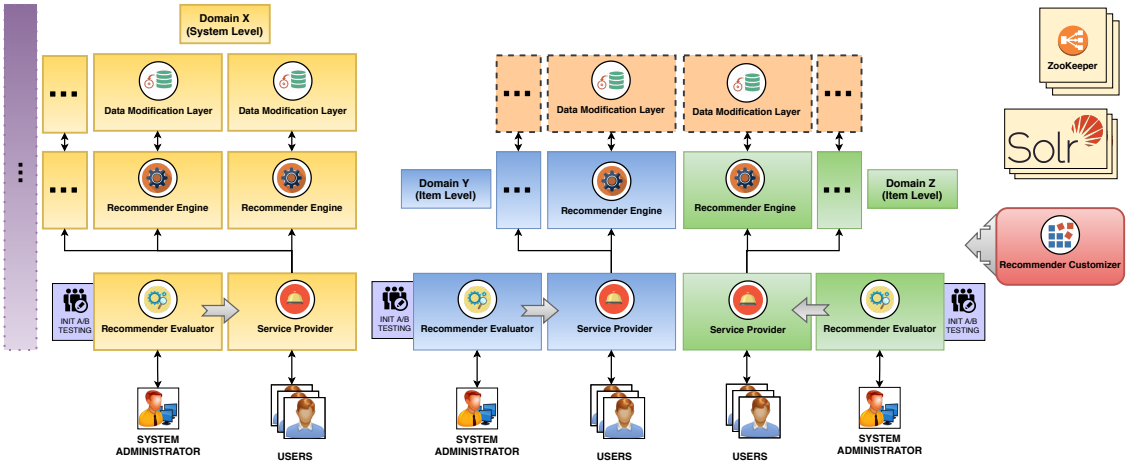


Figure 1.6: Proposed architecture for a multi-domain recommender system. Each component is a microservice (i.e., a standalone HTTP server) that is aware of its communicating partners' location (i.e., the URL). If a multi-domain scenario at the system level is not required, the same form of data storage can be used between individual domains. The corresponding customization through recommender profiles provides the domain-specific algorithm configuration. [Lacic et al., 2017]

RQ3: How can we balance the trade-off between accuracy and runtime in real-time recommender systems?

Problem. In the past decade, most research on recommender systems has focused on producing novel recommendation algorithms [Shani and Gunawardana, 2011] and improving the prediction accuracy [Rana and Jain, 2015]. Traditional recommender systems typically analyze data offline and generate an algorithmic model at regular time intervals in order to be used in a dynamic environment such as the Web. However, a user's choice depends on a variety of factors that are subject to change at any time. One example would be the news domain, where a majority of the user base are short-lived session users that interact with only few news articles. An

offline model training strategy which is updated after several hours or even days may miss the current context of the user's real-time demand. Hence, the interest of the research community has shifted into balancing the open problem of showing accurate recommendations while being able to capture and consider a user's real-time interest [Chandramouli et al., 2011, Huang et al., 2015, Rana and Jain, 2015].

Approach. The focus of the third research question is to investigate the impact on the accuracy while providing recommendations in real-time. This led us to first investigate if and how search engine technology can be adapted in order that recommender algorithms exhibit a real-time performance [Lacic et al., 2014a]. In [Lacic et al., 2015c, Lacic, 2016, Lacic, 2017], we further looked at how an increasing recommendation request load can impact the runtime performance of a given algorithm. In this study, we additionally explored how scaling the system can help to reduce the runtime of particular algorithms and the general response time of the overall recommendation system. In [Lacic et al., 2018a] we investigated how to speed up the nearest neighbor approach as it is one of the most explored and utilized techniques for personalized systems. Finally, in [Lacic et al., 2019b], we conducted a user study within the job domain to uncover the applicability of user embeddings in a real online setting while maintaining a mean runtime performance that is below 200 ms.

Findings and Contributions. We first demonstrated that by leveraging search engine technology (e.g., Apache Solr), we can generate recommendations under real-time constraints as well as handle data streams and frequent updates [Lacic et al., 2014a]. We further showed that under an exponentially growing workload (i.e., a load peak that is triggered by an increasing number of incoming recommendation requests), the recommendation system will continue to increase the average runtime performance of individual algorithms even though there is a maximal theoretical limit of recommendation requests that can be handled by the underlying hardware and software configuration. By scaling the system in a horizontal manner, it is however possible to increase this theoretical limit and therefore minimize the impact on the runtime performance of individual algorithms. By making such scalability experiments, it is possible to find out which algorithms should be used or combined (e.g., in case of hybrid approaches) in order to guarantee a real-time runtime per-

formance while maintaining the desired recommendation accuracy. [Lacic et al., 2015c, Lacic, 2016, Lacic, 2017]

To further improve the balance between accuracy and runtime we explored the Collaborative Filtering algorithm [Resnick et al., 1994, Sarwar et al., 2001] which is one of the most explored and utilized approaches for providing personalization. For this, we integrated a user pre-filtering step to build a smaller set of candidate neighbors in a greedy manner. That is, we focused on neighboring users which have a higher amount of overlapping items. Our experiments showed that in this way, it is possible to greatly speed up the runtime performance and in some cases not even sacrifice the overall accuracy. [Lacic et al., 2018a]

Finally, we also considered the utilization of latent item embeddings, as these have recently become one of the most popular methods to use in the recommender system's research community. For this, we conducted an online user study in the job domain for the scenarios of recommending similar jobs and personalizing the homepage. We showed how to utilize embeddings in order to improve Click-Through-Rate of the recommender while having a below 200 ms average runtime performance. Moreover, we showed that the recommendation scenario (i.e., context where the user sees the recommendations) in fact impacts the final online performance of a utilized recommendation algorithm. For instance, when recommending similar jobs, users expect to see something that is relevant to their recent interaction history. When compared to classical approaches, utilizing embeddings that focus on recent interactions led to both, an improved Click-Through-Rate and runtime performance. [Lacic et al., 2019b]

RQ4: How can we improve real-time recommendations beyond accuracy?

Problem. The focus of research on recommender systems has been in many cases only on assessing and improving the prediction accuracy [Jäschke et al., 2007, Shi et al., 2010, Liang et al., 2018]. But optimizing only on accuracy and not the human need for variety and discovery may easily lead to a lesser user experience, as this easily results in "*boring and ineffective recommendations*" [Zhang et al., 2012]. The research community has therefore acknowledged that in an online, real-time setting, factors other than accuracy may have a significant effect on the performance

of a recommender system [McNee et al., 2006]. To assess the true utility of a recommender algorithm, one would usually need to conduct a user study or set up an online A/B testing experiment. The former approach requires the involvement of active participants within a pre-defined time frame [Beel et al., 2013]. The latter requires access to a fully functional system with existing users. Moreover, to conduct A/B tests one would need to deal with other real-time constraints [Eksombatchai et al., 2018], like having a response time of recommendation requests which do not exceed a certain amount of milliseconds. As such, there is a growing trend to go beyond accuracy. This essentially means to investigate offline metrics that provide more in-depth insights into the quality and final utility of a recommender approach (e.g., [Ge et al., 2010, Belém et al., 2013]).

Approach. For the fourth research question, we first explored how the recently popularized utilization of item embeddings compares to standard recommendation algorithms on measures which go beyond recommendation accuracy [Lacic et al., 2018b]. For this, we looked at different strategies on how to combine embeddings in order to balance the trade-off between accuracy, diversity and novelty. We continued with this using an online study under real-time constraints in [Lacic et al., 2019b] and investigated the impact of the display context (i.e., where the utilized approach was in the end shown). In [Lacic et al., 2019a] we further investigated how measuring the semantic similarity between the list of recommended and expected items can help to better understand if a "hit" (i.e., a relevant item) has been generated. Finally, in [Lacic et al., 2020] we focused on a special case in the domain of real-time recommender systems, namely, recommending for anonymous session users. As research on session-based recommenders with respect to beyond accuracy measures is rather scarce, we investigated the performance of state-of-the-art session-based approaches as well as proposed how to extract and utilize latent embeddings of a particular session.

Findings and Contributions. We first explored how item embeddings impact recommendation accuracy in addition to diversity and novelty. For this, we investigated different strategies on how to combine embeddings that are derived from the user's history. We showed that the trade-off between accuracy and diversity can be balanced by using human memory theory and proposed an approach that models

the frequency and recency of user's past interactions. [Lacic et al., 2018b]

Using these insights, we further conducted a variety of A/B experiments and contributed to the sparse line of research on evaluating the impact of latent embeddings under real-time constraints. Here we focused on two popular recommendation scenarios: (i) to provide recommendations when a user is currently at the details page of a particular item and (ii) to personalize the items which are shown on the homepage. In the first scenario, we found that using embeddings that are based on more recent interactions tends to improve the online performance. In contrast, for the scenario of personalizing the homepage, we found that the combination of embeddings which is based on the frequency and recency of past user interactions exhibits the best online performance. [Lacic et al., 2019b]

We then focused on showing how measuring only accuracy may not be enough when evaluating the quality of recommender algorithms. We contributed with an alternative measure that measures the semantic relationship of recommended items and showed that even if the expected item was not found, the underlying algorithm may still have recommended an item that is relevant to the user. [Lacic et al., 2019a]

Finally, we addressed the problem of anonymous user sessions and evaluated state-of-the-art approaches with respect to beyond-accuracy metrics (i.e., novelty, serendipity, and coverage). Here, we additionally contributed with a novel approach that utilizes autoencoders in order to extract latent embeddings of a particular session. When compared to state-of-the-art baselines, we discovered that when we learn session embeddings based on interaction and content data and use them in a k-nearest neighbor manner, we can achieve the best results with respect to the beyond-accuracy measures, as well as competitive results in terms of accuracy. [Lacic et al., 2020]

1.3 Scientific Contributions

This section outlines the scientific contributions of this dissertation's research. With respect to the research questions of this thesis, the five main contributions are as follows:

1. It is shown that utilizing additional information sources can lead to more

robust recommenders with respect to accuracy, diversity, and user coverage (Research Question 1). Specifically, by combining several recommender approaches that rely on heterogeneous data sources, it is possible to increase the recommendation accuracy. Also, the use of additional data helps mitigate the cold-start problem, i.e., when a new user comes to the system.

2. As the amount and heterogeneity of available data sources heavily differ between application domains, a scalable and customizable architecture is proposed. Such an architecture is shown to be easily adapted for different recommendation scenarios and to be suited for providing recommendations in an environment where multiple domains need to supported at the same time (Research Question 2).

3. It is demonstrated how recommendations in a multi-domain setting can be provided in real-time (Research Question 3). Furthermore, it is shown that the trade-off between recommendation accuracy and runtime needs to be taken into account when considering to apply a specific recommendation strategy.

4. By applying novel methods based on neural autoencoders, it is shown that real-time recommendation approaches can be improved beyond the most commonly measured recommendation accuracy (Research Question 4). It is further shown that this can also be applied to a special scenario of recommending for anonymized user-sessions in real-time, where the recommendation problem is even harder because the available data is much sparser than when the user is known to the system.

5. Finally, two open-source Java-based frameworks are presented. With SocRecM, this thesis first shows how real-time recommendations can be provided by utilizing heterogeneous data sources in the e-commerce domain. With ScaR, this thesis improves on the real-time performance achieved with SocRecM with respect to an increasing number of incoming recommendation requests and extends the framework with the proposed notations of a multi-domain recommender system.

These contributions are presented in the 18 publications, which are listed in more detail in the following Chapter 3.

Chapter 2

Related Work

This chapter presents the state-of-the-art research related to the four research questions presented in the previous Chapter 1 of this thesis. This is done by first presenting an overview over algorithms that are usually utilized for solving the recommendation problem. Following that is a discussion on recent trends with respect to neural recommendation approaches. The inclusion of auxiliary data is addressed afterwards, as user-item interactions are not the only source of information that can be used to generate recommendations. With respect to running a recommender system in an online setting, work that is related to topics of scalability and real-time performance are then presented. Finally, this chapter ends with an overview of recent trends and ongoing problems the research community has in regard to evaluating recommender systems.

2.1 Recommender Algorithms

Recommender systems typically learn a model to predict the preference for each user-item pair. Most commonly, this is done in a nearest-neighbor fashion (i.e., user-based [Resnick et al., 1994] or item-based [Sarwar et al., 2001]) or by finding similar items in a content-based manner [Balabanović and Shoham, 1997]. In this respect, personalized recommendations using Matrix Factorization [Koren et al., 2009, Ma et al., 2011] dominate the literature. The most popular use case here was to formulate the problem as a rating prediction task, where the model would attempt to minimize the difference between the predicted and the original rating

value. One popular approach in this setting is, for example, Probabilistic Matrix Factorization (PMF) by Mnih and Salakhutdinov [Mnih and Salakhutdinov, 2008].

The research community has, however, recognized that rating prediction does not adequately depict a real-world recommendation scenario [Steck, 2013]. Thus, the problem is rather formulated as a ranking task where, out of all possible items, only a small number should be picked. In this scenario, approaches like Weighted Matrix Factorization (WMF) [Hu et al., 2008], Bayesian Personalized Ranking (BPR) [Rendle et al., 2009] or Sparse Linear Methods (SLIM) [Ning and Karypis, 2011], which are based on implicit feedback (e.g., a binary signal in the form of 1 or 0 if a user has clicked an item) are a popular choice. One caveat when adapting Matrix Factorization for a real-world scenario however, is that they need to be retrained when the data changes. Namely, this can happen to be a time-consuming task, especially in case of frequent data updates. Furthermore, empirical studies showed that the computational cost should not be neglected, as a large number of latent factors are needed so that Matrix Factorization can deal with sparse data. [Salakhutdinov and Mnih, 2008]

2.2 Neural Recommenders

Research on algorithmic advances in the recommender systems' community has changed in the last few years where the trend has moved more towards utilizing deep learning techniques [Dacrema et al., 2019]. The focus of earlier work was on explicit feedback and solving the rating prediction problem with neural networks. For that, Restricted Boltzmann Machines were a popular choice to serve as a basis for Collaborative Filtering [Georgiev and Nakov, 2013, Zheng et al., 2016]. With respect to formulating the problem as a ranking task, two early methods are Collaborative Denoising Autoencoder (CDAE) [Wu et al., 2016] and Collaborative Deep Learning (CDL) [Wang et al., 2015]. Both use denoising autoencoders where the former learns a latent user representation by leveraging only the user history while the latter jointly represents the collaborative data with content information. Another related approach is Neural Collaborative Filtering (NCF) [He et al., 2017] where the authors explore applying non-linear interactions between the latent factors of a user and item instead of using the dot product. One problem that such methods suffer

from is that the amount of model parameters grows linearly with the number of users and items. As this can make the model much more prone to overfitting, Liang et al. [Liang et al., 2018] proposed to use the generative variational autoencoder with multinomial likelihood for collaborative filtering (Multi-VAE). Other recent work like the one from Hidasi et al. [Hidasi et al., 2015, Hidasi and Karatzoglou, 2018] propose a recurrent neural network (RNN) based approach to model variable-length sequential data. They showed that RNNs (e.g., Gated Recurrent Units) can be adapted for this task. Others build on this idea and extend it by capturing additional information like context [Twardowski, 2016] or attention [Li et al., 2017].

In summary, recent advances in neural recommenders have been heavily influenced and inspired by natural language processing (NLP) research over the last 8 years [de Souza Pereira Moreira et al., 2021]. A good example for this is the introduction of transformer architectures in 2017 [Vaswani et al., 2017] which are the basis for recent transformer-based recommender systems like SASRec [Kang and McAuley, 2018], AttRec [Zhang et al., 2019], BERT4Rec [Sun et al., 2019] and Transformers4Rec [de Souza Pereira Moreira et al., 2021].

2.3 Information Sources

Traditional recommender systems often consider only user-item interactions and do not consider any additional contextual information like social connections, visited locations, the user's age, etc. In most cases, the available interaction data is sparse in nature, with having, for example, less than 5% of all possible user-item interactions [Truong et al., 2021]. In this respect, Adomavicius et al. [Adomavicius and Tuzhilin, 2008] extended the traditional user-item paradigm by representing the recommendation context using information that is obtained (1) explicitly, by direct user input, (2) implicitly, using methods to capture the information from the environment, or (3) inferred, by analyzing interactions and building implicit connections. For example, Jamali et al. [Jamali and Ester, 2010] tackle the rating prediction problem by utilizing a Matrix Factorization model that includes existing social relations. In [Ma et al., 2011], the authors introduce two social regularization methods to be applied for Matrix Factorization in order to improve both the Mean Absolute Error (MAE) and Root-Mean-Square Error (RMSE) when social information is available. The

authors of [Diaz-Aviles et al., 2012] improve Matrix Factorization for recommending topics in real-time by using Twitter streams. On the other hand, in [Yao et al., 2015] the authors rely on inferred data by adapting BPR to integrate topics that are generated using the popular LDA approach [Blei et al., 2003].

Overall, the community has acknowledged the importance of other types of information sources, ranging from social and location data to even measuring physiological signals or personality traits, in a variety of recommendation tasks [Guo et al., 2011, Bischoff, 2012, Elahi et al., 2013, Feng and Wang, 2012, Patro et al., 2020, Berndsen et al., 2020]. In recent years for example, simultaneously with the advances in deep learning, even visual features extracted from images have been shown as a strong signal in learning the fashion preference [Kang et al., 2017] and style [Liu et al., 2017b] of a user.

2.4 Scalable Recommender Systems

Modern recommendation systems tackle a multitude of operating challenges on a daily basis. Especially with a large user and item base, it becomes challenging to handle the increasing complexity of models (e.g., deep neural networks) within an easily scalable system [Saberian and Basilico, 2021]. Most of the existing work adapt their infrastructure and use offline batch processing (e.g., with frameworks such as Spark [Zaharia et al., 2010] or GraphLab [Low et al., 2012]) with which they generate and cache candidate recommendations before they need to be served to the user. This is the case for many stakeholders from industry like Netflix [Amatriain, 2013], Microsoft [Ronen et al., 2013], Nike [Essinger et al., 2021], among many others [Chan et al., 2013, Walunj and Sadafale, 2013, Dai et al., 2014]. Another recommender infrastructure is RecDB [Sarwat et al., 2013] where the authors have taken a database-driven approach. In this case, the system is integrated in a PostgreSQL DBMS which stores the generated recommendation scores within a set of views which can be queried using SQL statements.

In summary, to overcome the challenges of scalability like YouTube [Davidson et al., 2010], Twitter [Gupta et al., 2013] or Pinterest [Liu et al., 2017a], most systems combine infrastructure improvements with efficient machine learning algorithms [Garcin et al., 2014, Freno, 2017, Eksombatchai et al., 2018].

2.5 Real-time Recommendations

When applied in an online setting, one important aspect of recommender systems is that they need to provide recommendations in real-time while processing new streams of data incrementally. This is for instance the case when modeling a user's sequence of performed actions, in order to catch the shift in interest that can happen at any point in time [Shani et al., 2005, Smirnova and Vasile, 2017]. To capture the change in user feedback, frameworks for processing interactive data like Apache Tez [Saha et al., 2015] can be used to build high-performance batch processing applications.

In this regard, the authors of [Huang et al., 2015] present TencentRec, a system to provide real-time recommendations based on Item-Based Collaborative Filtering that tackles the conflict that exists when supporting "*real-time, accurate and big (data) challenges*". Similarly, Chandramouli et al. [Chandramouli et al., 2011] base their Collaborative Filtering approach on a stream processing system that utilizes explicit rating data. Gupta et al. [Gupta et al., 2014] propose a real-time twitter recommendation system using the temporally-correlated actions of each user's followers. They report a measured median latency of 7 seconds from the edge creation event to the delivery of the recommendation due to event propagation delays. Other related work [Das et al., 2007, Rendle and Schmidt-Thieme, 2008, Diaz-Aviles et al., 2012, Chen et al., 2013] work neglect the temporal ordering of information and propose a fast procedure to generate recommendations by focusing on scalability and easy parallelization. For example, to incorporate incremental updates, [Agarwal et al., 2010] propose to learn item-specific factors through online regression, which leads to a fast online bi-linear factor model.

2.6 Evaluating Recommendations

An offline experimental setup is to date the most popular approach to quantify the performance of a recommender system [Jannach et al., 2012]. Even though there is not one exact way how to measure the performance of a recommender approach, much work in the research community has based their algorithmic comparisons on accuracy related methods like Root Mean Squared Error or normalized Discounted Cumulative Gain [Herlocker et al., 2004, Gunawardana and Shani, 2009, Parra and

Sahebi, 2013]. Recent work, which focused on setting up online user studies, indicate that an improvement in the offline accuracy performance does not directly translate to the online setting with real users under real-time constraints [Gomez-Uribe and Hunt, 2015b, Maksai et al., 2015]. For example, there are even large differences in how the same recommender algorithm performs in only slightly different application scenarios [Beel et al., 2016]. Nevertheless, reports on online studies like from [Lu et al., 2020] are rather less common as they are hard to set up. Thus, since the argumentation from [McNee et al., 2006] that *"being accurate is not enough"*, the research direction is nowadays driven toward an evaluation of recommender systems that extends well beyond providing accurate predictions. For instance, this has led to investigating different trade-offs and measures like diversity, coverage or bias, just to name a few [Ge et al., 2010, Nilashi et al., 2016, Han and Yamana, 2017, Ludewig and Jannach, 2018].

Finally, related work has shown that reproducibility of research results is a challenging issue [Dacrema et al., 2019]. Hence, toolkits such as, e.g., LensKit [Ekstrand et al., 2011] or Rival [Said and Bellogín, 2014] have been introduced that drive the research community into that direction.

Chapter 3

Publications

This chapter presents the 18 publications and the corresponding contributions for this cumulative thesis. More specifically, the contributions of this thesis are divided into two parts. Firstly, the 6 main publications for each of the research questions are listed as well as included in this chapter. Secondly, this thesis also lists the 12 additional publications which are referenced in order to additionally strengthen the outcomes and findings of this thesis. A short overview of these publications, as well as their contribution to the respective research questions can be seen in Table 3.

3.1 Main Publications

This section describes the 6 main contributions of this thesis which have already been published. In addition to being listed in Table 3, each main publication is accompanied by a short description as well as a statement of the respective contribution. The printed versions of the listed work is in addition added at the end of this chapter.

MP1

Lacic, E., Kowald, D., Eberhard, L., Trattner, C., Parra, D., and Marinho, L. B. (2015). Utilizing online social network and location-based data to recommend products and categories in online marketplaces. *In Mining, Modeling, and Recommending'Things' in Social Media (pp. 96-115). Springer.*

Notation	Reference	Research Questions
Main Publications		
MP1	[Lacic et al., 2015a]	RQ1
MP2	[Lacic et al., 2017]	RQ2
MP3	[Lacic et al., 2014a]	RQ3
MP4	[Lacic, 2016]	RQ2, RQ3
MP5	[Lacic et al., 2019b]	RQ3, RQ4
MP6	[Lacic et al., 2020]	RQ4
Additional Publications		
AP1	[Lacic et al., 2015b]	RQ1
AP2	[Duricic et al., 2018]	RQ1
AP3	[Lacic et al., 2014a]	RQ1
AP4	[Reiter-Haas et al., 2017]	RQ1
AP5	[Lacic et al., 2014c]	RQ1, RQ2
AP6	[Traub et al., 2015]	RQ2, RQ3
AP7	[Kowald et al., 2018]	RQ2, RQ3
AP8	[Lacic et al., 2016]	RQ2
AP9	[Lacic et al., 2015c]	RQ2, RQ3
AP10	[Lacic et al., 2018a]	RQ3
AP11	[Lacic et al., 2018b]	RQ4
AP12	[Lacic et al., 2019a]	RQ4

Table 3.1: Overview of the 6 main publications as well as the 12 additional publications made for answering all four research questions of this thesis.

About MP1. This work presents the impact of features that are obtained from marketplace, social and location-based data, when applied to the task of producing product and category recommendations in online marketplaces [Lacic et al., 2015a]. With respect to Research Questions 1, not only is a detailed evaluation presented on the impact of different data sources, but also their combination with respect to creating a hybrid recommender approach that improves the robustness of recommendations in terms of accuracy, diversity and user coverage.

This work further lays the foundation for Research Question 2 and 3 as it becomes eminent to support the utilization of additional data sources in application domains

with different types of data as well as to answer the question on how to do that in real-time.

Contribution. The author of this thesis designed and performed the experiments, implemented all the source code (i.e., algorithms, evaluation measures, experimental pipeline, etc.) and analyzed the data. Lukas Eberhard crawled the Second Life dataset that is used in this work. Dominik Kowald and Christoph Trattner assisted in interpreting and presenting the experimental results. The author of this thesis took the lead in writing the manuscript. All authors discussed the results and contributed with writing the final manuscript.

MP2

Lacic, E., Kowald, D. and Lex, E. (2017). Tailoring Recommendations for a Multi-Domain Environment. *In Workshop on Intelligent Recommender Systems by Knowledge Transfer and Learning (RecSysKTL'17) co-located with the 11th ACM Conference on Recommender Systems (RecSys'17)*

About MP2. Most recommender systems adapt their algorithms for items that belong to one specific domain (e.g., movies, music, news, etc.). However, how to simultaneously adapt a recommender system to a diverse set of domains, each having their own requirements and data models, still continues to be an open challenge (i.e., Research Question 2).

The contribution of this paper is a unified design of a customizable and scalable recommendation system which accounts for multiple domains with diverse information sources [Lacic et al., 2017]. To do that, it addresses the design decisions that should be taken into account when building a multi-domain recommender system, as well as demonstrates how the contributed ScaR framework can be adapted for different domain experiments in an isolated manner.

Contribution. The author of this thesis implemented the ScaR recommender framework that is presented in this work as well as conducted the reported experiments. The author of this thesis also took the lead in writing the manuscript. All authors provided critical feedback and helped in writing the final manuscript.

MP3

Lacic, E., Kowald, D., Parra, D., Kahr, M., and Trattner, C. (2014). Towards a scalable social recommender engine for online marketplaces: The case of apache solr. *In Proceedings of the 23rd ACM International Conference on World Wide Web (WWW'14), pp. 817-822*

About MP3. This work presents how search engines can be utilized to build a real-time recommender system for a social marketplace. In order to address Research Question 3, [Lacic et al., 2014a] first compares the runtime performance of different recommendation approaches, and second, compares their mean response time in form of a performance load test with an increasing amount of recommendation requests with and without data updates happening simultaneously.

Thus, this work tackles the problem of providing recommendations in real-time with the Apache Solr search engine. This forms the basis for the two open-source frameworks (i.e., SocRecM and ScaR) which are a technical contribution of this thesis.

Contribution. The author of this thesis designed, implemented and performed all experiments reported in this work. Dominik Kowald, Christoph Trattner and Denis Parra assisted in interpreting the experimental results. The author of this thesis took the lead in writing the manuscript. All authors discussed the results and contributed with writing the final manuscript.

MP4

Lacic, E. (2016). Real-Time Recommendations in a Multi-Domain Environment. *In Extended Proceedings at Doctoral Consortium of the 27th ACM Conference on Hypertext and Social Media (HT'16)*

About MP4. This work presents how the ScaR recommender framework can provide a scalable and customizable architecture, which is needed when providing

recommendations in real-times for multiple domains, In order to address both, Research Question 2 and 3, [Lacic, 2016] investigates how an increasing request load (i.e., number of incoming recommendation requests) will impact the system. It becomes apparent that an exponential growth in the runtime happens as the request load increases. But, with scaling the recommender system horizontally, a significant increase in the runtime performance can be seen. This shows that having scalability is an important aspect when supporting multiple domains, as not every domain will have the same request load.

Contribution. The author of this thesis implemented, configured and conducted all the experiments which are reported in this work. This also includes setting up the necessary server infrastructure to run the experiments. The author of this thesis wrote the manuscript.

MP5

Lacic, E., Reiter-Haas, M., Duricic, T., Slawicek, V. and Lex, E. (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 (RecSys'2019). ACM*

About MP5. Learning latent item embeddings has recently become a popular technique for recommender systems, even though many works report on state-of-the-art performance in an offline setting, experiments on the user acceptance and utility of embeddings under real-time constraints are scarce. As this ties to both, the Research Question 3 and 4, [Lacic et al., 2019b] performs a multitude of A/B tests which are evaluated with respect to the Click-Through-Rate (i.e., user acceptance) and runtime performance. This online study is further conducted in the job domain on two different recommendation scenarios, i.e., to recommend similar jobs and personalizing the homepage.

This work shows that with respect to Research Question 3, certain scenarios exist where the runtime performance can be improved alongside the final user acceptance. Moreover, this work opens up the topic of Research Question 4 as it becomes clear that, there exist other factors that drive the user on what kind of recommendation

is expected in a certain scenario. This shows that only measuring accuracy in an offline setting is not enough.

Contribution. The author of this thesis implemented the algorithms which were used for the online study, investigated the necessary requirements for conducting it as well as defined the scope of the experiments. Markus Reiter-Haas and Tomislav Đuričić assisted in deploying the software. Markus Reiter-Haas assisted with preparing the graphical interpretations of the experimental results. The author of this thesis took the lead in writing the manuscript. All authors discussed the results and contributed with writing the final manuscript.

MP6

Lacic, E., Reiter-Haas, M., Kowald, D., Dareddy, M. R., Cho, J. and Lex, E. (2020). Using Autoencoders for Session-based Job Recommendations. In the Journal of *User Modeling and User-Adapted Interaction (UMUAI). Springer*

About MP6. This works tackles a special case of real-time recommender systems, namely, providing recommendations to short-lived anonymous session users. Specifically, [Lacic et al., 2020] presents different autoencoder architectures to encode sessions which can be used to generate a recommendation in a k-nearest neighbor manner. With respect to Research Question 4, the autoencoder architectures together with state-of-the-art session-based approaches are evaluated on measures which go beyond the usually utilized accuracy metric. That is, this works looks into the impact of session-based approaches on the system-based and session-based novelty as well as coverage. The outcome of this work shows that the proposed autoencoder-based methods can achieve a comparable accuracy while outperforming the state-of-the-art approaches with respect to beyond accuracy measures.

Contribution. The author of this thesis conceived the main idea, implemented all the proposed algorithms, as well as prepared and run the experiments that are reported in this work. Manoj Reddy Dareddy and Junghoo Cho assisted in framing the scope of this work. Elisabeth Lex and Dominik Kowald aided in interpreting the experimental results. Markus Reiter-Haas provided the proprietary dataset. The

author of this thesis took the lead in writing the manuscript. All authors discussed the results and contributed with writing the final manuscript.

3.2 Additional Publications

In addition to the main publications, the following 12 published papers have also contributed to the four research questions, but their printed versions are not in the main focus of this thesis. As such, each publication in this section is accompanied by a short overview of its general idea whereas the contribution to the respective research question can be seen in Table 3.

AP1

Lacic, E., Kowald, D., Traub, M., Luzhnica, G., Simon, J., and Lex, E. (2015). Tackling Cold-Start Users in Recommender Systems with Indoor Positioning Systems. *In Proceedings of the 9th ACM Conference on Recommender Systems (RecSys'15)*

About AP1. This paper looks at utilizing additional data sources to tackle the cold-start user problem (i.e., provide recommendations for new users in the system). As such, [Lacic et al., 2015b] showed that creating a user-location network out of the raw location data can produce even better recommendation accuracy for "extreme" cold-start users. This demonstrates that location-based data can be a rich source of useful for recommendations and thus, opens up potential future work on applying it at indoor locations (see Section 4.3).

Contribution. The author of this thesis designed and conducted all the experiments in this work. The author of this thesis took the lead in writing the manuscript. All authors discussed the results and contributed with writing the final manuscript.

AP2

Duricic, T., **Lacic, E.**, Kowald, D., and Lex, E. (2018). Trust-Based Collaborative Filtering: Tackling the Cold Start Problem Using Regular Equivalence. *In Proceedings of the 12th ACM Conference on Recommender Systems (RecSys'18)*

About AP2. This work continues the research on cold-start users and investigates the usage of explicit trust relationships as an additional data source to mitigate this problem. But as trust relationships between users are usually a sparse data source, this paper explored the use of regular equivalence for trust propagation in a cold-start user setting. The evaluation results suggest that trust propagation is an important feature when using trust networks as an additional data source. Moreover, regular equivalence was demonstrated as an effective technique for propagating trust in a network. [Duricic et al., 2018]

Contribution. Tomislav Duricic conceived the original idea as well as conducted all the experiments that are reported in this work. The author of this thesis assisted in the design of the experiments, preparing the source code for the algorithmic baselines, interpreting and reporting the results. Tomislav Duricic took the lead in writing the manuscript. All authors discussed the results and contributed with writing the final manuscript.

AP3

Lacic, E., Kowald, D., Seitlinger, P., Trattner, C., and Parra, D. (2014). Recommending items in social tagging systems using tag and time information. *In Proceedings of the 1st International Workshop on Social Personalisation co-located with the 25th ACM Conference on Hypertext and Social Media (HT'2014)*

About AP3. This publication dives further into the research on the utilization of additional data for the recommendation task. Specifically, [Lacic et al., 2014b] demonstrated that when social tags are additionally available in an online system, the recommendation performance can be improved by utilizing the temporal patterns of their usage.

Contribution. The author of this thesis designed and conducted all the experiments in this work. Dominik Kowald assisted in adapting the algorithms to utilize temporal patterns. The author of this thesis took the lead in writing the manuscript. All authors discussed the results and contributed with writing the final manuscript.

AP4

Reiter-Haas, M., Slawicek, V. and **Lacic, E.** (2017). Studo Jobs: Enriching Data With Predicted Job Labels. *In Workshop on Recommender Systems and Social Network Analysis (RS-SNA'17) co-located with the 17th International Conference on Knowledge Technologies and Data-driven Business (i-KNOW'17)*

About AP4. This work introduces the recommendation problem within the job domain. For this it was first needed to tackle the problem of missing job labels which would be needed for a better navigability of job recommendations. As such, [Reiter-Haas et al., 2017] shows that it is not necessarily needed to only rely on additional data sources when wanting to generate recommendations. Rather, it is possible with a high accuracy to enrich the current data with, for example, labels that in turn can be used as an input for the recommender approach or as a navigation component for the user when exploring at the provided recommendations.

Contribution. Markus Reiter-Haas conducted all the experiments that are reported in this work. The author of this thesis supervised and guided the design of the experiments and helped interpret the results. Markus Reiter-Haas took the lead in writing the manuscript. All authors discussed the results and contributed with writing the final manuscript.

AP5

Lacic, E., Kowald, D., and Trattner, C. (2014). Socrecm: A scalable social recommender engine for online marketplaces. *In Proceedings of the 25th ACM Conference on Hypertext and Social Media (HT'14), pp. 308-310.*

About AP5. The contributions of this work are two-fold. Firstly, it is shown that recommendation quality can be improved for cold-start users by utilizing social data as well as the effect of sequentially increasing the amount of social data. Secondly, this work presents the SocRecM framework which, which is one of the open-source frameworks that is contributed with this thesis. Thus, [Lacic et al., 2014c] has laid the foundation by providing a recommendation framework for online social marketplaces which is then used to extend it for a multi-domain environment.

Contribution. The author of this thesis designed and conducted all the experiments in this work, as well as implemented the presented SocRecM recommendation framework. The author of this thesis also took the lead in writing the manuscript. All authors discussed the results and contributed with writing the final manuscript.

AP6

Traub, M., Kowald, D., **Lacic, E.**, Schoen, P., Supp, G., and Lex, E. (2015). Smart booking without looking: providing hotel recommendations in the TripRebel portal. ,p. 50. ACM. (best demo honourable mention) *In Proceedings of the 15th International Conference on Knowledge Technologies and Data-driven Business (i-KNOW'15)*

About AP6. This work presents a recommender framework for the hotel (i.e., tourism) domain where data updates and new user-hotel interactions should be immediately taken into account for the calculation of recommendations. This is realized by starting to work on implementing the ScaR recommender framework out of the previously mentioned SocRecM framework in order to support the hotel domain. That is, [Traub et al., 2015] presented an adaptation of various state-of-the-art recommender algorithms with respect to scalability and real-time performance. Additionally, the work explored how to apply the use-cases and requirements which are specific for a system that recommends hotels.

Contribution. The author of this thesis implemented the recommendation algorithms used in this work and assisted in setting the research experiments. Matthias Traub took the lead in writing the manuscript. All authors discussed the results and contributed with writing the final manuscript.

AP7

Kowald, D., **Lacic, E.**, Theiler, D., and Lex, E. (2018). AFEL-REC: A Recommender System for Providing Learning Resource Recommendations in Social Learning Environments. *In the Social Interaction-Based Recommender Systems (SIR'18) Workshop co-located with the 27th International Conference on Information and Knowledge Management (CIKM'18)*

About AP7. This paper presents further adaptations of the ScaR framework within the social learning environment. The presented AFEL-REC system shows how to leverage the proposed software architecture in order to provide real-time recommendations of learning resources. In addition, the AFEL-REC system is built to handle any kind of data that is available in social learning environments like metadata of learning resources, user interactions or social tags. As such, [Kowald et al., 2018] provides the experimental results of three recommendation use-cases implemented in AFEL-REC and shows that recommendation accuracy and coverage can be improved by utilizing social data in form of tags.

Contribution. Dominik Kowald designed the experiments reported in this work. The author of this thesis assisted with adapting the ScaR recommender framework for the setting up the experiments as well as interpreting the results. Dominik Kowald took the lead in writing the manuscript. All authors discussed the results and contributed with writing the final manuscript.

AP8

Lacic, E., Traub, M., Kowald, D., Kahr, M., and Lex, E. (2016). Need Help? Recommending Social Care Institutions. *In Workshop on Recommender Systems and Big Data Analytics (RSBDA'16) co-located with the 16th International Conference on Knowledge Technologies and Data-driven Business (i-KNOW'16)*

About AP8. This works demonstrates how to further adapt the ScaR framework for the domain of Social Care Institutions. Moreover, a hybrid algorithm is proposed

which utilizes multiple data sources. That is, it is proposed to dynamically incorporate the time context of historical search results and the gathered negative feedback from previous recommendations [Lacic et al., 2016]. These two factors are especially important to consider in the domain of social care institutions as over time, the locations, availability and, responsibilities of individual institutions change.

Contribution. The author of this thesis designed and implemented the recommendation algorithms used in this work. Matthias Traub took the lead in writing the manuscript. All authors discussed the results and contributed with writing the final manuscript.

AP9

Lacic, E., Traub, M., Kowald, D., and Lex, E. (2015). ScaR: Towards a Real-Time Recommender Framework Following the Microservices Architecture. *In Workshop on Large Scale Recommender Systems (LSRS'15) co-located with the 9th ACM Conference on Recommender Systems (RecSys'15)*

About AP9. This work presents ScaR, a flexible Java-based recommender framework that was developed in this thesis. ScaR adopts an architecture that is based on microservices and utilizes the Apache Solr search engine in order to (1) generate recommendations in real-time, (2) provide scalability of different recommender algorithms, (3) support offline and online evaluations and, (4) incorporate frequent data updates without the necessity of time-expensive model retrainings.

Contribution. The author of this thesis designed and implemented the ScaR recommender framework that is presented in this work. The author of this thesis also conducted all the reported experiments and took the lead in writing the manuscript. All authors provided critical feedback and helped in writing the final manuscript.

AP10

Lacic, E., Kowald, D., and Lex, E. (2018). Neighborhood Troubles: On the Value of User Pre-Filtering To Speed Up and Enhance Recommendations. *In the International Workshop on Entity Retrieval (EYRE'18) co-located with the 27th International Conference on Information and Knowledge Management (CIKM'18)*

About AP10. The focus of this work is on one of the most popular recommender algorithms that is used to date, namely, Collaborative Filtering. The paper tackles the problem of increasing the runtime performance as such can play a crucial role in settings with a high load of recommendation requests and where scaling up the system is not feasible. The paper thus proposes a user pre-filtering step that can be done in a greedy fashion. This leads in a speed up with respect to the runtime performance and in some cases can even increase the recommendation accuracy.

Contribution. The author of this thesis designed and implemented the recommendation algorithms used in this work. The author of this thesis also conducted all the reported experiments and took the lead in writing the manuscript. All authors provided critical feedback and helped in writing the final manuscript.

AP11

Lacic, E., Kowald, D., Reiter-Haas, M., Slawicek, V. and Lex, E. (2018). Beyond Accuracy Optimization: On the Value of Item Embeddings for Student Job Recommendation. *In the 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)*

About AP11. This work explores how the recent popularization of learning item embeddings can be used to improve content-based recommendation performance beyond accuracy. For that, a model from the human memory theory was proposed to combine item embeddings and was applied in the job domain. This led to the

finding that modelling the frequency and recency of user-job interactions can result in more robust recommendations with respect to not only accuracy, but also diversity and novelty.

Contribution. The author of this thesis designed and conducted all the reported experiments in this work. Markus Reiter-Haas and Valentin Slawicek provided the dataset used in this work. The author of this thesis took the lead in writing the manuscript. All authors provided critical feedback and helped in writing the final manuscript.

AP12

Lacic, E., Kowald, D., Theiler, D., Traub, M., Kuffer, L., Lindstaedt, S., and Lex, E. (2019). Evaluating Tag Recommendations for E-Book Annotation Using a Semantic Similarity Metric. *In REVEAL Workshop co-located with ACM Conference on Recommender Systems (RecSys'2019)*

About AP12. This paper presents a hybrid tag recommender system for e-books and proposes to evaluate the system on a novel semantic similarity metric. The paper thus shows how to validate a tag recommender on semantic matches instead of focusing on direct "hits", where the recommendation needs to be an exact syntactical match. Interpreting a recommender in this way can still lead to a correct accuracy assessment but is in addition beneficial in better interpreting the recommendation quality.

Contribution. The author of this thesis designed the evaluation measures and algorithms, as well as conducted all the reported experiments in this work. Dominik Kowald assisted in the design of the experiments and interpreting the results. Lucky Kuffer provided the dataset used in this work. The author of this thesis and Dominik Kowald took a joint lead and contributed equally in writing the manuscript. All authors provided critical feedback and helped in writing the final manuscript.

Utilizing Online Social Network and Location-Based Data to Recommend Products and Categories in Online Marketplaces

Emanuel Lacic[1( )], Dominik Kowald[1], Lukas Eberhard[2], Christoph Trattner[3], Denis Parra[4], and Leandro Balby Marinho[5]

[1] Know-Center, Graz University of Technology, Graz, Austria
{elacic,dkowald}@know-center.at
[2] IICM, Graz University of Technology, Graz, Austria
lukas.eberhard@tugraz.at
[3] Norwegian University of Science and Technology, Trondheim, Norway
chritrat@idi.ntnu.no
[4] Pontificia Universidad Catlica de Chile, Santiago, Chile
dparra@ing.puc.cl
[5] UFCG, Campina Grande, Brazil
lbmarinho@dsc.ufcg.edu.br

Abstract. Recent research has unveiled the importance of online social networks for improving the quality of recommender systems and encouraged the research community to investigate better ways of exploiting the social information for recommendations. To contribute to this sparse field of research, in this paper we exploit users' interactions along three data sources (marketplace, social network and location-based) to assess their performance in a barely studied domain: recommending products and domains of interests (i.e., product categories) to people in an online marketplace environment. To that end we defined sets of content- and network-based user similarity features for each data source and studied them isolated using an user-based Collaborative Filtering (CF) approach and in combination via a hybrid recommender algorithm, to assess which one provides the best recommendation performance. Interestingly, in our experiments conducted on a rich dataset collected from SecondLife, a popular online virtual world, we found that recommenders relying on user similarity features obtained from the social network data clearly yielded the best results in terms of accuracy in case of predicting products, whereas the features obtained from the marketplace and location-based data sources also obtained very good results in case of predicting categories. This finding indicates that all three types of data sources are important and should be taken into account depending on the level of specialization of the recommendation task.

Keywords: Recommender systems · Online marketplaces · SNA · Social data · Location-based data · SecondLife · Collaborative filtering · Item recommendations · Product recommendations · Category prediction

M. Atzmueller et al. (Eds.): MUSE/MSM 2013, LNAI 8940, pp. 96–115, 2015.
DOI: 10.1007/978-3-319-14723-9_6

1 Introduction

Research on recommender systems has gained tremendous popularity in recent years. Especially since the hype of the social Web and the rise of social media and networking platforms such as Twitter or Facebook, recommender systems are acknowledged as an essential feature helping users to, for instance, discover new connections between people or resources. Especially in the field of e-commerce sites, i.e., online marketplaces, current research is dealing with the improvement of the prediction task in order to recommend products that are more likely to match peoples preferences.

Typically, these online systems calculate personalized recommendations using only one data source, namely marketplace data (e.g., implicit user feedback such as previously viewed or purchased products - see also e.g., [1–3]). Although this approach has been well established and performs reasonably well, nowadays, online marketplaces often also have the opportunity to leverage additional information about the users coming from social and location-based data sources (e.g., via Facebook-connect). Even though previous research has shown that this kind of data can be useful in the wide field of recommender systems (see Sect. 2), it remains an open problem how to fully exploit these additional data sources (social network and location-based data) to improve the recommendation task in online marketplaces.

Moreover, it is often not the most important thing in online marketplaces to predict the exactly right products to the users but to suggest domains of interests (i.e., product categories) the users could like and could use for further browsing (e.g., [1,4]). Thus, it is not only important to investigate to which extent social and location-based data sources can be used to improve product recommendations but also to which extent this data can also be used for the recommendation of product categories.

To contribute to this sparse field of research, in this paper we present a first take on this problem in form of a research project that aims at understanding how different sources of interaction data can help in recommending products and categories to people in an online marketplace. In this respect, we are particularly interested in studying the efficiency of different user similarity features derived from various dimensions, not only from the marketplace but also from online social networks and location-based data to recommend products and categories to people via a user-based Collaborative Filtering (CF) approach (we have chosen a user-based CF approach since user-based CF is not only a well-established recommender algorithm but also allows us to incorporate various user-based similarity features coming from different data sources, which has been shown to play an important role in making more accurate predictions [4,5]). Specifically, we raise the following two research questions:

- *RQ1*: To which extent can user similarity features derived from marketplace, social network and location-based data sources be utilized for the recommendation of products and categories in online marketplaces?

– *RQ2*: Can the different marketplace, social network and location-based user similarity features and data sources be combined in order to create a hybrid recommender that provides more robust recommendations in terms of prediction accuracy, diversity and user coverage?

In order to address these research questions, we examined content-based and network-based user similarity feature sets for user-based CF over three data sources (marketplace, social and location-based data) as well as their combinations using a hybrid recommender algorithm and assessed the results via a more comprehensive set of recommender evaluation metrics than previous works. The study was conducted using a large-scale dataset crawled from the virtual world of SecondLife. In this way, we could study the utility of each user similarity feature separately as well as combine them in the form of hybrid approaches to show which combinations, per data source and globally, provide the best recommendations in terms of recommendation accuracy, diversity and user coverage. Summing this up, the contributions of this work are the following:

– Contrary to previous work in this area [1,2], we not only employ one source of data (marketplace) for the problem of predicting product purchases but show how data coming from three different sources (marketplace, social and location-based data) can be exploited in this context.
– In contrast to related work in the field, we provide also an extensive evaluation of various content-based and network-based user similarity features via user-based Collaborative Filtering as well as their combinations via a hybrid recommender approach.
– Finally, we also provide evidence to what extent top-level and sub-level purchase categories can be predicted which is in contrast to previous work (e.g., [1]) where the authors only focused on the problem recommending top-level categories to the users.

To the best of our knowledge, this is the first study that offers such a comprehensive user similarity feature selection and evaluation for product and category recommendation in online marketplaces.

Overall, the paper is structured as follows: we begin by discussing related work in Sect. 2. Then we present the datasets (Sect. 3) and the feature description (Sect. 4) used in our extensive evaluation. After that, we present our experimental setup in Sect. 5 and show the results of our experiments in Sect. 6. Finally, on Sect. 7 we conclude the paper and discuss the outlook.

2 Related Work

Most of the literature that leverages social data for recommendations is focused on recommending users, (e.g., [2,6]), tags (e.g., [7]) or points-of-interest (e.g., [4]), although some works have exploited social information for item or product recommendation, being the most important ones model-based. Jamali et al. [5]

introduced SocialMF, a matrix factorization model that incorporates social relations into a rating prediction task, decreasing RMSE with respect to previous work. Similarly, Ma et al. [4] incorporated social information in two models of matrix factorization with social regularization, with improvements in both MAE and RMSE for rating prediction. Among their evaluations, they concluded that choosing the right similarity feature between users plays an important role in making a more accurate prediction.

On a more general approach, Karatzoglou et al. [8] use implicit feedback and social graph data to recommend places and items, evaluating with a ranking task and reporting significant improvements over past related methods. Compared to these state-of-the-art approaches, our focus on this paper is at providing a richer analysis of feature selection (similarity features) with a more comprehensive evaluation than previous works, and in a rarely investigates domain: product recommendation in a social online marketplace. For instance, in Guo et al. [2] or Trattner et al. [3] the authors leveraged social interactions between sellers and buyers in order to predict sellers to customers. Other relevant work in this context is the study of Zhang and Pennacchiotti [1] who showed how top-level categories can be better predicted in a cold-start setting on eBay by exploiting the user's "likes" from Facebook.

3 Datasets

In our study we rely on three datasets[1] obtained from the virtual world SecondLife (SL). The main reason for choosing SL over real world sources are manifold but mainly due to the fact that currently there are no other datasets available that comprise marketplace, social and location data of users at the same time. For our study we focused on users who are contained in all three sources of data, which are 7,029 users in total. To collect the data (see Table 1) we crawled the SL platform as described in our previous work [9,10].

3.1 Marketplace Dataset

Similar to eBay, SecondLife provides an online trading platform where users can trade virtual goods with each other. Every seller in the SL marketplace[2] owns her own seller's store and publicly offers all of the store's products classified into different categories (a hierarchy with up to a maximum of four different categories per product). Furthermore, sellers can apply meta-data such as price, title or description to their products. Customers in turn, are able to provide reviews to products.

We extracted 29,802 complete store profiles, with corresponding 39,055 trading interactions, and 2,162,466 products, out of which 30,185 were purchased.

[1] **Note:** The datasets could be obtained by contacting the fourth author of this work.
[2] https://marketplace.secondlife.com/.

Table 1. Basic statistics of the SL datasets used in our study.

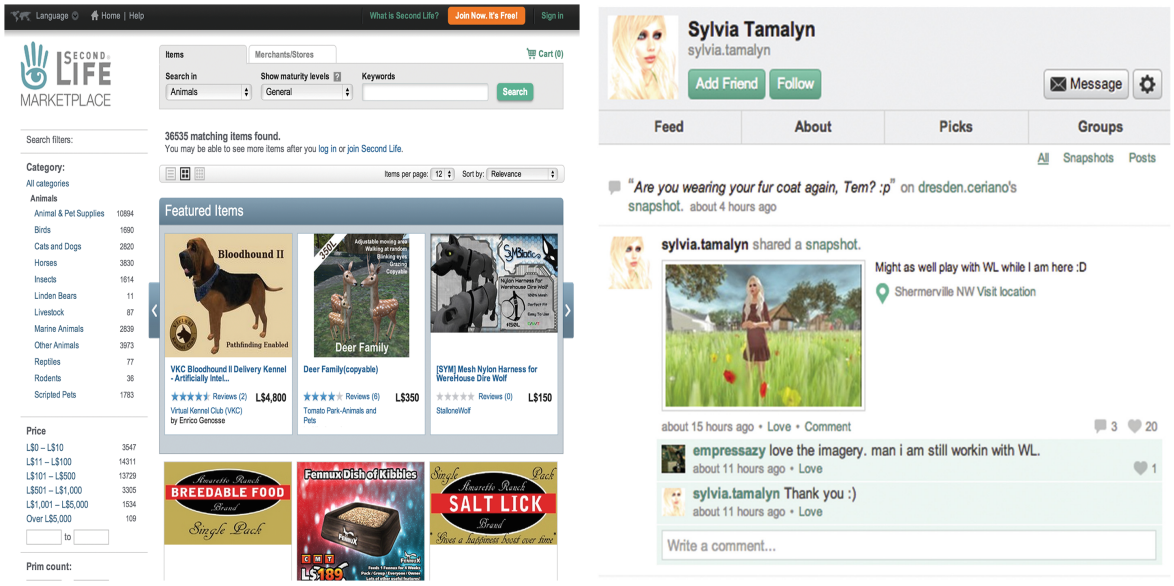
Marketplace dataset (Market)	
Number of products	30,185
Number of products with categories	24,276
Number of purchases	39,055
Number of purchases with categories	31,164
Mean number of purchases per user	5.56
Mean number of purchases per products	1.29
Mean number of categories per product	2.86
Number of top-level categories	23
Number of low-level categories	532
Mean number of top-level categories purchase	1,354.96
Mean number of low-level categories purchase	58.58
Number of sellers	8,149
Mean number of purchases per seller	3.70
Online social network dataset (Social)	
Number of interactions	490,236
Mean number of interactions	69.75
Number of groups	39,180
Mean number of groups per user	9,419
Number of interests	5.57
Mean number of interests per user	1.34
Location-based dataset (Location)	
Number of different favorite locations	10,538
Mean number of favorite locations per user	5.77
Number of different shared locations	5,736
Mean number of shared locations per user	1.94
Number of different monitored locations	1,887
Mean number of monitored locations per user	6.52

From the purchased products, 24,276 are described using categories which are differentiated in top-level categories and low-level categories (i.e., the assigned product categories on the lowest possible level of the category hierarchy). An example of a product in the marketplace of SL is shown in the first image of Fig. 1.

3.2 Online Social Network Dataset

The online social network MySecondLife[3] is similar to Facebook with regard to postings: users can interact with each other by sharing text messages and

[3] https://my.secondlife.com/.



(a) SecondLife store (b) SecondLife social stream

Fig. 1. Examples for a store in the marketplace and a social stream of an user in the online social network of the virtual world SecondLife.

commenting or loving (= liking) these messages. From the extracted 7,029 complete user profiles, we gathered 39,180 different groups users belong to, 9,419 different interests users defined for themselves and 490,236 interactions between them. The second image of Fig. 1 shows an example of an user profile in the online social network of SL.

3.3 Location-Based Dataset

The world of Second Life is contained within regions, i.e., locations which are independent from each other. Overall, we extracted three different sources of location-based data in our experiments:

(a) Favored Locations: Every user of SL can specify up to 10 so-called "Picks" in their profile representing her favorite locations that other users can view in the user's MySecondLife profile. We found that the extracted users picked 40,558 locations from 10,538 unique locations;
(b) Shared Locations: Users in SL can also share information about their current in-world position through in-world pictures called "snapshots", which also include in-world GPS information (similar as Foursquare). Overall, we identified 13,637 snapshots in 5,736 unique locations;
(c) Monitored Locations: As in real life, users in SL can create events in the virtual world and publicly announce them in a public event calendar. We collected these events, with an accurate location and start time, and extracted 157,765 user-location-time triples, with 1,887 unique locations.

4 Feature Description

As shown in our previous work (e.g., [10]), similarities between users can be derived in two different ways: either we calculate similarities between users on the content (= meta-data) provided directly by the user profiles or on the network structure of the user profiles interacting with each other. In the following sections we present more details about these ideas.

4.1 Content-Based User Similarity Features

We define our set of content-based user similarity features based on different types of entities or meta-data information that are directly associated with the user profiles in our data sources. In the case of the marketplace dataset these entities are purchased products, product categories and sellers of the products, in the case of the social network these are groups and interests the users have assigned, and in the case of the location-based data source these are favored locations, shared locations and monitored locations. Formally, we define the entities of a user u as $\Delta(u)$ in order to calculate the similarity between two users, u and v.

The first content-based user similarity feature we induce is based on the entities two users have in common. It is called *Common Entities* and is given by:

$$sim(u, v) = |\Delta(u) \cap \Delta(v)| \tag{1}$$

The second similarity feature, *Total Entities*, is defined as the union of two users' entities and is calculated by:

$$sim(u, v) = |\Delta(u) \cup \Delta(v)| \tag{2}$$

These two user similarity features are combined by *Jaccard's Coefficient for Entities* as the number of common entities divided by the total number of entities:

$$sim(u, v) = \frac{|\Delta(u) \cap \Delta(v)|}{|\Delta(u) \cup \Delta(v)|} \tag{3}$$

4.2 Network-Based User Similarity Features

In our experiments we consider all networks as an undirected graph $G\langle V, E\rangle$ with V representing the user profiles and $e = (u, v) \in E$ if user u performed an action on v (see also [10]). In the case of the social network, these actions are defined as social interactions, which are a combination of likes, comments and wallposts. In the case of the location-based dataset, actions between users are determined if they have met each other in the virtual world at the same time in the same location[4]. Furthermore, the weight of an edge $w_{action}(u, v)$ gives the frequency of

[4] **Note:** We derived the networks in our study from the location-based dataset only for the monitored locations, since the exact timestamps are not available for the favored nor the shared locations in the datasets.

a specific action between two users u and v. Finally, this network structure also let us determine the neighbors of users in order to calculate similarities based on this information. We define the set of neighbors of a node $v \in G$ as $\Gamma(v) = \{u \mid (u, v) \in E\}$.

The first network-based user similarity feature we introduce, uses the number of *Directed Interactions* between two users and is given by:

$$sim(u, v) = w_{action}(u, v) \tag{4}$$

In contrast to *Directed Interactions*, the following user similarity features are based on the neighborhood of two users: The first neighborhood similarity feature is called *Common Neighbors* and represents the number of neighbors two users have in common:

$$sim(u, v) = |\Gamma(u) \cap \Gamma(v)| \tag{5}$$

To also take into account the total number of neighbors of the users, we introduced *Jaccard's Coefficient for Common Neighbors*. It is defined as:

$$sim(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|} \tag{6}$$

A refinement of this feature was proposed as *Adamic Adar* [11], which adds weights to the links since not all neighbors in a network have the same tie strength:

$$sim(u, v) = \sum_{z \in \Gamma(u) \cap \Gamma(v)} \frac{1}{log(|\Gamma(z)|)} \tag{7}$$

Another related similarity feature introduced by Cranshaw et al. [12], called *Neighborhood Overlap*, measures the structural overlap of two users. Formally, this is written as:

$$sim(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u)| + |\Gamma(v)|} \tag{8}$$

The *Preferential Attachment Score*, first mentioned by Barabasi et al. [13], is another network-based similarity feature with the goal to prefer active users in the network. This score is the product of the number of neighbors of each user and is calculated by:

$$sim(u, v) = |\Gamma(u)| \cdot |\Gamma(v)| \tag{9}$$

5 Experimental Setup

In this section we provide a detailed description of our experimental setup. First, we describe the recommender approaches we have chosen in order to evaluate our three data sources (marketplace, social network and location-based data) as well as our derived user similarity features for the task of recommending products and categories. Afterwards, we describe the evaluation methodology and the metrics used in our study.

5.1 Recommender Approaches

In this subsection we describe the recommender approaches we have used to tackle our research questions described in the introductory section of this paper. All mentioned approaches have been implemented into our scalable big data social recommender framework *SocRec* [9,14], an open-source framework which can be obtained from our Github repository[5].

Baseline. As baseline for our study, we used a simple MostPopular approach recommending the most popular products or categories in terms of purchase frequency to the users.

Recommending Products. The main approach we adopt to evaluate our data sources and user similarity features for the task of recommending products is a *User-based Collaborative Filtering (CF)* approach. The basic idea of this approach is that users who are more similar to each other, e.g., have similar taste, will likely agree or rate on other resources in a similar manner [15]. Out of the different CF approaches, we used the non-probabilistic user-based nearest neighbor algorithm where we first find the k-nearest similar users and afterwards recommend the resources of those user as a ranked list of top-N products to the target user that are new to her (i.e., she has not purchased those products in the past). As outlined before, we have chosen this approach since user-based CF is not only a well-established recommender algorithm but also allows us to incorporate various user-based similarity features coming from different data sources, which has been shown to play an important role in making more accurate predictions [4,5].

The similarity values of the user pairs $sim(u, v)$ are calculated based on the user similarity features proposed in Sect. 4 (i.e., constructing the neighborhood). Based on these similarity values, each item i of the k most similar users for the target user u is ranked using the following formula [15]:

$$pred(u, i) = \sum_{v \in neighbors(u)} sim(u, v) \tag{10}$$

Recommending Categories. For the task of recommending categories and in contrast to previous work (e.g., [1]), we are not only focusing here on the prediction of top-level categories but also on the prediction of low-level categories. The prediction of categories was implemented as an extension of product predictions. Thus, for each product in the list of recommended products (i.e., the products obtained from the k-nearest neighbors of user u based on a user similarity feature), we extracted the assigned category on the highest level in the case of predicting top-level categories and the assigned category on the lowest level in the case of predicting low-level categories. Afterwards, we assigned a score to

[5] https://github.com/learning-layers/SocRec.

each extracted category e_i in the set of all extracted categories E_u for the target user u based on a similar method as proposed in [1]:

$$pred(u, e_i) = \frac{purc(E_u, e_i)}{\sum_{e \in E_u} purc(E_u, e)} \tag{11}$$

where $purc(E_u, e_i)$ gives the number of times the category e_i occurs in the set of all extracted categories E_u for user u.

Combining User Similarity Features and Data Sources. To further explore how to combine our data sources and features for recommendation, we investigated different hybridization methods (see also [16]). The hybrid approach chosen in the end is known as *Weighted Sum*. The score of each recommended item in the *Weighted Sum* algorithm is calculated as the weighted sum of the scores for all recommender approaches. It is given by:

$$W_{rec_i} = \sum_{s_j \in S} (W_{rec_i, s_j} \cdot W_{s_j}) \tag{12}$$

where the combined weighting of the recommended item i, W_{rec_i}, is given by the sum of all single weightings for each recommended item in an approach W_{rec_i, s_j} multiplied by the weightings of the recommender approaches W_{s_j}. We weighted each recommender approach W_{s_j} based on the nDCG@10 value obtained from the individual approaches (calculated based on an additional evaluation set where we withheld 20 purchased products - see Sect. 5.2).

We also experimented with other hybrid approaches, known as *Cross-source* and *Mixed Hybrid* [16]. However, these approaches have not yielded better results than the *Weighted Sum* algorithm.

5.2 Evaluation Method and Metrics

To evaluate the performance of each approach in a recommender setting, we performed a number of off-line experiments. Therefore, we split the SL dataset in two different sets (training and test set) using a method similar to the one described in [9], i.e., for each user we withheld 10 purchased products from the complete dataset and added them to the test set to be predicted. Since we did not use a p-core pruning technique to prevent a biased evaluation, there are also users with less than 10 relevant products. We did not include these users into our evaluation since they did not have enough purchase data available that could be used to produce reasonable recommendations based on the marketplace data, although this data is worthwhile for our user-based CF approach to find suitable neighbors. Thus, we used a post-filtering method, where all the recommendations were still calculated on the whole datasets but accuracy estimates were calculated only based on these filtered user profiles (= 959 users).

To finally quantify the performance of each of our recommender approaches, we used a diverse set of well-established metrics in recommender systems [17,18]. These metrics are as follows:

Recall (R@k) is calculated as the number of correctly recommended products divided by the number of relevant products, where r_u^k denotes the top k recommended products and R_u the list of relevant products of a user u in the set of all users U. Recall is given by [19]:

$$R@k = \frac{1}{|U|} \sum_{u \in U} \left(\frac{|r_u^k \cap R_u|}{|R_u|}\right) \tag{13}$$

Precision (P@k) is calculated as the number of correctly recommended products divided by the number of recommended products k. Precision is defined as [19]:

$$P@k = \frac{1}{|U|} \sum_{u \in U} \left(\frac{|r_u^k \cap R_u|}{k}\right) \tag{14}$$

Normalized Discounted Cumulative Gain (nDCG@k) is a ranking-dependent metric that not only measures how many products can be correctly predicted but also takes the position of the products in the recommended list with length k into account. The nDCG metric is based on the *Discounted Cumulative Gain (DCG@k)* which is given by [20]:

$$DCG@k = \sum_{k=1}^{|r_u^k|} \left(\frac{2^{B(k)} - 1}{log_2(1+k)}\right) \tag{15}$$

where $B(k)$ is a function that returns 1 if the recommended product at position i in the recommended list is relevant. nDCG@k is calculated as DCG@k divided by the ideal DCG value iDCG@k which is the highest possible DCG value that can be achieved if all the relevant products would be recommended in the correct order. Taken together, it is given by the following formula [20]:

$$nDCG@k = \frac{1}{|U|} \sum_{u \in U} \left(\frac{DCG@k}{iDCG@k}\right) \tag{16}$$

Diversity (D@k), as defined in [17], can be calculated as the average dissimilarity of all pairs of resources in the list of recommended products r_u^k. Given a distance function $d(i, j)$ that is the distance, or the dissimilarity between two products i and j, D is given as the average dissimilarity of all pairs of products in the list of recommended products [17]:

$$D@k = \frac{1}{|U|} \sum_{u \in U} \left(\frac{1}{k \cdot (k-1)} \sum_{i \in R} \sum_{j \in r_u^k, j \neq i} d(i, j)\right) \tag{17}$$

User Coverage (UC) is defined as the number of users for whom at least one product recommendation could have been calculated ($|U_r|$) divided by total number of users $|U|$ [21]:

$$UC = \frac{|U_r|}{|U|} \tag{18}$$

All mentioned performance metrics are calculated and reported based on the top-10 recommended products.

6 Results

In this section we highlight the results of our experiments for predicting products, low-level and top-level categories in terms of algorithmic performance in order to tackle our two research questions presented in Sect. 1. Our evaluation has been conducted in two steps: in the first step we compared the different recommender approaches with the corresponding user similarity features isolated (*RQ1*), see Table 2) and in the second step we combined these approaches in the form of hybrid recommendations (*RQ2*, see Table 3). All results are presented by recommender accuracy, given by nDCG@10, P@10 (Precision) and R@10 (Recall), D@10 (Diversity) and UC (User Coverage).

6.1 Recommendations Based on Single User Similarity Features

The results for the recommendation of products, low-level categories and top-level categories, using content-based and network-based user similarity features derived from our three data sources (marketplace, social and location-based data), are shown in Table 2 in order to address our first research question (*RQ1*). The results also include the *Most Popular (MP)* approach as a baseline. Additionally, the performance of the different data sources is also shown in Fig. 2 in form of Recall/Precision plots.

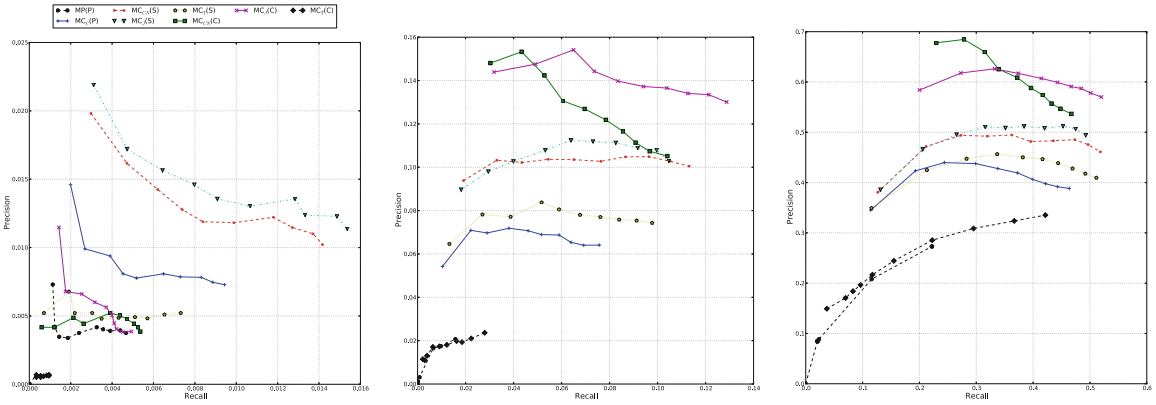
Recommending Products. Regarding the task of predicting product purchases (first column in Table 2), the best results in terms of recommender accuracy are reached by the network-based features based on interactions (e.g., loves, comments, wallposts) between the users in the social network. Surprisingly these approaches clearly outperform the user-based CF approaches relying on marketplace data, which implies that social interactions of the users are a better predictor to recommend products to people than marketplace data.

Another interesting finding is that the neighborhood-based features (*Common Neighbors*, *Jaccard*, *Neighborhood Overlap* and *Adamic/Adar*) also seem to be better indicators to determine the similarity between users than the direct interactions between these pairs. Only the *Preferential Attachment Score* based recommender approach does not perform well in this context, although it still performs better than the features derived from the marketplace data source. This is to some extent expected and reveals that the individual's taste is more driven by the user's peers rather than by the popular users in the SecondLife social network.

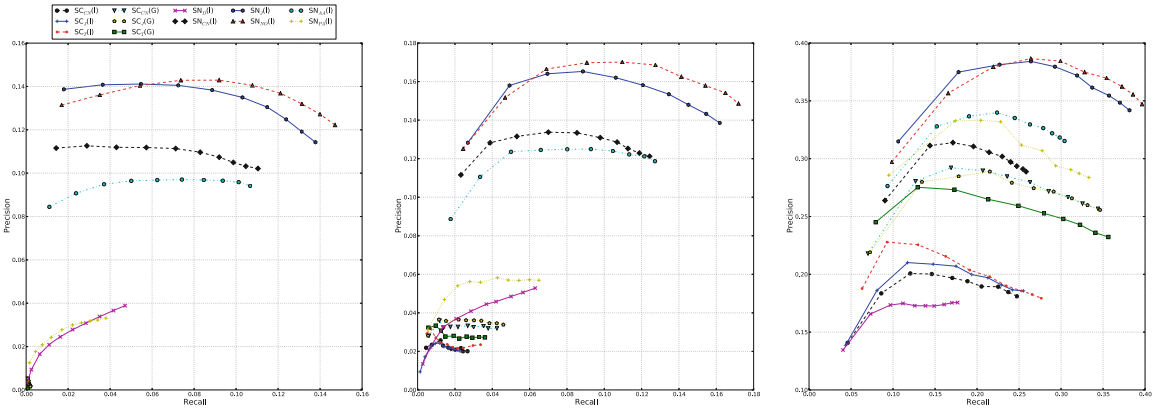
In terms of the marketplace and location-based user similarity features, the results reveal that they do not provide high estimates of accuracy. This is

Table 2. Results for the user-based CF approaches based on various user similarity features showing their performance for the tasks of predicting products, low-level categories and top-level categories, respectively (*RQ1*). *Note:* Bold numbers indicate the highest accuracy values per feature set and "*" indicate the overall highest accuracy estimates.

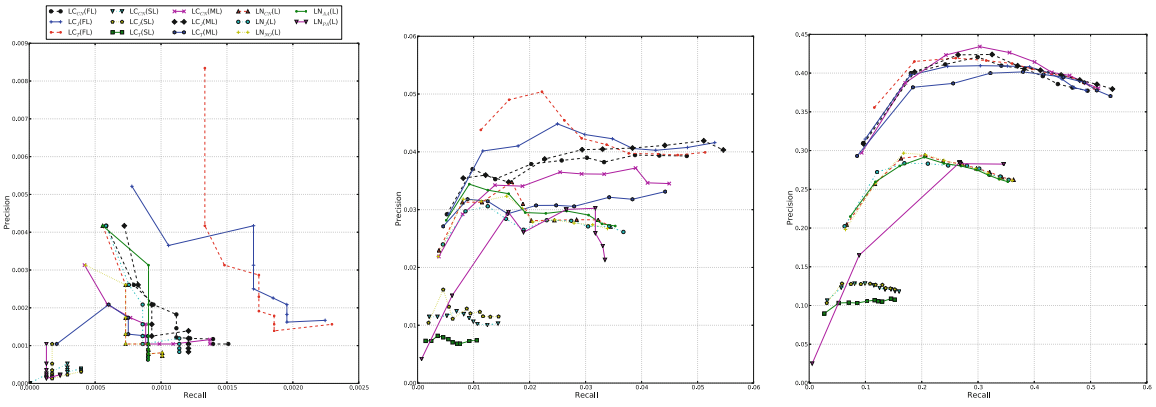
			Products			low-level categories			top-level categories				
		User Sim. Feature	*nDCG*@10	*P*@10	*R*@10	*nDCG*@10	*P*@10	*R*@10	*nDCG*@10	*P*@10	*R*@10	*D*@10	*UC*
		Most Popular	.0048	.0037	.0047	.0185	.0207	.0157	.2380	.2730	.2221	.6392	100.0%
Market	Content	Common Purchases	.0097	.0073	.0094	.0724	.0641	.0757	.4557	.3884	.4636	.5892	90.51%
		Common Sellers	.0146	.0102	.0142	.1119	.1005	.1132	.5251	.4610	.5183	.6372	99.06%
		Jaccard Sellers	**.0158**	**.0114**	**.0154**	.1092	.1029	.1047	.5061	.4940	.4927	.6054	99.06%
		Total Sellers	.0065	.0052	.0073	.0929	.0743	.0977	.5079	.4094	.5113	.6566	99.06%
		Common Categories	.0050	.0039	.0054	.1090	.1051	.1041	.5073	.5366	.4674	.6123	99.48%
		Jaccard Categories	.0058	.0039	.0049	**.1361**	**.1301**	**.1288**	**.5456***	**.5701***	**.5200***	.6364	99.48%
		Total Categories	.0007	.0006	.0009	.0225	.0236	.0280	.3317	.3353	.4215	.6575	99.48%
Social	Content	Common Groups	.0022	.0010	.0014	.0402	.0320	.0425	.3233	**.2567**	.3439	.4307	64.13%
		Jaccard Groups	**.0027**	**.0016**	**.0021**	**.0433**	**.0339**	**.0459**	**.3272**	.2557	**.3464**	.4332	64.13%
		Total Groups	.0006	.0005	.0007	.0324	.0272	.0361	.3214	.2323	.3563	.4466	64.13%
		Common Interests	.0005	.0002	.0002	.0235	.0201	.0267	.2285	.1810	.2474	.3185	46.51%
		Jaccard Interests	.0003	.0002	.0003	.0219	.0201	.0227	.2424	.1857	.2551	.3161	46.51%
		Total Interests	.0004	.0002	.0003	.0285	.0235	.0337	.2639	.1793	.2765	.3319	46.51%
	Network	Directed Interactions	.0345	.0389	.0471	.0582	.0528	.0630	.1743	.1756	.1769	.2169	38.79%
		Common Neighbors	.1107	.1021	.1104	.1216	.1212	.1243	.2633	.2887	.2584	.3300	62.88%
		Jaccard Common Neighbors	.1381	.1143	.1378	.1523	.1386	.1618	.3726	.3419	.3814	.4455	71.53%
		Neighborhood Overlap	**.1434***	**.1222***	**.1471***	**.1620***	**.1486***	**.1719***	.3855	.3471	.3965	.4514	71.53%
		Adamic/Adar	.1013	.0941	.1067	.1241	.1187	.1272	.3028	.3153	.3042	.3762	69.34%
		Pref. Attach. Score	.0317	.0331	.0380	.0630	.0569	.0650	.3202	.2838	.3332	.4420	70.70%
Location	Content	Common Favored Locations	.0019	.0010	.0015	.0427	.0393	.0481	.4674	.3773	.4946	.6437	96.35%
		Jaccard Favored Locations	.0028	**.0017**	.0022	.0472	**.0416**	.0531	.4636	.3777	.4919	.6490	96.35%
		Total Favored Locations	**.0031**	.0016	**.0023**	.0459	.0400	.0513	.4794	.3802	.5061	.6635	96.35%
		Common Shared Locations	.0003	.0003	.0004	.0130	.0103	.0144	.1449	.1180	.1599	.2067	30.45%
		Jaccard Shared Locations	.0005	.0003	.0004	.0134	.0115	.0145	.1420	.1208	.1522	.2042	30.45%
		Total Shared Locations	.0000	.0000	.0000	.0092	.0074	.0106	.1340	.1076	.1520	.2031	30.45%
		Common Monitored Locations	.0016	.0010	.0014	.0408	.0345	.0449	.4825	**.3804**	.5139	.6734	98.23%
		Jaccard Monitored Locations	.0017	.0008	.0012	**.0473**	.0403	**.0546**	**.4987**	.3795	**.5387**	.6760	98.23%
		Total Monitored Locations	.0011	.0006	.0009	.0366	.0331	.0442	.4770	.3703	.5354	.6757	98.23%
	Network	Common Neighbors	.0015	.0007	.0010	.0298	.0271	.0345	.3377	.2623	**.3632**	.4609	67.05%
		Jaccard Common Neighbors	**.0016**	**.0008**	**.0011**	**.0322**	.0261	**.0367**	.3306	.2623	.3545	.4579	67.05%
		Neighborhood Overlap	.0014	.0007	.0010	.0295	.0267	.0339	.3359	.2614	.3608	.4615	67.05%
		Adamic/Adar	.0015	.0006	.0009	.0320	**.0272**	.0353	.3332	.2598	.3530	.4595	67.05%
		Pref. Attach. Score	.0003	.0002	.0002	.0270	.0213	.0335	**.3634**	**.2825**	.3458	.4583	70.59%



(a) Products (b) low-level categories (c) top-level categories



(d) Products (e) low-level categories (f) top-level categories



(g) Products (h) low-level categories (i) top-level categories

Fig. 2. Recall/Precision plots for the single user similarity features derived from the marketplace (a, b, c), social (d, e, f) and location-based (g, h, i) data sources, showing the performance of each feature for $k = 1$–10 recommended items, low-level categories or top-level categories, respectively (*RQ1*). *Note:* Each feature name in the legends is derived in the following way: the first two letters describe the data source, the subscript denotes the user similarity feature and the value in brackets defines the used data field (e.g., $SN_{NO}(I)$ stands for the Social Network data source, the Neighborhood Overlap similarity feature and Interactions data field).

interesting since our previous work [10,22] showed that these features perform extremely well in predicting tie strength or social interactions between users. However, the features derived from the marketplace and the location-based data sources provide the best results with respect to Diversity (D) and User Coverage (UC).

Recommending Categories. Regarding the tasks of predicting low-level and top-level categories, the second and third column of Table 2 report the accuracy estimates for the different user similarity features based on the extracted categories. As expected, all user similarity features end up with a much higher accuracy than for predicting products, especially in the case of top-level categories, because of the lower level of specialization of these recommendation tasks. In the case of the low-level category predictions, the approaches based on social interaction features still perform better than the approaches based on features of the marketplace or location-based data sources. Interestingly, the content-based user similarity features derived from the social network as well as the location-based features, which performed the worst at product predictions, perform much better for low-level categories, now also outperforming the *MP* baseline.

In the case of the top-level category recommendations, it can be seen that the user similarity features of all three data sources provide quite similar results in terms of recommender accuracy. The approach based on the Jaccard's coefficient for categories performs best in terms of nDCG@10, P@10 and R@10. This result is very interesting since this feature is based on the marketplace data source that provided quite bad results in the case of product predictions. Summed up, we see that user similarity features derived from all data sources are very useful indicators for recommendations, although they depend on the level of specialization of the recommendation task (*RQ1*).

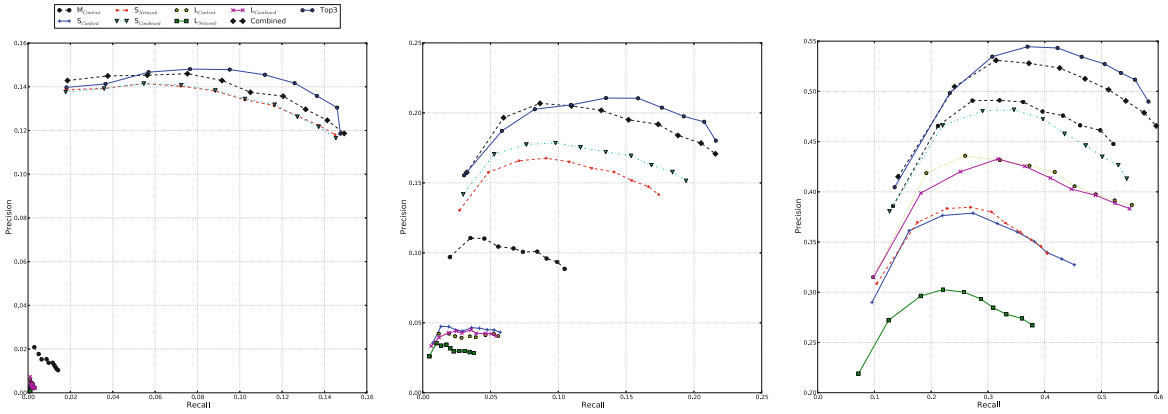
6.2 Recommendations Based on Combined Data Sources

The findings of the last subsection suggests that a combination of features from all three data sources (marketplace, social network and location-based data) should provide more robust recommendations in case of both tasks, predicting products and categories (*RQ2*). Thus, Table 3 shows the results of the hybrid approaches based on theses data sources in order to tackle our second research question. As before, the first column indicates the results for the product prediction, the second column for the low-level category prediction and the third column for the top-level category prediction. Additionally, Fig. 3 shows the performance of the hybrid approaches based on the three data sources for $k = 1$–10 recommended products, low-level categories or top-level categories, respectively, in form of Recall/Precision plots.

Recommending Products. Regarding the product prediction task, we see again that the recommender approaches based on the social network data source clearly outperform the ones based on the marketplace and location-based data sources as well as the *MP* baseline. Furthermore, when combining all three data sources, not only the overall recommendation accuracy is increased with respect

Table 3. Results for the hybrid approaches based on our three data sources for the tasks of predicting products, low-level categories and top-level categories. The results show that all three data sources (marketplace, social network and location-based data) are important indicators for calculating recommendations, since a hybrid combination of all data sources provided the best results in case of predicting products, low-level categories and top-level categories (*RQ2*). *Note:* Bold numbers indicate the highest accuracy values across the data sources and "*" indicate the overall highest accuracy estimates.

			Products			low-level categories			top-level Categories				
	Sets		*nDCG*@10	*P*@10	*R*@10	*nDCG*@10	*P*@10	*R*@10	*nDCG*@10	*P*@10	*R*@10	*D*@10	*UC*
	Most Popular		.0082	.0021	.0122	.0185	.0207	.0157	.2380	.2730	.2221	.5945	100.00%
Weighted Sum	Market	Content	.0151	.0103	.0142	.1051	.0886	.1046	.5232	**.4476**	.5210	.6421	99.79%
	Social	Content	.0030	.0019	.0025	.0702	.0432	.0571	.5555	.3274	.4523	.5603	81.65%
		Network	.1416	**.1180**	.1450	**.2454***	.1416	.1740	.5708	.3385	.4047	.4591	71.53%
		Combined	**.1418**	.1165	**.1454**	.2104	**.1515**	**.1942**	**.5901**	.4131	.5446	.6245	92.70%
	Location	Content	.0036	.0021	.0030	.0556	.0406	.0556	.5535	.3870	**.5535**	.6923	100.00%
		Network	.0015	.0007	.0010	.0535	.0284	.0378	.5359	.2672	.3783	.4864	70.59%
		Combined	.0036	.0022	.0031	.0540	.0406	.0540	.5497	.3832	.5497	.6914	100.00%
	Combined		**.1460***	**.1187***	**.1493***	.2163	.1708	.2159	**.5978***	.4656	**.5965***	.6642	100.00%
	Combined Top 3		.1459	.1186	.1475	.2161	**.1801***	**.2161***	.5829	**.4898***	.5829	.6530	100.00%



(a) Products (b) low-level categories (c) top-level categories

Fig. 3. Recall/Precision plots for the hybrid approaches showing the performance of each data source for $k = 1$–10 recommended products, low-level categories or top-level categories, respectively (*RQ2*).

to nDCG@10, P@10 and R@10, but also the User Coverage (UC) is increased to the maximum of 100 %.

This means that the hybrid approach combines the strengths of the user similarity features of all three data sources in order to be capable of providing accurate recommendations for all users in the datasets. Another hybrid approach shown in Table 3 combines only the best user similarity features from each data source (referred to as *Top 3*) and reaches higher accuracy estimates, but lower Diversity (D).

Recommending Categories. In contrast to the results of the product predictions, that showed that the recommender based on the social network data source clearly outperforms the recommenders based on the marketplace and location-based data sources, the results of the category predictions (second and third column of Table 3) do not show that big differences between the three data sources. With respect to the low-level category predictions, we again observe that the recommender based on the social network data source still provides the highest accuracy estimates.

Interestingly, in this case also the recommenders based on the other two data sources provide reasonable results, which has not been the case of predicting products, where the location-based recommender even was outperformed by the *MP* baseline. Based on these results we would assume that marketplace and location-based data sources are suitable of providing accurate predictions in more general recommendation tasks. The results for the top-level category predictions prove this assumption since in this case the recommenders based on marketplace and location-based data sources even provide better results in terms of recommender accuracy, Diversity and User Coverage than the one based on social network data in case of the content-based features. As before, the combination of all three data sources provide again the best results. Summed up, this results prove our assumption derived from *RQ1*, that all three data sources

are important for calculating recommendations, since a combination of all data sources provided the best results in case of predicting products, low-level categories and top-level categories (*RQ2*).

7 Conclusions and Future Work

In this work we presented first results of a recently started project that tries to utilize various user similarity features derived from three data sources (marketplace, social network and location-based data) to recommend products and points of interests (i.e., low-level and top-level categories) to people in an online marketplace setting. This section concludes the paper with respect to our two research questions and gives an outlook into the future.

The first research question of this work (*RQ1*) dealt with the question as to which extent user similarity features derived from marketplace, social network and location-based data sources can be utilized for the recommendation of products and categories in online marketplaces. To tackle this question we implemented various user-based Collaborative Filtering (CF) approaches based on the user similarity features from the data sources and tested them isolated. As the results have shown, the user-based CF approaches that utilize features of online social network data to calculate the similarities between users performed best in case of predicting products, significantly outperforming the other approaches relying on both – marketplace and location-based user data. However, this behavior changed in the case of predicting low-level and top-level categories where the differences between the three data sources got substantially smaller. Surprisingly, with respect to the top-level category predictions, the marketplace and location-based features even reached the highest results in terms of accuracy, Diversity (D) and User Coverage (UC).

These results showed that user similarity features of all three data sources are important indicators for recommendations and suggests that combining them should result into more robust recommendations, especially in cases of multiple recommendation tasks on different levels of specialization (topics and categories). Thus, our second research question (*RQ2*) tried to tackle the question if the different marketplace, social network and location-based user similarity features and data sources can be combined in order to create a hybrid recommender that provides more robust recommendations in terms of prediction accuracy, diversity and user coverage. In order to address this question we implemented and evaluated hybrid recommenders that combined the features of the data sources. The results proved our assumption and showed that hybrid recommender that combined user similarity features of all three data sources provided the best results across all accuracy metrics (nDCG@10, P@10, R@10) and all settings (product, low-level category and top-level category recommendations). Moreover, this hybrid recommender also provided a User Coverage of 100 % and thus, is able to provide these accurate recommendation to all users in the datasets.

Although the results of this study are based on a dataset obtained from the virtual world SecondLife, we believe that it bears great potential to create a

sequence of interesting studies that may have implications for the "real" world (see e.g., [23]). For instance, one of the potential interesting issues we are currently exploring is predicting products and categories to users in a cold-start setting (i.e., for users that only have purchases a few or even no products in the past) by a diversity of features. Other important work we plan is the use of state-of-the-art model-based approaches in order to assess whether the signals extracted from similarity features in the current analysis can be replicated (in the case of social data) or improved (in the case of location data) for different recommendation tasks.

We have also shown that using the interaction information between users improves not only the task of product recommendation, but also the recommendation of low-level and top-level categories. Thus, we are also interested in studying the extent to which recommendations can be improved by utilizing content-based similarity features derived from the users' social streams.

Acknowledgments. This work is supported by the Know-Center and the EU funded project Learning Layers (Grant Agreement 318209). Moreover, parts of this work were carried out during the tenure of an ERCIM "Alain Bensoussan" fellowship programme. The Learning Layers project is supported by the European Commission within the 7th Framework Program, under the DG Information society and Media (E3), unit of Cultural heritage and technology-enhanced learning. The Know-Center is funded within the Austrian COMET Program - Competence Centers for Excellent Technologies - under the auspices of the Austrian Ministry of Transport, Innovation and Technology, the Austrian Ministry of Economics and Labor and by the State of Styria. COMET is managed by the Austrian Research Promotion Agency (FFG).

References

1. Zhang, Y., Pennacchiotti, M.: Predicting purchase behaviors from social media. In: Proceedings of WWW '13, pp. 1521–1532 (2013)
2. Guo, S., Wang, M., Leskovec, J.: The role of social networks in online shopping: Information passing, price of trust, and consumer choice. In: Proceedings of EC '11, pp. 157–166. ACM (2011)
3. Trattner, C., Parra, D., Eberhard, L., Wen, X.: Who will trade with whom? Predicting buyer-seller interactions in online trading platforms through social networks. In: Proceedings of WWW '14, pp. 387–388. ACM (2014)
4. Ma, H., Zhou, D., Liu, C., Lyu, M.R., King, I.: Recommender systems with social regularization. In: Proceedings of WSDM '11, pp. 287–296. ACM (2011)
5. Jamali, M., Ester, M.: A matrix factorization technique with trust propagation for recommendation in social networks. In: Proceedings of RecSys '10, pp. 135–142. ACM, New York (2010)
6. Bischoff, K.: We love rock'n'roll: analyzing and predicting friendship links in Last.fm. In: Proceedings of WebSci '12, pp. 47–56. ACM (2012)
7. Feng, W., Wang, J.: Incorporating heterogeneous information for personalized tag recommendation in social tagging systems. In: Proceedings of KDD '12, pp. 1276–1284. ACM (2012)

8. Delporte, J., Karatzoglou, A., Matuszczyk, T., Canu, S.: Socially enabled preference learning from implicit feedback data. In: Blockeel, H., Kersting, K., Nijssen, S., Železný, F. (eds.) ECML PKDD 2013, Part II. LNCS, vol. 8189, pp. 145–160. Springer, Heidelberg (2013)
9. Lacic, E., Kowald, D., Parra, D., Kahr, M., Trattner, C.: Towards a scalable social recommender engine for online marketplaces: The case of apache solr. In: Proceedings of WWW '14, pp. 817–822. ACM (2014)
10. Steurer, M., Trattner, C.: Acquaintance or partner? Predicting partnership in online and location-based social networks. In: Proceedings of ASONAM'13. IEEE/ACM (2013)
11. Adamic, L., Adar, E.: Friends and neighbors on the web. Soci. Netw. **25**, 211–230 (2003)
12. Cranshaw, J., Toch, E., Hong, J., Kittur, A., Sadeh, N.: Bridging the gap between physical location and online social networks. In: Proceedings of the 12th ACM International Conference on Ubiquitous Computing, pp. 119–128. ACM (2010)
13. Barabási, A., Albert, R.: Emergence of scaling in random networks. Science **286**, 509–512 (1999)
14. Lacic, E., Kowald, D., Trattner, C.: Socrecm: A scalable social recommender engine for online marketplaces. In: Proceedings of HT '14, pp. 308–310 (2014)
15. Schafer, J.B., Frankowski, D., Herlocker, J., Sen, S.: Collaborative filtering recommender systems. In: The adaptive web. Springer (2007) 291–324.
16. Bostandjiev, S., O'Donovan, J., Höllerer, T.: Tasteweights: a visual interactive hybrid recommender system. In: Proceedings of RecSys '12, pp. 35–42. ACM (2012)
17. Smyth, B., McClave, P.: Similarity vs. Diversity. In: Aha, D.W., Watson, I. (eds.) ICCBR 2001. LNCS (LNAI), vol. 2080, pp. 347–361. Springer, Heidelberg (2001)
18. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. (TOIS) **22**, 5–53 (2004)
19. Van Rijsbergen, C.J.: Foundation of evaluation. J. Doc. **30**, 365–373 (1974)
20. Parra, D., Sahebi, S.: Recommender systems: sources of knowledge and evaluation metrics. In: Velásquez, J.D., Palade, V., Jain, L.C. (eds.) Advanced Techniques in Web Intelligence-2. SCI, vol. 452, pp. 149–176. Springer, Heidelberg (2013)
21. Ge, M., Delgado-Battenfeld, C., Jannach, D.: Beyond accuracy: evaluating recommender systems by coverage and serendipity. In: Proceedings of the Fourth ACM Conference on Recommender Systems, pp. 257–260. ACM (2010)
22. Steurer, M., Trattner, C.: Predicting interactions in online social networks: an experiment in second life. In: Proceedings of the 4th International Workshop on Modeling Social Media, p. 5. ACM (2013)
23. Szell, M., Sinatra, R., Petri, G., Thurner, S., Latora, V.: Understanding mobility in a social petri dish. Scientific Reports 2 (2012)

Tailoring Recommendations for a Multi-Domain Environment

Emanuel Lacic
Know-Center Graz
Graz, Austria
elacic@know-center.at

Dominik Kowald
Know-Center Graz
Graz, Austria
dkowald@know-center.at

Elisabeth Lex
Graz University of Technology
Graz, Austria
elisabeth.lex@tugraz.at

ABSTRACT

Recommender systems are acknowledged as an essential instrument to support users in finding relevant information. However, the adaptation of recommender systems to multiple domain-specific requirements and data models still remains an open challenge. In the present paper, we contribute to this sparse line of research with guidance on how to design a customizable recommender system that accounts for multiple domains with heterogeneous data. Using concrete showcase examples, we demonstrate how to setup a multi-domain system on the item and system level, and we report evaluation results for the domains of (i) LastFM, (ii) FourSquare, and (iii) MovieLens. We believe that our findings and guidelines can support developers and researchers of recommender systems to easily adapt and deploy a recommender system in distributed environments, as well as to develop and evaluate algorithms suited for multi-domain settings.

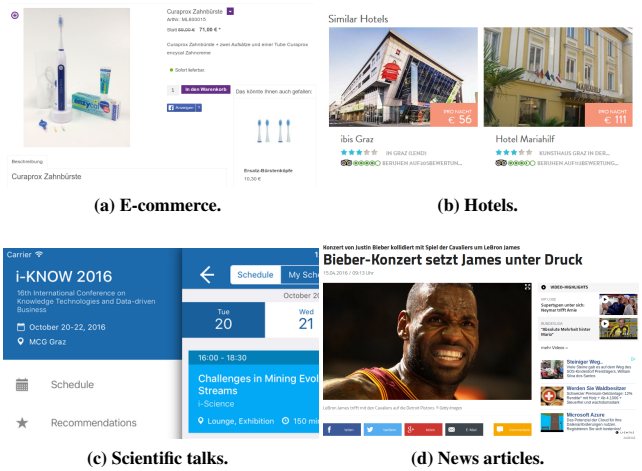
KEYWORDS

Recommender Systems; Multi-Domain Recommendation; Heterogeneous Data; Customizing Recommendation Approaches

1 INTRODUCTION

In the past decade, there has been a vast amount of research in the field of recommender systems. Most of these systems offer recommendations adapted for items belonging to a single domain (e.g., movies , music , news , etc.). However, supporting different domain-specific data models is still an open challenge, which is neglected by many recommender systems and that has just been recently taken up by the recommender systems' research community.

The work of [2] has actually been the first attempt to define the concept of a domain in the context of recommender systems. The authors distinguish between four different domain notations, namely (i) attribute level, where items are of the same type (e.g., movie genres), (ii) type level, where items are of similar types but share some attributes (e.g., movies and tv shows), (iii) item level, where items are not of the same type and differ in most or all attributes (e.g., movies and books) and, (iv) system level, where items and users belong to different systems (e.g., LastFM and MovieLens). Moreover, they distinguish between the concept of multi-domain recommendations and cross-domain recommendations. The goal of cross-domain recommendation is to utilize the knowledge derived from one or more source domains in order to generate predictions for a target domain. However, they also state that multi-domain approaches have mainly focused on the provision of cross-domain recommendations by jointly considering user preferences for items in various systems.



(a) E-commerce. (b) Hotels.

(c) Scientific talks. (d) News articles.

Figure 1: Providing recommendations in multiple domains requires supporting heterogeneous data structures, allow for domain-specific algorithm customization, as well as to support service isolation and fault tolerance.

Other related works implicitly share this definition [2] and focus on cross-domain recommendations rather than on multi-domain ones. Thus, the main focus has been to tackle the data sparsity problem (e.g., via active transfer learning [14]) by utilizing Collaborative Filtering [4, 7, 10] and Content-based Filtering approaches [3, 11].

In the present paper, we build upon these works and we extend the scope of multi-domain recommender systems by introducing several guidelines concerning topics such as data heterogeneity and customization that should be taken into consideration. We base our findings on real-world applications (e.g., e-commerce[1], hotels[2], scientific talks[3] or news articles[4] – just to name a few) and propose a practical approach on how to support muti-domain recommendations on both the item and system level as described by [2].

To the best of our knowledge, this is the first work which addresses the question of what design decisions should be taken into consideration when building a recommender system for a multi-domain environment.

[1] http://www.mymanou.com/
[2] https://www.triprebel.com/
[3] http://uscn.me/rr220
[4] http://www.clef-newsreel.org/

RecSysKTL Workshop @ ACM RecSys '17, August 27, 2017, Como, Italy

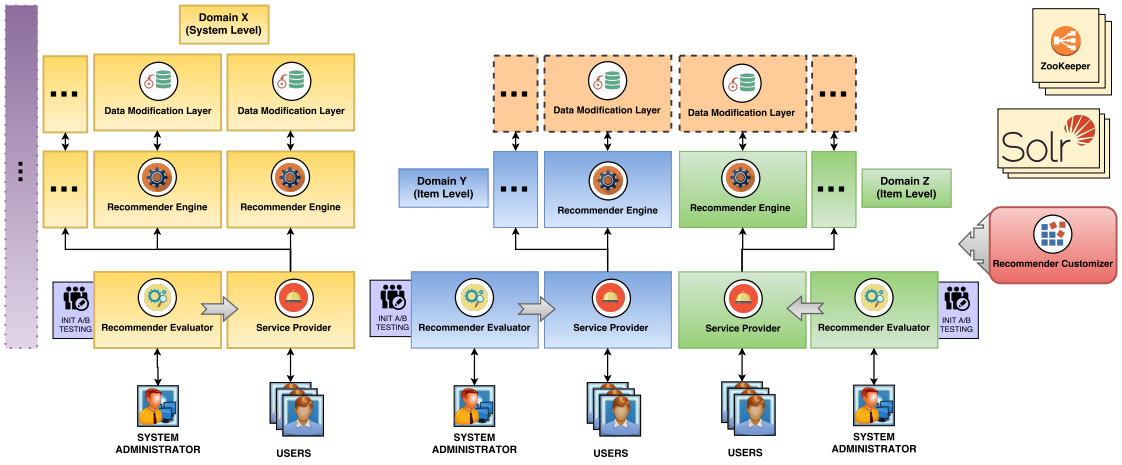


Figure 2: Proposed system architecture of a multi-domain recommendation environment showing how the various modules work together. Each module is a standalone HTTP server, which is aware of the location (i.e., URL) of its communicating partners. In case of an item level multi-domain scenario, the same data storage is shared between the domains and customization via recommender profiles provides the domain-specific algorithm configuration.

2 A MULTI-DOMAIN RECOMMENDATION APPROACH

In this section, we categorize and propose guidelines to extend the notation of a multi-domain recommender system. We base our guidelines on our previous work [5, 6, 12], which we have already applied in live settings in various domains (e.g., e-commerce, hotels, conference or news as shown in Figure 1).

Thus, we propose four issues that should be addressed when providing multi-domain recommendation, i.e., (i) service isolation, (ii) data heterogeneity, (iii) recommender customization, and (iv) fault tolerance. While we focus on item and system level domain notations, our findings can be adapted for both the attribute and type level by means of additional recommender customization (e.g., filtering by the item category).

2.1 Service Isolation

When supporting multi-domain recommendations on the system level, effective hardware utilization is crucial. Actually, as each domain has different requirements with respect to the request load, the hardware utilization rate can be improved by sharing the same hardware resources across multiple domains. Additionally, in such a scenario, performance isolation is crucial. Specifically, a high request load in combination with possible performance-intensive operations needed for one domain should not impact the performance in another domain. For example, news recommender systems usually have a requirement of providing session-based recommendations within 100 milliseconds and need to cope with challenging load peaks during morning hours and the lunch break at working days [13]. Thus, in cases where the request load is too large for a particular domain, it should be possible to dynamically scale the system and handle such performance intensive load peeks.

Correspondingly, we propose to separate a recommender system's logic into several microservices by adopting the Microservices architecture design pattern[5]. An example of such an architecture is shown in Figure 2. Here, five different modules take care of (i) data handling, (ii) calculating recommendations, (iii) balancing incoming recommendation requests, (iv) domain-specific customization, and (v) evaluating recommendations. To support horizontal scaling and to coordinate all deployed modules, as well as the corresponding system level domain assignments, we use Apache ZooKeeper[6]. This overall approach can be extended with virtualization and container technologies such as Docker[7] or LXC[8]. These lightweight resource containers provide features such as portability, more efficient scheduling and resource management, as well as less virtualization overhead, which are beneficial when implementing a multi-domain recommender on the system level.

2.2 Heterogeneous Data

As the amount of data is doubled approximately every 40 months [8], most recommender systems migrate from traditional databases to distributed systems that can scale more easily and handle massive streams of heterogeneous data. As such, a multi-domain recommender system needs to handle a diverse set of data (e.g., ratings, views, likes, check-ins, etc.), while at the same time enabling an easy integration of new types by modifying the underlying schema.

In our case, we leverage the Apache Solr search engine and found its schema-less mode[9] to be a great fit to support multi-domain recommendations on an item level, as it allows dynamic schema construction by indexing data without the need to edit it manually.

[5]http://microservices.io/patterns/microservices.html
[6]http://zookeeper.apache.org/
[7]http://www.docker.com
[8]https://linuxcontainers.org/
[9]https://cwiki.apache.org/confluence/display/solr/Schemaless+Mode

```
{
  "id": "e12c4fb-ba85-46d5-896d-af65d1f3b48c",
  "item": "5a2bc423-15dc-47d3-8a8c-f543dc267a7c",
  "users_listened": [3907, 57017]
  "users_listened_count": 2
  "domain": "LastFM"
},
{
  "id": "8fe89bab-9631-4e87-81a0-a8dd3ffba774c",
  "user": "23861",
  "item": 4105,
  "rating": 1.0
  "domain": "MovieLens"
}
```

Listing 1: Example of different schema strategies to store data in the same place for item level multi-domain recommendations.

For example, as shown in Listing 1, we could easily derive different data structures to store and generate recommendations in the corresponding music and movie domain. Moreover, we found that the capability for horizontal scaling (i.e., creating shards and replicas) is extremely important when integrating new domains on an item-level (see Figure 2) as such a strategy easily increases the amount of data that needs to be stored and processed.

2.3 Customizing Recommendation Approaches

An important aspect of a multi-domain recommender system is customization. Typically, different application domains have different domain-specific data features, which means that, for example, a music recommender could solely use implicit user interactions (e.g., listened songs), whereas an e-commerce recommender could use explicit ones (e.g., ratings). On top of that, one would also need to separately determine and setup the correct algorithmic parameters for each domain. For example, in case of the Collaborative Filtering algorithm, the similarity function (e.g., Cosine or Jaccard similarity) and the neighbourhood size need to be determined for each domain. For other domains, one may need to setup custom filtering criteria (e.g., recommend items that are suited for minors or are part of a specific category). Thus, a multi-domain recommendation approach should be aware of the underlying data structures and domain-specific parameters.

As such, we propose to outsource the domain-specific algorithm setup into so-called recommender profiles. In our applications, we defined a customizer module (see Figure 2), which enables to set domain-specific algorithm configurations and to transfer it to modules utilized in a specific domain environment. This way, we can manage domain-specific configurations and dynamically integrate additional domains. An example of such recommender profiles for the music domain is given in Listings 2, 3 and 4.

Here, we define a configuration by a unique reference id, a reference to the algorithm implementation (e.g., class name) and the specific domain-relevant parameters. In case a recommender profile is created or updated for a particular domain at runtime, the changes need to be propagated throughout the whole system to every domain-dependant module. As such, each module from the same domain will be informed about the changes and the updated profile will be used as soon as a new recommendation request is received.

```
id: ktl_mp_lastfm
# reference for a MP implementation
algorithm: MostPopularGeneric

# algorithm specific parameters
parameters:
  # item-level domain?
  domain: lastFM
  # on what do I calculate the popularity?
   count_fields : [ users_listened_count ]
    user_action_fields : [ users_listened ]
```

Listing 2: A MostPopular recommender profile for the LastFM domain.

```
id: ktl_ub_cf_lastfm
# reference for a user-based-CF implementation
algorithm: GenericUBCF

# algorithm specific parameters
parameters:
  # item-level domain?
   domain: lastFM
  # How to calculate user similarity ?
    similarity_function : OVERLAP # JACCARD, COSINE, etc.
    neighbourhood_size: 40
    user_action_fields : [ users_listened ]
```

Listing 3: A Collaborative Filtering recommender profile for the LastFM domain.

```
id: ktl_hybrid_cs_lastfm
# reference for a hybrid implementation
algorithm: CrossSourceHybrid

# algorithm specific parameters
parameters:
  # item-level domain given by combining profiles
  profile_ids : [ ktl_mp_lastfm , ktl_ub_cf_lastfm ]
  recommender_weights: [0.1, 0.9]
```

Listing 4: A Cross-Source Hybrid recommender profile for the LastFM domain.

2.4 Fault Tolerance

Deploying multiple modules in a distributed manner increases the probability of unexpected behaviour (e.g., hardware shutdown, I/O problems, software bugs, etc.). As we have proposed to use microservices, it is not necessary to cope with central node failures as it is the case with a master-slave architecture. In case a module fails, ZooKeeper, or any other orchestration service like Eureka or Consul, should remove the faulty module from its list of "live nodes" and no further requests will be redirected to it.

Thus, the module will not necessarily cause any major problems as long as there is another module of the same type available. When experiencing a high request load, it should be possible to deploy and register an additional module to ZooKeeper on the fly. To further improve the reliability of the system, multiple ZooKeeper instances can be used in a cluster in order to overcome the outage of single instances. In such a way, the runtime performance can be guaranteed for both item and system level multi-domain recommendations.

3 DOMAIN EXPERIMENTS

In order to demonstrate the application of our guidelines for providing recommendations in multiple domains, we performed several experiments on well-known datasets used in recommender systems research, (i.e., LastFM[10], Foursquare[11] and MovieLens20M[12]). The LastFM dataset consists of 359,348 users, 268,736 artists and 17,559,530 implicit user interactions that denote the listening relationship between users and artists. Foursquare provides 2,809,581 ratings on a 5-star scale for different venues (i.e., restaurants) and contains 2,153,471 users as well as 1,143,092 venues in general. The MovieLens20M dataset has 138,493 users, 27,032 movies as well as 19,999,603 movie reviews on a 5-star rating scale with a step size of 0.5 (i.e., a 10-star scale).

We focused on providing recommendations for cold-start users and as such, we removed all users that interacted with more than 20 items. Next, we split the remaining data in two different sets (i.e., training and test sets) using a method similar to the one described in [9]. Thus, for each user, we withheld 10 items that were used for testing and the rest was used for training. This has resulted in an evaluation set of 2,409 users for LastFM, 41,628 users for FourSquare and 4,486 users for the MovieLens20M dataset. On these datasets, we evaluated a simple MostPopular (MP) approach (e.g., Listing 2), a user-based Collaborative Filtering (CF) approach (e.g., Listing 3) as well as a hybrid combination [1] of both (e.g., Listing 4). Additionally, we evaluated different neighborhood sizes N for the CF approach in order to optimize the results. For reasons of simplicity, we used a naive item overlap metric to measure the similarity between users (i.e., $OV(u_t, u_c) = |\Delta(u_t) \cap \Delta(u_c)|$, where $\Delta(u)$ corresponds to the set of items some user u has interacted with in the past). However, as shown in Listing 3 this is a domain-specific parameter that could be easily adapted.

The results of our evaluation are shown in Table 1 by means of the nDCG@10 and User Coverage (UC) metrics. The aim of this simple experiment was to show how different parameter setups can impact the performance in different domains. As shown, the neighbourhood size is one example of a parameter that needs to be optimized for a specific domain. By choosing the best parameter combination for the hybrid approach, we can provide more robust recommendations for all users in each domain.

4 CONCLUSION

In this work, we presented our approach on providing recommendations in a multi-domain environment. Specifically, we introduced the concept of recommender profiles in order to customize existing algorithms with domain-specific configuration. Apart from that, we provided guidelines with respect to service isolation, heterogeneous data and fault tolerance. Finally, we provided customization examples as well as evaluation results for the domains of (i) LastFM, (ii) FourSquare, and (iii) MovieLens. We believe that our findings and proposed guidelines are of use for developers and researchers of recommender systems to tailor and develop recommendations for multi-domain and distributed environments.

[10] http://mtg.upf.edu/node/1671

[11] https://archive.org/details/201309_foursquare_dataset_umn

[12] https://grouplens.org/datasets/movielens/20m/

	Approach		*nDCG@10*	*UC*
LastFM	MP		.0180	100%
	CF	N = 20	.1113	93.70%
		N = 30	.1129	
		N = 40	**.1135**	
		N = 50	.1120	
		N = 60	.1112	
	Hybrid		.1005	100%
Foursquare	MP		.0256	100%
	CF	N = 20	.0364	49.58%
		N = 30	.0403	
		N = 40	.0426	
		N = 50	.0440	
		N = 60	**.0452**	
	Hybrid		.0339	100%
MovieLens	MP		.0658	100%
	CF	N = 20	.0910	100%
		N = 30	.0945	
		N = 40	.0981	
		N = 50	.0965	
		N = 60	**.0984**	
	Hybrid		.0999	100%

Table 1: Evaluation results of our multi-domain experiment.

For future work, we plan to perform a more elaborate study using the proposed recommender profiles to study differences in domain-specific configurations (e.g., does a semantic relationship impact the choice of domain-specific parameters like the similarity metric or different filtering criteria?). Moreover, we plan to investigate datasets with textual contents in order to further explore how hybrid weightings may impact the performance in a specific domain with respect of not only accuracy but also diversity.

Acknowledgment. This work was funded by the Horizon 2020 project MoreGrasp (643955).

REFERENCES

[1] S. Bostandjiev, J. O'Donovan, and T. Höllerer. Tasteweights: a visual interactive hybrid recommender system. In *Proc.*, RecSys '12, pages 35–42. ACM, 2012.
[2] I. Cantador, I. Fernández-Tobías, S. Berkovsky, and P. Cremonesi. Cross-domain recommender systems. In *Recommender Systems Handbook*. Springer, 2015.
[3] A. M. Elkahky, Y. Song, and X. He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *Proc. WWW'15*.
[4] S. Gao, H. Luo, D. Chen, S. Li, P. Gallinari, and J. Guo. Cross-domain recommendation via cluster-level latent factor model. In *Proc. of ECML-PKDD'13*.
[5] E. Lacic, D. Kowald, and C. Trattner. Socrecm: A scalable social recommender engine for online marketplaces. In *Proc. of the ACM Hypertext 2014*.
[6] E. Lacic, M. Traub, D. Kowald, and E. Lex. Scar: Towards a real-time recommender framework following the microservices architecture. In *Proc. of LSRS'15*.
[7] B. Loni, Y. Shi, M. Larson, and A. Hanjalic. Cross-domain collaborative filtering with factorization machines. In *ECIR*, pages 656–661. Springer, 2014.
[8] A. McAfee and E. Brynjolfsson. Big Data: The management revolution. *Harvard Business Review*, 90(10):60–68, 2012.
[9] D. Parra-Santander and P. Brusilovsky. Improving collaborative filtering in social tagging systems for the recommendation of scientific articles. In *Proc. of WI-IAT '10*, pages 136–142. IEEE Computer Society.
[10] S. Sahebi and P. Brusilovsky. It takes two to tango: An exploration of domain pairs for cross-domain collaborative filtering. In *Proc. of ACM RecSys 2015*.
[11] S. Sahebi and T. Walker. Content-based cross-domain recommendations using segmented models. In *CBRecSys@ RecSys*, pages 57–64, 2014.
[12] M. Traub, D. Kowald, E. Lacic, P. Schoen, G. Supp, and E. Lex. Smart booking without looking: Providing hotel recommendations in the triprebel portal. In *Proc. of i-KNOW '15*.
[13] S. Werner and A. Lommatzsch. Optimizing and evaluating stream-based news recommendation algorithms. In *CLEF (Working Notes)*, pages 813–824, 2014.
[14] L. Zhao, S. J. Pan, E. W. Xiang, E. Zhong, Z. Lu, and Q. Yang. Active transfer learning for cross-system recommendation. In *AAAI*, 2013.

Towards a Scalable Social Recommender Engine for Online Marketplaces: The Case of Apache Solr

Emanuel Lacić
Graz University of Technology
Graz, Austria
elacic@know-center.at

Dominik Kowald
Know-Center
Graz, Austria
dkowald@know-center.at

Denis Parra
Pontificia Universidad Catolica
Santiago, Chile
dparra@ing.puc.cl

Martin Kahr
BLANC-NOIR GmbH
Graz, Austria
martin.kahr@blanc-noir.at

Christoph Trattner
Know-Center
Graz, Austria
ctrattner@know-center.at

ABSTRACT

Recent research has unveiled the importance of online social networks for improving the quality of recommenders in several domains, what has encouraged the research community to investigate ways to better exploit the social information for recommendations. However, there is a lack of work that offers details of frameworks that allow an easy integration of social data with traditional recommendation algorithms in order to yield a straight-forward and scalable implementation of new and existing systems. Furthermore, it is rare to find details of performance evaluations of recommender systems such as hardware and software specifications or benchmarking results of server loading tests.

In this paper we intend to bridge this gap by presenting the details of a social recommender engine for online marketplaces built upon the well-known search engine Apache Solr. We describe our architecture and also share implementation details to facilitate the re-use of our approach by people implementing recommender systems. In addition, we evaluate our framework from two perspectives: (a) recommendation algorithms and data sources, and (b) system performance under server stress tests. Using a dataset from the SecondLife virtual world that has both trading and social interactions, we contribute to research in social recommenders by showing how certain social features allow to improve recommendations in online marketplaces. On the platform implementation side, our evaluation results can serve as a baseline to people searching for performance references in terms of scalability, model training and testing tradeoffs, real-time server performance and the impact of model updates in a production system.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering*

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
http://dx.doi.org/10.1145/2567948.2579245 .

Keywords

social recommender engine; scalability; online marketplaces; Apache Solr

1. INTRODUCTION

Recommender systems aim at helping users to find relevant information in an overloaded information space [11]. Although there are well known methods (Content-based [1], Collaborative Filtering [16, 18], Matrix Factorization [10]) and libraries to implement, evaluate and extend recommenders (Apache Mahout[1], Graphlab[2], MyMediaLite[3], among others [8]), the deployment of a real-time recommender from scratch which considers a combination of algorithms and data sources, the effect of large volumes of data and hardware configuration, or the impact of model updates in the recommender performance remains unsolved or at least not publicly available for the research community. In this paper, we contribute to bridge this gap between research on recommender algorithms and system deployment by presenting in detail our approach to implement a social marketplace recommender. We describe our solution in terms of the data model that allows modularity and extensibility, as well as the system architecture that relies on the Apache Solr project to facilitate the scaling of our approach to big data.

We support our decision of using Solr on recent work that shows the strong relation between memory-based recommendation approaches and ready-to-use text analytic techniques [3]. Since Solr has already most of these techniques implemented, documented and optimized by a well established open-source community, we believe that it provides not only a good basis for a large-scale search engine but it also provides a good foundation to implement an efficient and scalable social recommender engine for online marketplaces. We appeal for Apache Solr since one might have to consider several dimensions of the data or already existing indices based on Apache Lucene, the kernel search engine Apache Solr is built upon.

To evaluate our implementation we consider diverse metrics – accuracy and ranking metrics along diversity, and user coverage – to unveil not only the performance of each recommender algorithm isolated but also to show the importance of each single feature and data source. In addition, a performance benchmarking experiment was conducted to show the scalability of our approach.

[1]http://mahout.apache.org
[2]http://graphlab.org
[3]http://www.mymedialite.net

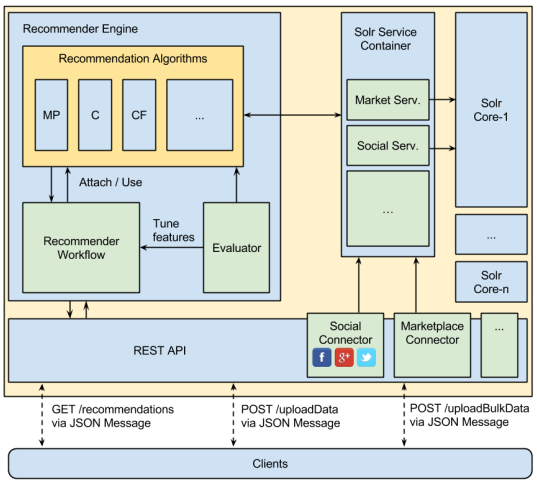


Figure 1: System architecture.

In detail, the paper is structured as follows: we begin by describing our system architecture and implementation details in Section 2. Section 3 describes the experimental setup we chose to evaluate our framework, while in Section 4 we present the results in terms of recommendation quality and also system's performance. After that, in Section 5, we discuss related work in the area and how it differs from our approach. Finally, Section 6 concludes the paper and provides an outlook to future work.

2. APPROACH

In the following sections we describe the architecture and highlight some implementation details of our approach towards a scalable social recommender engine for online marketplaces. The engine described below was implemented in Java as a joint effort with the Austrian start-up company Blanc Noir[4] and was designed in a modular way based on Apache Solr as an highly efficient data processing and storing unit.

2.1 Implementation

The overall system architecture of our framework is illustrated in Figure 1. It consists of the following four main components:

The **Recommender Engine** consists of the implemented recommender algorithms (e.g., Most Popular, Content-based, Collaborative Filtering, etc.) that can be attached to the Recommendation Workflow component. The algorithms can be called separately, in a specific sequence or combined (e.g., as a hybrid approach). This design gives our framework not only the flexibility that new algorithms can be easily implemented and instantiated (see Listing 1) but also that new recommendation workflows can be defined based on a given use case or domain.

Moreover, the Recommender Engine component contains an Evaluator that can can be used to test and tune the different algorithms (and the combinations of those) based on various evaluation metrics (see Sections 3.2 and 4.1).

The **Solr Service Container** acts as an abstraction layer for the Solr core functionalities to encapsulate the different queries and methods (e.g., facets or MoreLikeThis) into data-driven services (e.g., Marketplace or social services) that can be consumed by the recommender engine. Furthermore, this modular abstraction allows the Solr backend to be replaced by another data store or search strategy (e.g., ElasticSearch[5]) if needed.

The **Solr Cores** contain the indexed data used to generate recommendations. Each core is described by its own Solr schema that specifies a data structure and that can be extended easily by simply adding new data fields to it and calling Solr's RELOAD function. Currently, we store four types of data structures in the Solr cores: (1) user profiles, (2) item profiles, (3) user actions (e.g., purchases) and (4) social interactions (e.g., comments). New Solr cores can be added if new data structures are needed in the data model. For adding a new Solr core, a new data schema needs to be defined and registered in the solr.xml configuration file, which can be automatically done using Solr's CREATE function.

Another major reason for using Solr is its support for horizontal scaling. Since version 4.0, full automatic index distribution and searching across multiple machines (either shards or replicas) is supported. Under a scenario where the maximum capacity for handling queries per second is reached, horizontal scaling with additional replicas can be performed. On the other hand, with sharding on multiple machines, Solr supports the need to store large amounts of data distributively.

The **REST API** is the interface to our framework that a client can either use to request item recommendations, based on a specific algorithm or workflow, or to update the data model (e.g., if a user has purchased an item) via JSON messages. These JSON messages are fully configurable and let the client, for example, define user-specific data filters to, e.g., request only recommendations for a given item category.

The data updates are handled by data connectors, where each connector is responsible for a different type of data. Currently there are two connectors in the system, one for social data (e.g., Facebook, G+ and Twitter streams) and the other one for marketplace data (e.g., purchases).

Listing 1: Example of how to implement and run a new recommender strategy.

```
// Implement the recommender strategy
public interface RecommendStrategy {
   public RecommendResponse recommend(RecommendQuery q,
        Integer maxResults, SolrServer SolrServer);
}
// Run the new recommender strategy
RecommendStrategy strategyToUse = new MyStrategyImpl();
Filter filter = new ContentFilter(); // optional
RecommendationService.getRecommendations("some_user",
       "some_product", 10, filter, strategyToUse);
```

2.2 Recommender Algorithms

Currently, our framework implements four algorithms types to recommend items (in our case products) to users. This set of algorithms can easily be extended or adapted as explained in Section 2.1.

MostPopular (MP): This approach recommends for any user the same set of items, which are weighted and ranked by purchase frequency.

Collaborative Filtering (CF): Consists of recommending items to a target user that have been previously favorited, consumed or liked by similar users, the neighbors. This method is also known as K-NN because it is usually accomplished in two steps: first, find the K nearest neighbors based on some similarity metric, and second, recommend items that the neighbors have liked that the target user still has not consumed [20].

[4]http://blanc-noir.at/

[5]http://www.elasticsearch.org/

In our case, we construct the neighborhood of a user based on two types of features: marketplace features (purchases and categories), and social features (interests, groups and interactions) as shown in Table 2. As an example: In the case of purchases (CF_p), we get all purchased items of the target user and query all users that have also bought these items through the Solr data model in order to recommend their purchased items to the target user. The resultant lists of users and items are ranked and weighted using Solr's facet queries. The necessary queries for this process are the following:

```
// Find similar users based on purchased items using
     Solr's facet queries
/select?q=id:("some_product_1")+OR+id:("some_product_2")&
      facet=true&facet.field=my_users_field
// Find items purchased by those similar users that are
     new to the target user
/select?q=my_users_field:("user_1"^5+OR+"user_2"^3)&
      fq:-id:("some_product_1")+OR+-id:("some_product_2")
```

Content-based Recommendations (C): Content-based recommendation systems analyse item meta-data to identify other items that could be of interest for a specific user. This can be done based on user profile data or on the meta-data of the items that the user has liked or purchased in the past [15]. Our implementation of a content-based recommender is based on the second method and uses the built-in MoreLikeThis functionality of Solr that finds similar items for one or multiple given items by matching their content. We use two different types of meta-data features, namely the title and the description of items (see Table 2). There are several parameters for the MoreLikeThis function that can be set, e.g., the minimum document frequency (mindf), the minimum term frequency (mintf), minimum word length (minwl), etc. In the current implementation, both frequency parameters are set to 1 and the word length to 4, which gives us a good trade-off between accuracy and scalability. However, our implementation allows the application developer also to set the parameters herself, if needed. New content-based recommendation algorithms with different features can be developed by implementing the aforementioned *RecommendStrategy* Interface. The listing below shows how a content-based recommender can be called and customized in terms of the field (mlt.fl) used to match items with similar content:

```
/select?q=id:("some_product_id")&mlt=true&
 mlt.fl=description
```

Hybrid Recommendations (CCF): All three mentioned recommender algorithms have unique strengths and weaknesses, e.g., CF suffers from sparse data and cold start problems, while content-based approaches suffer from item meta-data to be utilized[4]. Hybrid recommenders combine different algorithms to tackle this issue in order to produce more robust recommendations [5]. Considering that we want to favor items recommended by more than one method, we chose to implement the hybrid approach called Cross-Source Hybrid defined in [4]:

$$W_{rec_i} = \sum_{s_j \in S} (W_{rec_i,s_j} \cdot W_{s_j}) \cdot |S_{rec_i}| \quad (1)$$

, where the combined weighting of the recommended item i, W_{rec_i}, is given by the sum of all single weightings for each recommender source W_{rec_i,s_j} multiplied by the weightings of the recommender sources W_{s_j}. Furthermore, it uses the number of recommender sources where i appears $|S_{rec_i}|$ to strongly favor items that have been identified by more than one recommender. We use this approach to combine the different features and algorithms shown in Table 2 where each recommender source can be weighted according to its impact on the given data (e.g., its Mean Average Precision value as described in Section 3.2). This hybridization approach is just one of the many approaches of how to combine different recommender strategies as described in the aforementioned work by Burke [5]. Hence, implementing the $RecommendStrategy$ interface can lead to other approaches also in terms of the feature selection process, if needed (e.g., [17]).

Marketplace (Market)	
Number of users	72,822
Number of purchases	265,274
Mean number of purchases per user	3.64
Number of products	122,360
Mean number of purchases per products	2.17
Online Social Network (Social)	
Number of users	64,500
Number of likes	1,492,028
Number of comments	347,755
Mean number of likes per user	14.91
Mean number of comments per user	3.47
Number of groups	260,137
Mean number of groups per user	8.91
Number of interests	88,371
Mean number of interests per user	1.57

Table 1: Basic statistics of the SL dataset.

3. EXPERIMENTAL SETUP

In the following sections we describe in detail the dataset and the evaluation method and metrics used for our evaluation.

3.1 Dataset

In order to evaluate our social recommender architecture, we relied on two different sources of data to predict future product purchases (see also [25]) – online Marketplace data and an online social network data obtained from the virtual world SecondLife[6] (SL). The reason for choosing SL over other real world sources is the lack of freely available datasets that combine both social network with marketplace data from the same set of users. The overall statistics of whole dataset can be found in Table 1.

Similar to eBay, every seller in the SL marketplace[7] owns her own sub-page – called the seller's store – where all items offered are presented to the general public. As with other trading platforms such as Amazon, sellers in the SL Marketplace have the possibility to apply meta-data information such as price, title, or description to their products. Customers in turn are able to provide reviews or ratings to products. In order to crawl all stores and corresponding meta-data information as well as interactions from the SL marketplace, we exploited the fact that every store has a unique URI built from the URL pattern http://marketplace.secondlife.com/stores/STORE_ID, where STORE_ID is an incremental integer starting at 1. With this exploit at hand, we were able to download 72,822 complete user profiles with corresponding 265,274 purchases from the stores.

The online social network MySecondLife[8] was introduced by Linden Labs, in July 2011. It can be compared to Facebook regarding postings and check-ins but aims only at residents of the virtual world. Hence, users can interact with each other by sharing text messages, and commenting or liking (= loving) these mes-

[6]https://secondlife.com/
[7]https://marketplace.secondlife.com/
[8]https://my.secondlife.com/

	Set	Name	Alg.	Feature	$nDCG$	MRR	MAP	F_1	D	UC
Market	CCF_m	CF_p	CF	purchases	.0812 (.0305)	.1310 (.0492)	.0628 (.0236)	.0442 (.0166)	.4404 (.1654)	37.56%
		CF_c	CF	categories	.0312 (.0145)	.0202 (.0094)	.0226 (.0105)	.0146 (.0068)	.4945 (.2298)	46.47%
		C_t	C	title	.0361 (.0168)	.0250 (.0116)	.0267 (.0124)	.0155 (.0072)	.6000 (.2789)	46.30%
		C_d	C	description	.0370 (.0172)	.0260 (.0121)	.0280 (.0130)	.0153 (.0071)	.5991 (.2785)	46.61%
Social	CF_s	CF_i	CF	interests	.0018 (.0003)	.0012 (.0002)	.0012 (.0002)	.0006 (.0001)	.3814 (.0630)	16.52%
		CF_g	CF	groups	.0257 (.0129)	.0205 (.0103)	.0215 (.0108)	.0128 (.0064)	.3816 (.1913)	50.13%
		CF_l	CF	likes	.0120 (.0010)	.0084 (.0007)	.0096 (.0008)	.0084 (.0007)	.3269 (.0273)	8.35%
		CF_{co}	CF	comments	.0112 (.0008)	.0084 (.0006)	.0096 (.0007)	.0084 (.0006)	.3147 (.0225)	7.15%
		CF_{in}	CF	interactions	.1670 (.0192)	.1174 (.0135)	.1417 (.0163)	.1626 (.0187)	.3235 (.0372)	11.50%

Table 2: Results of the performance experiment for each recommendation approach with corresponding features (normalized to the actual UC in the row). Values in brackets represent the results normalized to 100% UC.

sages. A user pro- file can be accessed through a unique URL, https://my.secondlife.com/en/USER_ID, where USER_ID depicts the user's name. The necessary names were extracted from the SL Marketplace dataset. In order to gather the whole network we also extracted all interaction partners recursively until no new user could be found. All over, 1,839,783 interactions (likes, comments) were downloaded for 64,500 user profiles.

3.2 Evaluation Method and Metrics

To evaluate the performance of our recommendation methods in terms of accuracy, ranking, diversity and coverage, we performed a number of off-line experiments. Therefore, we split the SL dataset in two different sets (training and test set) using a method similar to the described in [14], i.e. for each user we withheld 10 purchased items (= products) from the training set and added them to the test set to be predicted. Since we did not use a p-core pruning technique to prevent a biased evaluation as suggested in related work [7], there are also users with less than 10 relevant items. For these users, we considered half of their purchased items for training and the other half for testing. With that method at hand we are able to simulate cold-start users for whom there is no item in the training set and the only relevant item is used in the test set.

For the evaluation metrics we used a diverse set of well-established measures in recommender systems. In particular, we report F_1-score (F_1), Mean Reciprocal Rank (MRR), Mean Average Precision (MAP), Normalized Cumulative Discounted Gain ($nDCG$), User Coverage (UC) [13], and Diversity (D) [21]. All performance metrics are reported for 10 recommended items (k=10).

To assess the performance of our recommender framework in terms of execution time and scalability we conducted two evaluations. In the first one, we compared the runtime of train and test datasets using different approaches and data sources. In the second, we compared different stress tests within three different scenarios. In scenario (i), we report the mean response time (in seconds) according to an increasing number of requests, in scenario (ii), we report the mean response time with 10% new randomly generated data updates (i.e., purchases) during the recommendation process and in (iii) we report the mean time needed to persist data on the disc for a different number of updates. The experiments have been executed on an IBM System x3550 server with two 2.0 GHz six-core Intel Xeon E5-2620 processors, a 1TB ServeRAID M1115 SCSI Disk and 128 GB of RAM using one Apache Solr 4.3.1. instance and Ubuntu 12.04.2.

4. RESULTS

In this section we present the results of our experiments in respect to the recommender performance and the scalability of our framework.

4.1 Algorithmic Performance

The evaluation of the performance of the recommender algorithms and data sources has been conducted in two steps, first we compared the different approaches and features on their own (see Table 2) and then we compared the combinations of those (see Table 4). The results for the different algorithms are calculated related to their user coverage and so are based only on the users where they were able to calculate recommendations as suggested in related work [9, 2]. Furthermore, also the values based on all users in the datasets are shown in parenthesis.

Table 2 shows that the best results for the accuracy metrics (F_1, MRR, MAP and $nDCG$) are reached by CF_{in} followed by CF_p, the CF approaches based on social interactions and purchases. However, the results also reveal that CF_{in} only provides a small user coverage (UC) value, where CF_p performs much better and CF_g (CF based on groups) performs best. The best diversity (D) values are reached by the two content-based approaches based on title and description (C_t and C_d). Another thing that comes apparent is, that all shown approaches clearly outperform CF_i (CF based on interests). Although the user's interest seems conceptually a good metric to assess user similarity, in the SL social network it is defined by free-written keywords and phrases of the user, which would require additional validation or processing steps in order to exploit it as an efficient source of similarity.

This pattern of results also shows that the different algorithms and features have their unique strengths and weaknesses and that a hybrid combination of those should increase the overall recommender quality in terms of accuracy, diversity and user coverage [5]. Table 4 proofs this assumption and shows the combination of the marketplace-based approaches (CCF_m = CF_p + CF_c + C_t + C_d), the combination of the social based approaches (CF_s = CF_i + CF_g + CF_l + CF_{co} + CF_{in}) and the combination of both together with MP (All = CCF_m + CF_s + CF_s) to also address the issue of cold-start users. It can be seen that our hybrid approach not only outperforms the other approaches on all metrics but also provides a UC of 100% and so it can provide recommendations for all users in the datasets.

4.2 Framework Scalability

The recommender scalability has been evaluated in two ways, first we compared the runtime of the different approaches and features, as well as the hybrid combinations of those, and second we compared the mean response time of the algorithms in form of a stress test with an increasing number of requests in three scenarios.

The results of the runtime comparison are shown in Table 3. The table reveals the mean test time ($\overline{Test}$) that is needed to calculate recommendations for a user and the overall time ($|Test + Train|$) that is needed to process all the users from the test set together with the training time (711 seconds) for building the data model

Type	CF_r	CF_c	C_n	C_d	CF_i	CF_g	CF_l	CF_c	CF_{in}	MP	CCF_m	CF_s	All
$\overline{Test}$	0.020	0.097	0.029	0.094	0.024	0.023	0.011	0.013	0.021	0.016	0.194	0.024	0.197
$\|Test + Train\|$	2,167	7,775	2,823	7,556	2,459	2,386	1,513	1,658	2,240	1,876	14,838	2,459	15,057

Table 3: Results of the runtime experiment (in seconds) for each single recommendation approach and feature together with the hybrid approaches and MP as a baseline.

in Solr (i.e., indexing the data). In general these results reveal that Solr is capable of providing real-time recommendations for users as the maximum mean test time is only 0.197 seconds for our hybrid approach.

Figure 2 shows the results of the stress test with an increasing number of requests in three scenarios, first without data updates during the recommendation process, the second one is similar but includes a 10% rate of data updates (i.e., randomly generated purchases), and the third scenario shows the time needed to update data. It can be seen in the first plot (without data updates) that the mean response time follows a near linear progress for our combined hybrid approach which clearly shows the scalability of Apache Solr and our framework. Most surprisingly this is also the case in the second plot that also takes data updates during the recommendation time into account and so shows the capability of Solr in maintaining its data index in near real-time. The third plot shows that Solr is also designed to handle a high number of update requests as there is a much sharper increase in the mean update time for a small number of update requests than for a high number.

This shows that our framework based on Solr already contains algorithms that not only provide a good trade-off between recommendation accuracy, diversity and user coverage, but also provide and calculate recommendations in real-time and at scale. Furthermore, there are additional ways to optimize Apache Solr (e.g., soft commits, using an SSD disk, ...) to even better tackle the performance of committing new or existing data.

5. RELATED WORK

There are already multiple frameworks and approaches out there that focus on scalable recommendation mechanisms. Most of these approaches are based on Collaborative Filtering techniques to predict the user's ratings for items, such as movies or products, based on the user's preferences in the past. However, the computational complexity of these calculations is typically very high, especially in the case of real-time streams.

To tackle this issue, previous work focused on distributed and scalable data processing frameworks such as Apache Hadoop or Mahout based on the map/reduce paradigm (e.g., [26] or [23]). In contrast to our framework based on Apache Solr, these approaches lack the mechanisms that enable near real-time updates of the data model (data indexing) in case of new user interactions (e.g., a user purchased an item) and updates of the data schema in case of new data sources that have to be plugged in (e.g., data from HBase tables). Furthermore, it is not trivial to handle social- and content-based data with these framework, whereas this functionality comes directly out-of-the-box with Solr (e.g., with the MoreLikeThis function) together with powerful full text search functionalities. An alternative method to improve Collaborative Filtering is based on Matrix Factorization as for example proposed by Diaz-Aviles et al. [6]. However, in this work the authors focus on the near real-time processing of Twitter streams for topic recommendations and not on item recommendations in social online marketplaces as it is done with our framework.

Other approaches use database systems in order to "query" the recommendations from a data model or to simply cache the already calculated recommendations. One example for a database-driven online recommender framework is the ***RecDB*** project by Sarwat et al. [19] which is built on the basis of a PostgreSQL database with an extended SQL statement set. The authors show that RecDB can provide near real-time recommendations for movies, restaurants and research papers. Although these approaches perform fairly good, it has been shown that relational database management systems are insufficient for full text searches, that are the basis for content-based recommendations, where information retrieval software like Solr greatly speed up the response time of the requested queries [24].

Measure	MP	CCF_m	CF_s	All
$nDCG$	.0078	.0678 (.0316)	.0182 (.0103)	.0387
MRR	.0054	.0420 (.0196)	.0126 (.0071)	.0249
MAP	.0054	.0485 (.0226)	.0133 (.0075)	.0278
F_1	.0032	.0354 (.0165)	.0115 (.0065)	.0188
D	.3801	.4877 (.2274)	.3770 (.2129)	.4276
UC	100%	46.63%	56.47%	100%

Table 4: Results of the performance experiment for the hybrid approaches together with MP as a baseline (normalized to the actual UC in the row). Values in brackets represent the results normalized to 100% UC.

To date, there is only few research available that focus on the usage of search engines and information retrieval systems to implement recommendation services. In [22] a method is presented to implement a k-nearest neighbor-based recommendation system on top of full text search engines (MySQL and SphinxSearch) that provides linear scalability relative to the data size. Another work in this context is a recent contribution by Parra et al. [12] who implemented a recommender system for scientific talks based on Apache Solr. Although the latter mentioned contribution provides insights on how to implement a near real-time recommender system based on Apache Solr, they lack of extensive explanations and evaluations of how such an approach performs in a big data scenario.

6. CONCLUSIONS

In this paper we have presented the implementation details and evaluation of an online social marketplace recommender with a focus on two kind of readers: researchers and professionals in the area of recommender systems. On the research side, we provided results that highlight the importance of social features (interactions in the form of likes and comments) in order to improve the accuracy, diversity and coverage of product recommendations. From the side of professionals, we provided a description of our framework based on Apache Solr with detailed results in terms of performance and scalability in order to serve as a baseline for people interested in implementing a recommender system, information rarely found in current literature. Our framework evaluation considers dimensions such as hardware configuration, model training and testing trade-offs, real-time recommendation performance and the impact of model updates over the whole system performance.

We plan different tasks to extend our current study. In terms of algorithms, we would like to explore whether other hybridization techniques (weighted, mixed, etc.) can provide us alternative ways to combine methods and data sources, in order to produce

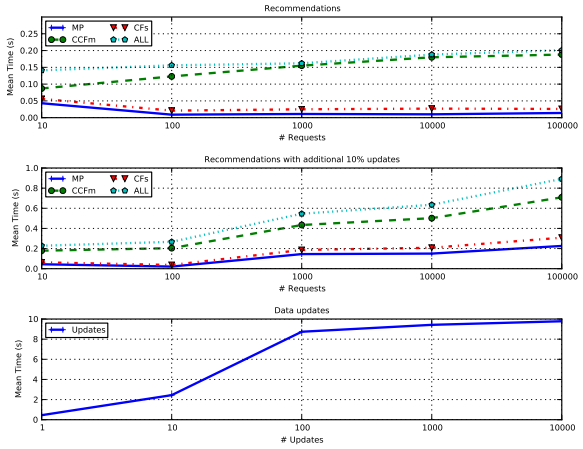


Figure 2: Scalability tests (in seconds) for 10 to 100,000 recommendation requests in three different scenarios.

an improvement in the quality of our recommendations. We also look forward to test and integrate matrix factorization techniques and study its impact in terms of recommendation quality and system scalability. We also intend to run user studies to make sure that improvements in accuracy, diversity and user coverage have a significant positive impact on user engagement and satisfaction.

Regarding the platform, our current work proofed the feasibility of only one well-known search engine backend to be easily utilized and extended as a collaborative and content filtering recommender engine. Therefore it is our aim to also investigate in depth other popular backend search solutions such as ElasticSearch to compare it with our current implementation based on Apache Solr. In this respect, we are also interested in a comparative study that investigates the performance of several collaborative and content filtering approaches grounded on SQL, Mahout or search engines (Solr, ElasticSearch). Also, we are interested in utilizing other data sources, in depth scalability experiments through sharding, and on testing different feature selection methods for recommender systems.

Acknowledgments: The authors would like to thank Michael Steurer and Lukas Eberhard for crawling the SL dataset and Alan Said and Alejandro Bellogin for value comments on the paper. This work is supported by the Know-Center. The first and the second author of this paper are supported by grants from the EU funded project Learning Layers (Nr. 318209).

7. REFERENCES

[1] M. Balabanović and Y. Shoham. Fab: content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, Mar. 1997.

[2] A. Bellogín and J. Parapar. Using graph partitioning techniques for neighbour selection in user-based collaborative filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 213–216. ACM, 2012.

[3] A. Bellogin, J. Wang, and P. Castells. Bridging memory-based collaborative filtering and text retrieval. *Information Retrieval*, 16(6):697–724, 2013.

[4] S. Bostandjiev, J. O'Donovan, and T. Höllerer. Tasteweights: a visual interactive hybrid recommender system. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 35–42. ACM, 2012.

[5] R. Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[6] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl. Real-time top-n recommendation in social streams. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 59–66. ACM, 2012.

[7] S. Doerfel and R. Jäschke. An analysis of tag-recommender evaluation procedures. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 343–346. ACM, 2013.

[8] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Mymedialite: A free recommender system library. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 305–308, New York, NY, USA, 2011. ACM.

[9] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[10] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[11] S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 1097–1101, New York, NY, USA, 2006. ACM.

[12] D. Parra, P. Brusilovsky, and C. Trattner. User controllability in an hybrid talk recommender system. In *Proceedings of the ACM 2014 International Conference on Intelligent User Interfaces*, IUI '14, pages 305–308, New York, NY, USA, 2014. ACM.

[13] D. Parra and S. Sahebi. Recommender systems : Sources of knowledge and evaluation metrics. In *Advanced Techniques in Web Intelligence-2: Web User Browsing Behaviour and Preference Analysis*, pages 149–175. Springer-Verlag, 2013.

[14] D. Parra-Santander and P. Brusilovsky. Improving collaborative filtering in social tagging systems for the recommendation of scientific articles. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 1, pages 136–142. IEEE, 2010.

[15] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[16] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[17] R. Ronen, N. Koenigstein, E. Ziklik, and N. Nice. Selecting content-based features for collaborative filtering recommenders. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 407–410. ACM, 2013.

[18] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[19] M. Sarwat, J. Avery, and M. F. Mokbel. Recdb in action: recommendation made easy in relational databases. *Proceedings of the VLDB Endowment*, 6(12):1242–1245, 2013.

[20] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[21] B. Smyth and P. McClave. Similarity vs. diversity. In D. Aha and I. Watson, editors, *Case-Based Reasoning Research and Development*, volume 2080 of *Lecture Notes in Computer Science*, pages 347–361. Springer Berlin Heidelberg, 2001.

[22] J. Suchal and P. Návrat. Full Text Search Engine as Scalable k-Nearest Neighbor Recommendation System. In M. Bramer, editor, *Artificial Intelligence in Theory and Practice III*, pages 165–173. Springer Berlin Heidelberg, 2010.

[23] S. G. Walunj and K. Sadafale. An online recommendation system for e-commerce based on apache mahout framework. In *Proceedings of the 2013 annual conference on Computers and people research*, pages 153–158. ACM, 2013.

[24] O. Yilmazel, B. Yurekli, B. Yilmazel, and A. Arslan. Relational Databases versus Information Retrieval Systems : A Case Study. *IADIS International Conference Applied Computing 2009*, pages 1–4, 2009.

[25] Y. Zhang and M. Pennacchiotti. Predicting purchase behaviors from social media. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 1521–1532, 2013.

[26] Z.-D. Zhao and M.-s. Shang. User-based collaborative-filtering recommendation algorithms on hadoop. In *Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on*, pages 478–481. IEEE, 2010.

Real-Time Recommendations in a Multi-Domain Environment

Emanuel Lacic
KTI
Graz University of Technology
Graz, Austria
elacic@know-center.at

ABSTRACT

Recommender systems are acknowledged as an essential instrument to support users in finding relevant information. However, adapting to different domain specific data models is a challenge, which many recommender frameworks neglect. Moreover, the advent of the big data era has posed the need for high scalability and real-time processing of frequent data updates, and thus, has brought new challenges for the recommender systems' research community. In this work, we show how different item, social and location data features can be utilized and supported to provide real-time recommendations. We further show how to process data updates online and capture user's real-time interest without recalculating recommendations. The presented recommendation framework provides a scalable and customizable architecture suited for providing real-time recommendations to multiple domains. We further investigate the impact of an increasing request load and show how the runtime can be decreased by scaling the framework.

Keywords

scalability; real-time recommendations; Apache Solr; multi-domain;

1. MOTIVATION

In the past decade, there has been a vast amount of research in the field of recommender systems. Most of that work focuses on developing novel approaches [22] and improving accuracy [16]. Thus, many well known methods are available, such as Content-Based Filtering [15], Collaborative Filtering [21] or Matrix Factorization [13], all having their unique strengths and weaknesses. These approaches are traditionally adapted and applied with the focus on a single domain model (e.g., marketplace, hotel, conference, etc.). However, to support a diverse set of domains is becoming an important issue for modern recommender systems [12].

In most domains, the prediction task is usually viewed as a two-dimensional problem which one needs to solve (e.g., utilizing user-item interactions). But nowadays it is not enough to support multiple domains on the basis of only one common data feature. With the arrival of the big data era, recommender systems are expected to analyze a lot of data, to support various data types and to handle streams of new data (i.e., volume, variety and velocity defining the Big Data problem). In such large-scale settings, traditional recommender systems usually analyze the data offline and update the generated model in regular time intervals. However, in many domains, choices made by users depend on factors which are susceptible to change anytime. Lets take a shopping mall for example, where a user triggers frequent indoor location updates via a smartphone application while moving through the mall. Employing an offline model update strategy that lasts hours or days may potentially miss the current location context of the user and fail to provide the right recommendations to match user's real-time demand. As a consequence, being able to capture user's real-time interests is gaining momentum and is currently of high demand [5, 16, 4].

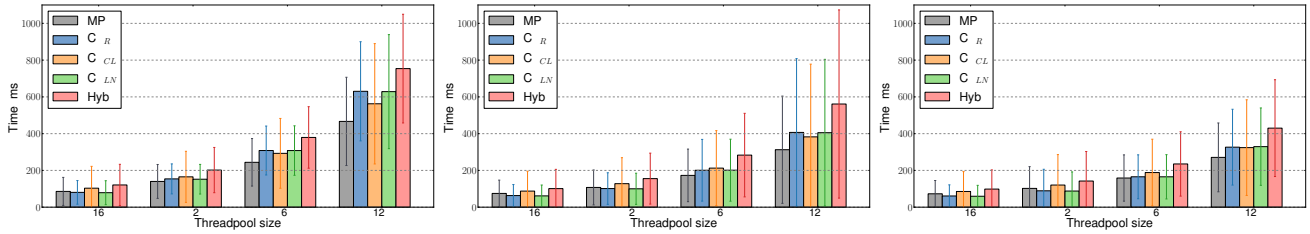
2. BACKGROUND

Most existing work, which focuses on real-time recommendations (e.g., Netflix [1], Microsoft [17], among others, e.g., [3, 23]), use offline batch processing frameworks like Apache Hadoop, Mahout or Spark. Other approaches use a relational database system to provide near real-time recommendations by querying the recommendations from a generated data model [19]. However, to capture user's real-time interest, streaming data needs to be processed online, thus needing to tackle the conflicting accuracy, real-time and big data requirements. For example, recent research from Huang et al. [5] and Chandramouli et al. [4] goes into that direction by utilizing a scalable Item-based Collaborative Filtering approach to provide real-time recommendations.

But, by focusing on the common user-item interactions, additional contextual information is usually neglected. As such, the research community has also looked into exploiting social or location data (e.g., [2, 13]). In doing so, personalized recommendations using Matrix Factorization dominate the literature. Jamali et al. [6] predicted ratings using a Matrix Factorization model that incorporates social relations. Ma et al. [13] improved both Mean Absolute Error and Root-Mean-Square Error by incorporating social information, using social regularization in two Matrix Factorization models. In general, Matrix Factorization based approaches need to be retrained, when the data changes. This tends to be time-consuming, especially in case of frequent data updates where it fails to capture user's real-time demand. Furthermore, empirical studies showed that a large number of factors are needed so that Matrix Factorization based approaches can deal with sparse data [18].

3. APPROACH AND METHODS

In this work, we are interested in finding out to what extent different data features (i.e., item, social or location) can be utilized or even be combined for real-time recommendation. To perform



(a) 1 processing node (b) 2 processing nodes (c) 4 processing nodes

Figure 1: Scalability experiment with five recommendation approaches (having the hybrid run the four approaches in parallel), making 325,005 independent recommendation requests to process. The exponentially increasing request loads (simulated by threadpools that continuously fire requests) are handled in three scenarios: (a) localy with only 1 processing node being deployed, (b) scaling the framework with 2 distributed nodes, and (c) having 4 distributed nodes to process the incoming recommendation requests.

this task, we rely on data crawled from the virtual world of SecondLife [1] and perform an extensive evaluation in terms of *nDCG* and *User Coverage* [7] of the different content- and network-based data features. The main reason for choosing SecondLife data over other sources are manifold, but mainly due to the fact that currently there are no other datasets available that comprise extensive item, social and location data of users at the same time. Building up on these results, the aim is to provide a general framework which can (1) process streaming data online while providing real-time recommendations, (2) support a multi-domain environment and the corresponding data features, and (3) provide a scalable architecture to cope with increasing request loads.

Recently, search engines have gained attention in the context of recommender systems [14]. While the results are promising, they do not provide explanations and evaluations of how such an approach would perform in a big data, nor in a real-time multi-domain environment. As such, the aim of this work is to proof the benefits of using search engines to support different data features while providing real-time recommendations. One issue, however, is that in this way scalability problems are only tackled on the data side of the domain. In order to truly be able to support multiple domains, a recommender framework is needed which can additionally (1) be customized with domain specific models and approaches, and (2) cope with an increasing request load a domain could experience. Using the already mentioned SecondLife dataset, but also a much larger Foursquare dataset [20], we simulate an increasing recommendation request load which such a framework needs to handle.

4. OUTCOMES

In [7] we showed to what extent different data features (derived from item, social and location data) can be utilized for recommending items, low-level and top-level categories. In our results, we showed that approaches which utilize social data features can outperform the ones based on item or location features in case of recommending items. In case of recommending categories, these differences get substantially smaller and even change in favor to item and location data. Moreover, our results suggests that combining the data sources should result into more robust recommendations, especially in cases of recommendation tasks on different levels of specialization (i.e., categories). In a similar fashion, we also showed in [10] that location data can especially be helpful in tackling cold-start users which have no interaction data whatsoever.

In [8], we proofed the benefits of using the search engine Apache Solr [2] to provide real-time recommendations. We showed that a recommender system is able to process data updates in real-time and immediately consider these updates (i.e., user's real-time interest) in the recommendation process without the need for recalculations. In [9] we also presented the first open-source recommender framework based on the Apache Solr search engine. But as previously mentioned, we considered the scalability issues only on the data side of a domain, and not within the framework (e.g., handling an increased request rate). For that purpose, we recently presented *ScaR* [11]. *ScaR* adopts the microservices architecture and was built with the focus on providing a scalable and customizable architecture suited for providing real-time recommendations to multiple domains. Different domains can run (and scale) the framework in isolated environments. The domain specific data features and recommendation approaches can be dynamically customized using a dedicated microservice which synchronizes the change to all domain-relevant nodes.

To demonstrate *ScaR's* scalability performance, Figure 1 reports a runtime experiment on the Foursquare dataset with an increasing number of request loads. As described in [11], we requested five different recommendation approaches for 65,001 users, making it 325,005 independent recommendations requests to process. We performed this experiment by simulating an increasing request rate (load) to the system, having 16, 32, 64 and 128 threads simultaneously requesting recommendations. These experiments were then repeated three times: first having 1 local processing node and then, scaling it to 2 and 4 distributed nodes. As seen, the local deployment has an exponential increase in the runtime as the load grows. Such behaviour is somewhat expected as the number of incoming recommendation requests cause a load spike and the processing threads consequently cause to much context switching. But, as we deploy additional nodes, we can see a significant decrease in the growth of the mean processing runtime when compared to the local deployment, which is crucial in cases when a maximal runtime needs to be guarantied.

5. PLAN AND TIMELINE

With respect to the future research workplan, the aim is to further look into feasible strategies to balance the trade-off between accuracy and runtime in a multi-domain environment. For a thesis conclusion, the idea is to find out how recent the utilized history data and the candidate recommendations need to be (i.e., by considering the exact time or a sliding window approach) in order to even better recommend user's real-time interest. This would not only lead to better accuracy but also to a better performance, as less data will need to be processed.

[1] http://secondlife.com/
[2] http://lucene.apache.org/solr/

6. REFERENCES

[1] X. Amatriain. Big & personal: Data and models behind netflix recommendations. In *Proc. of BigMine '13*.

[2] K. Bischoff. We love rock 'n' roll: Analyzing and predicting friendship links in last.fm. In *Proceedings of the 4th Annual ACM Web Science Conference*, WebSci '12, pages 47–56. ACM, 2012.

[3] S. Chan, T. Stone, K. P. Szeto, and K. H. Chan. Predictionio: a distributed machine learning server for practical software development. In *Proc. of CIKM '13*.

[4] B. Chandramouli, J. J. Levandoski, A. Eldawy, and M. F. Mokbel. Streamrec: A real-time recommender system. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 1243–1246, 2011.

[5] Y. Huang, B. Cui, W. Zhang, J. Jiang, and Y. Xu. Tencentrec: Real-time stream recommendation in practice. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 227–238, 2015.

[6] M. Jamali and M. Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, pages 135–142. ACM, 2010.

[7] E. Lacic, D. Kowald, L. Eberhard, C. Trattner, D. Parra, and L. Marinho. Utilizing online social network and location-based data to recommend products and categories in online marketplaces. In *Mining, Modeling, and Recommending 'Things' in Social Media*, pages 96–115. Springer, 2015.

[8] E. Lacic, D. Kowald, D. Parra, M. Kahr, and C. Trattner. Towards a scalable social recommender engine for online marketplaces: The case of apache solr. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*, WWW Companion '14, pages 817–822. International World Wide Web Conferences Steering Committee, 2014.

[9] E. Lacic, D. Kowald, and C. Trattner. Socrecm: A scalable social recommender engine for online marketplaces. In *Proceedings of the 25th ACM Conference on Hypertext and Social Media*, HT '14, pages 308–310, 2014.

[10] E. Lacic, D. Kowald, M. Traub, G. Luzhnica, J. Simon, and E. Lex. Tackling cold-start users in recommender systems with indoor positioning systems.

[11] E. Lacic, M. Traub, D. Kowald, and E. Lex. Scar: Towards a real-time recommender framework following the microservices architecture.

[12] Q. Liu and D. R. Karger. Kibitz: End-to-end recommendation system builder. In *Proceedings of the 9th ACM Conference on Recommender Systems*. ACM, 2015.

[13] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 287–296. ACM, 2011.

[14] D. Parra, P. Brusilovsky, and C. Trattner. User controllability in an hybrid talk recommender system. In *Proceedings of the ACM 2014 International Conference on Intelligent User Interfaces*, IUI '14, pages 305–308. ACM, 2014.

[15] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[16] C. Rana and S. K. Jain. A study of the dynamic features of recommender systems. *Artificial Intelligence Review*, 43(1):141–153, 2015.

[17] R. Ronen, N. Koenigstein, E. Ziklik, M. Sitruk, R. Yaari, and N. Haiby-Weiss. Sage: Recommender engine as a cloud service. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 475–476, 2013.

[18] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 880–887. ACM, 2008.

[19] M. Sarwat, J. Avery, and M. F. Mokbel. Recdb in action: Recommendation made easy in relational databases. *Proc. VLDB Endow.*, 6(12):1242–1245, Aug. 2013.

[20] M. Sarwat, J. J. Levandoski, A. Eldawy, and M. F. Mokbel. Lars*: An efficient and scalable location-aware recommender system. *IEEE Trans. on Knowl. and Data Eng.*, 26(6):1384–1399, June 2014.

[21] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. The adaptive web. chapter Collaborative Filtering Recommender Systems, pages 291–324. Springer-Verlag, 2007.

[22] G. Shani and A. Gunawardana. Evaluating recommendation systems. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 257–297. Springer US, 2011.

[23] S. G. Walunj and K. Sadafale. An online recommendation system for e-commerce based on apache mahout framework. In *Proceedings of the 2013 annual conference on Computers and people research*, pages 153–158. ACM, 2013.

Should we Embed? A Study on the Online Performance of Utilizing Embeddings for Real-Time Job Recommendations

Emanuel Lacic*
Know-Center GmbH
Graz, Austria
elacic@know-center.at

Markus Reiter-Haas*
Moshbit GmbH
Graz, Austria
markus.reiter-haas@moshbit.com

Tomislav Duricic
Graz University of Technology
Graz, Austria
tduricic@know-center.at

Valentin Slawicek
Moshbit GmbH
Graz, Austria
valentin.slawicek@moshbit.com

Elisabeth Lex
Graz University of Technology
Graz, Austria
elisabeth.lex@tugraz.at

ABSTRACT

In this work, we present the findings of an online study, where we explore the impact of utilizing embeddings to recommend job postings under real-time constraints. On the Austrian job platform Studo Jobs, we evaluate two popular recommendation scenarios: (i) providing similar jobs and, (ii) personalizing the job postings that are shown on the homepage. Our results show that for recommending similar jobs, we achieve the best online performance in terms of Click-Through Rate when we employ embeddings based on the most recent interaction. To personalize the job postings shown on a user's homepage, however, combining embeddings based on the frequency and recency with which a user interacts with job postings results in the best online performance.

CCS CONCEPTS

• **Information systems** → **Recommender systems**.

KEYWORDS

Job Recommendations; Online Evaluation; Real-time; Item Embeddings; Frequency; Recency; BLL Equation;

ACM Reference Format:
Emanuel Lacic, Markus Reiter-Haas*, 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 *Thirteenth ACM Conference on Recommender Systems (RecSys '19), September 16–20, 2019, Copenhagen, Denmark.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3298689.3346989

1 INTRODUCTION

Job recommender systems have become an integral part of both academia and industry for a few decades now [24], which is also illustrated by the fact that XING[1] has organized two recent RecSys Challenges [1, 2]. In the past, research on job recommendations has mainly employed various Collaborative- and Content-Based Filtering approaches or their hybrid combinations [3, 27] to improve the recommendation accuracy. Recently, learning latent item representations (i.e., embeddings) for recommender systems has become a popular technique and has shown state-of-the-art performance in the job domain. For example, the authors of [26] use Doc2Vec [17] to create job embeddings based on job-related content features. To test their approach, they conduct an offline evaluation, where they manually score the quality of similar jobs from a small subset of 100 randomly selected jobs. [23, 28] further propose an ability-aware neural network to match the content of resumes with the content of job requirements. Other works [5, 16, 22] define the task at hand as an item-to-item recommendation problem and evaluate embedding approaches also in offline studies.

However, whether a user indeed accepts a recommendation can only be measured with either user studies or online evaluations. User studies are to date rarely used as they require active participation of users over a period of time [6] and online evaluations are expensive to set up, as they need a fully functional system with a significant userbase [7]. As a consequence, related work that reports on online studies of job recommender systems is scarce. For example, recent work [12] explored in an online study how to increase the engagement toward underserved jobs. Besides, in the case of the RecSys 2017 Challenge, the top-25 participating teams were allowed to publish their solutions once per day to be rolled out on the XING platform [2]. In line with recent research [8, 11], recommendation approaches used in an online evaluation usually need to consider real-time constraints [9, 25], such as having response times, which are below 100-200 milliseconds or immediately considering data updates in the next recommendation request.

The present work. In this work, we contribute to the sparse line of research on evaluating job embeddings under real-time constraints in an online setting. For this, we learn job embeddings using the popular Doc2Vec approach. We obtain fixed-length vectors from the job description text and investigate their impact on the online performance of recommending job postings in real-time. Similarly, as in our previous work [16], we further represent a user's browsing behavior by combining the extracted embeddings using a model

*Both authors contributed equally to this work.

RecSys '19, September 16–20, 2019, Copenhagen, Denmark

ACM ISBN 978-1-4503-6243-6/19/09…$15.00
https://doi.org/10.1145/3298689.3346989

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

	Test	Distinct Users	Reco Requests	Days	Approach	CTR	↗	Runtime (ms)	↘
Similar Jobs	Impact of embeddings	8,576	31,968	32	CBF	0.0194	18.04%	51	23.53%
					LAST	**0.0229***		**39****	
	Influence of frequency and recency	4,715	18,464	15	LAST	**0.0249****	75.35%	**67****	28.72%
					BLL	0.0142		94	
	Merit of recency	3,375	11,992	15	$BLL_{d=0.6}$	**0.0174***	35.94%	97	2.06%
					$BLL_{d=0.4}$	0.0128		**95**	
Homepage	Influence of frequency and recency	9,620	26,334	25	BLL	**0.0671***	15.69%	**114****	13.64%
					CF	0.0580		132	
	Combining frequency and recency	9,313	24,907	19	HYB_{BLL}	**0.0471****	33.05%	172	38.37%
					CF	0.0354		**106****	

Table 1: We report the mean Click-Through Rate (CTR) and the mean Runtime of the approaches utilized in the corresponding A/B tests. The increase ↗ in accuracy and decrease ↘ in runtime is reported for the best performing approach. Moreover, we use the $*$ symbol to indicate it the results are significantly better with a p-value < 0.05 and the $**$ for results with a p-value < 0.0005.

from human memory theory, that integrates factors of frequency and recency of job posting interactions. To measure the real impact of such an approach, we perform several A/B tests on the Studo Jobs platform. That is, we compare against two popular recommendation use-cases that we tackle under the real-time constraint: (i) providing content-based recommendations for similar job postings to the one currently viewed by the user and, (ii) personalizing the homepage with job postings using collaborative filtering. Our findings suggest that in situations when we recommend similar job postings, using embeddings based on the most recent interaction tends to improve the online performance. In contrast, combining embeddings based on the frequency and recency with which a user interacts with job postings improves the online performance when we personalize the job postings on the homepage.

2 RECOMMENDATION STUDY

Our study is carried out in the Studo Jobs[2] platform. We tackle two distinguished recommendation scenarios, which the platform supports. First, we recommend similar jobs. Second, we personalize the ranked list of all possible job postings in the system to improve engagement on the homepage of Studo.

As shown in our previous offline study in [16], learning embeddings on the textual description of job postings can improve both the accuracy and diversity of content-based recommendations. In the present work, we learn embeddings of job postings by utilizing Doc2Vec [17], a variation of the widely popular Word2Vec [19] approach. In order to investigate the online performance of the extracted job embeddings [3] under real-time constraints, we employ two variants for performing content-based recommendation of job postings, which are described in this section. For evaluation, we measure both Click Through Rate (CTR) and runtime.

Utilizing the most recent job interaction (LAST). A natural way of using embeddings is to apply them in a content-based manner (e.g., [20]). That is, given a reference vector representation, the task is to find the top-k similar vectors (i.e., job postings) using the Cosine similarity. As described in [16], to obtain this reference vector, we use the embedding of the last (i.e., most recent) job posting with which the user has interacted. With this recommendation strategy, we can study the online performance when we recommend jobs that are similar to the one the user is currently viewing.

Integrating interaction frequency and recency (BLL). One issue of the previously mentioned LAST recommendation strategy is that it solely focuses on the factors of interaction recency. However, related work has shown that past interaction frequency and recency are crucial factors for personalization [13, 16]. In this respect, the cognitive architecture ACT-R defines the Base-Level Learning (BLL) equation, which integrates these two factors to model the information access in human memory. Thus, to simultaneously account for both frequency and recency factors of job posting interactions, we use the BLL equation to model a user's browsing behavior:

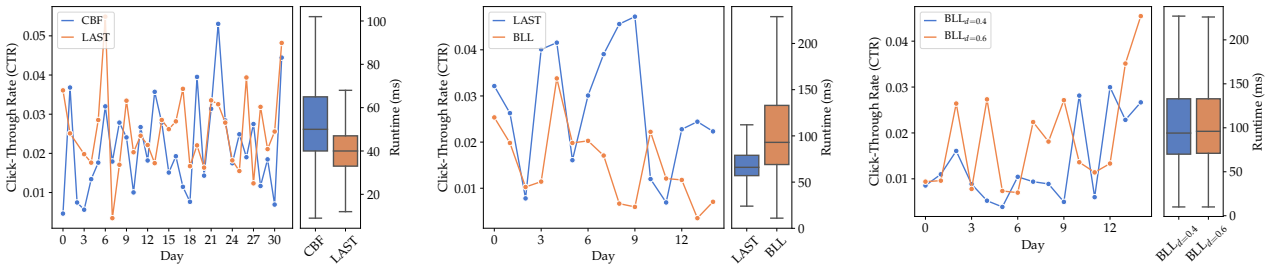
$$BLL_{u,j} = \ln(\sum_{i=1}^{n} (TS_{ref} - TS_{j,i})^{-d}) \quad (1)$$

where $BLL_{u,j}$ is the BLL value for a given user u and a given job j, and n states the number of times u has interacted with j in the past. Moreover, $TS_{j,i}$ is the timestamp (in seconds) of when u has interacted with j for the i-th time and TS_{ref} is a reference timestamp such as the time when the job recommendations are requested. The parameter d is used to set the time-dependent decay of item exposure in human memory and unless stated otherwise, we set it to its default value of 0.5 (i.e., according to Anderson et al. [4]).

In this work, we use Anderson's model of human memory theory to create a reference vector representation that can be used in a content-based manner (i.e., to find similar job postings). For that, we first normalize the BLL values using a softmax function and then multiply them with the vector representations assigned to the individual job postings from a user's browsing history. This way, we form a weighted sum of embeddings based on how frequently and recently the user has interacted with the particular job postings. As shown in [16], utilizing embeddings in such a way results in a recommendation performance with a higher diversity when compared to the previously mentioned LAST approach.

Adapting for real-time job recommendations. In practice, response times of recommendations need to be below 100-200 milliseconds [9, 25]. To adapt the LAST and BLL approaches for an online setting (i.e., to provide recommendations in real-time), we further propose to store the learned job embeddings in the form of payloads in Apache Lucene. Payloads are a general purpose array of bytes that are associated with a Lucene token at a particular position. Each

[2] https://studo.co/jobs , the predecessor of the Talto career platform (https://talto.com)
[3] We obtain the embeddings using a Doc2Vec model that we train with a window size of 20, a learning rate of 0.025 and 10 negative samples.



(a) Impact of embeddings (b) Influence of frequency and recency (c) Merit of recency

Figure 1: Analysis of incorporating job embeddings to recommend similar jobs. The reported results show a daily CTR and the distribution of the measured runtime performance.

job posting is thus annotated with multiple positions of the latent vector dimensions. The latter positional information can be used for fast retrieval and calculation of vector similarities (i.e., utilizing the Cosine similarity) at runtime[4]. For the online study on the Studo Jobs platform, we use Apache Solr[5] to store, retrieve and calculate the similarity of job embeddings in real-time.

Experimental setup. In our experiments, we measure the Click-Through Rate (CTR) and the runtime performance. To obtain the CTR, we compute the percentage of recommended job postings with which the users have interacted. For the runtime analysis, we measure the time it takes to generate each recommendation in milliseconds. We compare two approaches at a time (i.e., conduct an A/B test) to avoid being subject to periodical changes and other anomalies (e.g., server load differences). For this reason, we divide our userbase into two equal groups and assign them to one of the two approaches that we evaluate. We further perform a chi-squared test on the measured recommendation outcome (i.e., a user either did or did not engage with a recommendation) and a t-test on the runtime performance to determine if the differences in the reported results are statistically significant. Concerning real-time constraints, all recommendation approaches in the Studo Jobs platform calculate new recommendations for every request and filter out those job postings that the user has already interacted with in the current session.

3 SIMILAR JOBS

When users view a particular job posting in the Studo Jobs platform, recommendations with similar, alternative jobs are shown to them. The location of the shown recommendations depends on the layout of the device used. On the desktop, the recommendations appear in the sidebar, while on a mobile device they will appear under the job posting description. Furthermore, this type of recommendation only suggests a short list of 3 alternative job postings and does not suffer from a cold-start problem (i.e., users have at least one interaction).

Baseline: Content-Based Filtering (CBF). A popular method in many systems for recommending similar items (i.e., jobs) is Content-Based Filtering [3, 21]. This method analyses item metadata to identify other items that could be of interest for a specific user. In Studo, this is done using TF-IDF on the description text of the job posting with which the user currently interacts. Besides being a typical pick for recommending similar items, another reason for using CBF is that it can easily be adapted for an online setting, where recommendations need to be served in real-time[6].

Impact of embeddings. The initial aim of this work is to investigate if utilizing embeddings, which we learn from the textual content of job postings, can outperform traditional content-based recommendations when used in a similar item scenario. For this, we first did a preliminary A/B test of the LAST approach to evaluate the impact of the embedding size. In the case of Studo, embeddings larger than 100 did not contribute to a higher CTR, but did increase the overall runtime performance. Table 1 thus reports on all A/B tests, for which, we use 100 as the dimension size of job embeddings.

In Figure 1a, we report the performance of the LAST approach when compared to the CBF baseline. The CTR varies over the 32-day testing period, but overall, using job embedding from the currently viewed job posting leads to a significant increase of the CTR by 18.04%. Moreover, utilizing embeddings in such a way resulted in a 23.53% lower runtime (i.e., as reported in Table 1), which is a desirable effect when providing recommendations in real-time.

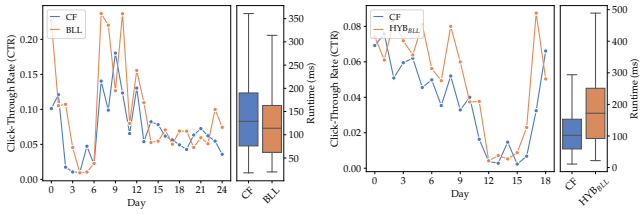
Influence of frequency and recency. Building upon the insights on the impact of using embeddings, we evaluate the model of human memory theory during a shorter, 15-day period. That is, we investigate if we can further enhance the recommendation performance by using the proposed BLL equation from Section 2 to create the reference job vector. Interestingly enough, Figure 1b clearly shows that modeling a user's browsing behavior in this manner did not result in better performance than the LAST approach in terms of both CTR and runtime. As seen in Table 1, we get the highest relative difference in CTR which did not justify the increased computational overhead, that resulted in higher runtime performance.

Merit of recency. We hypothesize that the BLL approach did not exhibit a better performance due to the specific recommendation scenario where it was applied in (i.e., showing similar jobs to the currently viewed one). This suggests that factors of recency are

[4] An example of how to implement the retrieval of similar vectors in Elastic Search, a search engine that is built on top of Apache Lucene can be found at the following link: https://github.com/lior-k/fast-elasticsearch-vector-scoring.

[5] A search engine that, similar as Elastic Search, is built on top of Apache Lucene: https://lucene.apache.org/solr/.

[6] As shown in [15], we leverage the built-in functionality of the Apache Solr search engine to recommend jobs with the most similar textual content.



(a) Influence of frequency and recency **(b) Combining frequency and recency**

Figure 2: Online performance of job embeddings when used to personalize the homepage.

especially influential in this setting and as such, we perform an additional experiment to confirm this effect.

In our previous experiment, we set the time parameter d from the BLL equation to have the default value of 0.5. However, this parameter changes the rate at which things will be "forgotten". Thus, it controls the decay of the impact of consumed items at an exponential rate. The question is therefore on whether a shorter memory (i.e., higher time decay) or a long memory is better for the setting of recommending similar job postings. For this experiment, the exponents $d = 0.6$ (shorter memory) and $d = 0.4$ (longer memory) were compared. As seen in Figure 1c and Table 1, favoring shorter memory (i.e., recency) when calculating the BLL equation resulted in a significantly better CTR. Indeed, this confirms the described effect where users expect recommendations which are similar to the more recent browsing behavior.

4 HOMEPAGE

As in many other systems, personalization in the Studo Jobs platform starts already on the homepage. The homepage consists of a list of 25 job postings from which the first 5 are the calculated recommendations. The advantage in this setting is the seamless integration of recommendations with the list of available jobs. Moreover, users not only first visit the homepage when they access the Studo Jobs portal, but also often come back to it after they stop exploring a given job posting. Such behavior results in the homepage being responsible for more than 80% of all recommendations that the user has interacted with and thus suggests to be a better fit for applying the model of human memory theory to represent the user's browsing behavior.

Baseline: Collaborative Filtering (CF). To this day, one of the most explored and utilized techniques for personalizing a system in real-time is Collaborative Filtering [10, 15, 18]. The Studo Jobs portal uses the User-Based Collaborative Filtering approach to personalize the job postings on the homepage. In that setting, a target user will get those job postings recommended that have been previously interacted by similar users (i.e., the neighbors). As shown in [14], to provide recommendations in real-time, the inverted-index structure available in the Apache Solr search engine is used to find the k-nearest neighbors using the Cosine similarity metric.

Influence of frequency and recency. As previously stated, we hypothesize that by incorporating the factors of frequency and recency from a user's browsing behavior, we can further enhance the online performance of recommendations on the homepage. For this, we use the BLL equation on the extracted embeddings from the user's interaction history to create a reference vector representation and recommend the top-k similar job postings. To account for cold-start users, we utilize the most popular job postings as a fallback[7].

As seen in Figure 2a and the second part of Table 1, using the BLL equation on embeddings from the user's job history manages to significantly outperform the CF baseline for both, the CTR and the runtime performance. Such results suggest that the scenario of personalizing the homepage is indeed a setting where the user expects the recommendations to consider both, factors of frequency and recency of her browsing behavior.

Combining frequency and recency. Instead of replacing the CF baseline with the BLL approach, we further explore the efficacy of a hybrid combination which uses these two approaches in a round-robin fashion. We assume that for the homepage, where the user interacts most with the provided recommendations, it would make sense to allow picking from multiple sources of relevant job postings since such a recommendation strategy has often lead to the best performance in offline evaluation settings (e.g., [16, 27]).

As seen in Figure 2b and the last row of Table 1, the hybrid combination of the BLL approach and the CF baseline also performs significantly better than the CF baseline concerning CTR. The relative improvement of 33.05% for the CTR in this A/B test is much better than in the case when we just used the BLL approach on its own. This, however, comes with a trade-off, namely, with a significant increase in the runtime.

5 CONCLUSION

In this work, we contributed to the sparse line of research on evaluating job embeddings under real-time constraints in an online setting. We performed a variety of A/B tests on the Studo Jobs platform and ran evaluations concerning CTR and runtime for two different recommendation scenarios, namely, recommending similar jobs and personalizing the job postings that are shown on the homepage.

We found that for the case of recommending similar jobs, using embeddings based on the most recent interaction provides the best online performance. In contrast, combining embeddings based on the frequency and recency with which a user interacts with job postings significantly improves the online performance when we personalize the jobs on the homepage.

Limitations and Future Work. While Doc2Vec is a popular choice for learning item embeddings, other deep learning methods such as, e.g., Autoencoders or Convolutional Neural Networks might also perform well for this task. Furthermore, we did not explore the impact of using additional user or job-related metadata on the quality of learned embeddings. We also did not study the effects of the time-dependent decay parameter d from the model of human memory theory to a greater extent for personalizing the jobs shown on the homepage. As such, we aim to tackle these points as future work. Finally, the data we used for this study is proprietary and we currently cannot release it to the research community.

Acknowledgements. This work was supported by the FFG Data Market Austria (DMA) project. The authors also thank the team of the Talto career platform for supporting this research experiment.

[7] Such a fallback strategy is used for every recommendation approach on the homepage of the Studo Jobs portal (including the CF baseline).

REFERENCES

[1] F. Abel, A. Benczúr, D. Kohlsdorf, M. Larson, and R. Pálovics. Recsys challenge 2016: Job recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 425–426. ACM, 2016.

[2] F. Abel, Y. Deldjoo, M. Elahi, and D. Kohlsdorf. Recsys challenge 2017: Offline and online evaluation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 372–373. ACM, 2017.

[3] S. T. Al-Otaibi and M. Ykhlef. A survey of job recommender systems. *International Journal of Physical Sciences*, 7(29):5127–5142, 2012.

[4] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of the mind. *Psychological review*, 111(4):1036, 2004.

[5] O. Barkan and N. Koenigstein. Item2vec: neural item embedding for collaborative filtering. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2016.

[6] J. Beel, M. Genzmehr, S. Langer, A. Nürnberger, and B. Gipp. A comparative analysis of offline and online evaluations and discussion of research paper recommender system evaluation. In *Proceedings of the international workshop on reproducibility and replication in recommender systems evaluation*, pages 7–14. ACM, 2013.

[7] P. G. Campos, F. Díez, and I. Cantador. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1-2):67–119, 2014.

[8] B. Chandramouli, J. J. Levandoski, A. Eldawy, and M. F. Mokbel. Streamrec: a real-time recommender system. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1243–1246. ACM, 2011.

[9] C. Eksombatchai, P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1775–1784. International World Wide Web Conferences Steering Committee, 2018.

[10] T. George and S. Merugu. A scalable collaborative filtering framework based on co-clustering. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 4–pp. IEEE, 2005.

[11] Y. Huang, B. Cui, W. Zhang, J. Jiang, and Y. Xu. Tencentrec: Real-time stream recommendation in practice. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 227–238. ACM, 2015.

[12] K. Kenthapadi, B. Le, and G. Venkataraman. Personalized job recommendation system at linkedin: Practical challenges and lessons learned. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 346–347. ACM, 2017.

[13] D. Kowald, S. C. Pujari, and E. Lex. Temporal effects on hashtag reuse in twitter: A cognitive-inspired hashtag recommendation approach. In *Proceedings of the 26th International Conference on World Wide Web*, pages 346–347, 2017.

[14] E. Lacic, D. Kowald, and E. Lex. Neighborhood troubles: On the value of user pre-filtering to speed up and enhance recommendations. *In Proceedings of the International Workshop on Entity Retrieval (EYRE'2018) co-located with CIKM'18*, 2018.

[15] E. Lacic, D. Kowald, D. Parra, M. Kahr, and C. Trattner. Towards a scalable social recommender engine for online marketplaces: The case of apache solr. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 817–822. ACM, 2014.

[16] E. Lacic, D. Kowald, M. Reiter-Haas, V. Slawicek, and E. Lex. Beyond accuracy optimization: On the value of item embeddings for student job recommendations. *In Proceedings of the Workshop on Multi-dimensional Information Fusion for User Modeling and Personalization (IFUP'2018) co-located with WSDM'18*, 2018.

[17] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning (ICML'14)*, pages 1188–1196, 2014.

[18] G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, (1):76–80, 2003.

[19] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[20] C. Musto, G. Semeraro, M. de Gemmis, and P. Lops. Learning word embeddings from wikipedia for content-based recommender systems. In *European Conference on Information Retrieval*, pages 729–734. Springer, 2016.

[21] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[22] V.-T. Phi, L. Chen, and Y. Hirate. Distributed representation based recommender systems in e-commerce. In *DEIM Forum*, 2016.

[23] C. Qin, H. Zhu, T. Xu, C. Zhu, L. Jiang, E. Chen, and H. Xiong. 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*, pages 25–34. ACM, 2018.

[24] R. Rafter, K. Bradley, and B. Smyth. Personalised retrieval for online recruitment services. In *The BCS/IRSG 22nd Annual Colloquium on Information Retrieval (IRSG 2000), Cambridge, UK, 5-7 April, 2000*, 2000.

[25] A. Said, J. Lin, A. Bellogín, and A. de Vries. A month in the life of a production news recommender system. In *Proceedings of the 2013 workshop on Living labs for information retrieval evaluation*, pages 7–10. ACM, 2013.

[26] J. Yuan, W. Shalaby, M. Korayem, D. Lin, K. AlJadda, and J. Luo. Solving cold-start problem in large-scale recommendation engines: A deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1901–1910. IEEE, 2016.

[27] C. Zhang and X. Cheng. An ensemble method for job recommender systems. In *Proceedings of the Recommender Systems Challenge*, RecSys Challenge '16, pages 2:1–2:4. ACM, 2016.

[28] C. Zhu, H. Zhu, H. Xiong, C. Ma, F. Xie, P. Ding, and P. Li. Person-job fit: Adapting the right talent for the right job with joint representation learning. *ACM Transactions on Management Information Systems (TMIS)*, 9(3):12, 2018.

User Modeling and User-Adapted Interaction (2020) 30:617–658
https://doi.org/10.1007/s11257-020-09269-1

Using autoencoders for session-based job recommendations

Emanuel Lacic[1] · Markus Reiter-Haas[2] · Dominik Kowald[1] · Manoj Reddy Dareddy[3] · Junghoo Cho[3] · Elisabeth Lex[4]

Received: 15 April 2019 / Accepted in revised form: 1 June 2020 / Published online: 1 July 2020

Abstract

In this work, we address the problem of providing job recommendations in an online session setting, in which we do not have full user histories. We propose a recommendation approach, which uses different autoencoder architectures to encode sessions from the job domain. The inferred latent session representations are then used in a k-nearest neighbor manner to recommend jobs within a session. We evaluate our approach on three datasets, (1) a proprietary dataset we gathered from the Austrian student job portal Studo Jobs, (2) a dataset released by XING after the RecSys 2017 Challenge and (3) anonymized job applications released by CareerBuilder in 2012. Our results show that autoencoders provide relevant job recommendations as well as maintain a high coverage and, at the same time, can outperform state-of-the-art session-based recommendation techniques in terms of system-based and session-based novelty.

Keywords Job recommendations · Session-based recommendation · Autoencoders · Session embeddings · Accuracy · Novelty

1 Introduction

People increasingly use business-oriented social networks such as LinkedIn[1] or XING[2] to attract recruiters and to look for jobs (Kenthapadi et al. 2017). Users of such networks make an effort to create personal profiles that best describe their skills, interests, and previous work experience. Even with such carefully structured content, it remains a non-trivial task to find relevant jobs (Abel 2015). As a consequence, the field of job recommender systems has gained much traction in academia

[1] http://linkedin.com.
[2] http://xing.com.

✉ Elisabeth Lex
elisabeth.lex@tugraz.at

Extended author information available on the last page of the article

and the industry (Lacic et al. 2019; Siting et al. 2012). The main challenge that job recommender systems tackle is to retrieve a list of jobs for a user based on her preferences or to generate a list of potential candidates for recruiters based on the job's requirements (Hong et al. 2013).

Besides, most online job portals offer the option to browse the available jobs anonymously in order to attract users to the portal. As a consequence, the only data a recommender system can exploit are anonymous user interactions with job postings during a session. In other words, the problem of recommending jobs is a session-based recommendation problem (Jannach and Ludewig 2017). That is, the aim is to recommend the next relevant job in an anonymous session.

In our ongoing work with the Austrian start-up Studo,[3] we have started to address the problem of recommending jobs in a session-based environment. In their student job portal Studo Jobs,[4] we have observed an increasing volume of anonymous user sessions that look for new jobs.[5] For example, over the past six months, anonymous job-related browsing has doubled from approximately 30,000 to 60,000 job interactions. Therefore, in this paper, we address the problem of recommending jobs in a session-based environment.

Recently, neural networks have gained attention in the context of session-based recommender systems (e.g., Hidasi et al. 2015; Li et al. 2017; Lin et al. 2018; Wu et al. 2018, 2019; Yuan et al. 2019). The idea is to extract latent information about a user's preferences from anonymous, short-lived sessions. For example, autoencoders (Kramer 1991) are neural networks designed to learn meaningful representations, i.e., embeddings, and to reduce the dimensionality of input data. Example applications are data compression (Theis et al. 2017), clustering and dimensionality reduction (Makhzani et al. 2015) as well as recommender systems, where they have been used to find latent similarities between users and items and to predict user preferences (Sedhain et al. 2015; Strub et al. 2016).

Their ability to preserve the most relevant features, while reducing dimensionality, inspired our idea to explore the use of autoencoders to infer latent session representations in the form of embeddings and to use these embeddings to generate recommendations in a k-nearest-neighbor manner. To that end, in this paper, we introduce a recommendation approach, which employs different autoencoder architectures, (1) a classic autoencoder (Kramer 1991), (2) a denoising autoencoder (Vincent et al. 2008) and (3) a variational autoencoder (Jordan et al. 1999), to learn embeddings of job browsing sessions. The inferred latent session representations are then used in a k-nearest neighbor manner to recommend jobs within a session. Besides, we use two types of input data to train and test our approach, i.e., interaction data from sessions and content features of job postings, for which interactions took place during a session. We assess the performance of our approach in the form

[3] https://studo.co.

[4] The jobs platform in Studo, which is the predecessor of the Talto career platform (https://talto.com)

[5] We observe this trend independent from the changes in authenticated sessions, which fluctuate heavily over the year. The cause of this trend is that both the total number of sessions and the average ratio of anonymous sessions to authenticated sessions are growing.

of offline evaluations on three datasets from the job domain: firstly, a dataset collected from the Austrian online student job portal Studo Jobs; secondly, the job dataset that was provided by XING after the RecSys Challenge 2017 (Abel et al. 2017); and finally, a dataset from a Kaggle competition on job recommendation sponsored by CareerBuilder. Our approach is compared to the state-of-the-art session-based recommender approaches (Hidasi and Karatzoglou 2018; Hidasi et al. 2015; Jannach and Ludewig 2017; Ludewig and Jannach 2018; Rendle et al. 2009) not only with respect to accuracy but also in terms of system-based and session-based novelty as well as coverage (Zhang et al. 2012). This is grounded in the growing awareness that factors other than accuracy contribute to the quality of recommendations (Herlocker et al. 2004; McNee et al. 2006). Moreover, novelty is especially an important metric for the job domain since applying to popular jobs may decrease a user's satisfaction due to high competition and less chance of getting hired (see e.g., Kenthapadi et al. 2017).

Contributions and findings The main contributions of this paper and the corresponding findings are as follows:

- We present a recommendation approach, which uses different autoencoder architectures to encode sessions from the job domain. We use the inferred latent session representations in a k-nearest neighbor manner to recommend jobs within a session.
- We compare our approach to methods from recent work (Hidasi and Karatzoglou 2018; Hidasi et al. 2015; Jannach and Ludewig 2017; Ludewig and Jannach 2018; Rendle et al. 2009) on the state-of-the-art session-based recommendation.
- We evaluate the efficacy of our approach on three datasets: firstly, a proprietary dataset collected from the online student job portal Studo Jobs; secondly, a publicly available job dataset that was provided by XING after the RecSys Challenge 2017; and thirdly, a publicly available job dataset from the job platform CareerBuilder.
- We train and test the autoencoders on two sources of job-related data: (1) interaction data from sessions and (2) content features of job postings, for which interactions took place during a session. Our results show that variational autoencoders provide competitive job recommendations in terms of accuracy compared to the state-of-the-art session-based recommendation algorithms.
- We additionally evaluate all session-based job recommender approaches in terms of the beyond-accuracy metrics with system-based and session-based novelty as well as coverage. We find that autoencoders can produce more novel and surprising recommendations compared to the baselines and, at the same time, provide relevant jobs for the user while maintaining a high coverage.
- We provide the implementation of our approach as well as a more detailed hyperparameter description in a public GitHub repository[6] in order to foster reproducible research.

[6] https://github.com/lacic/session-knn-ae.

Organization of the paper The remainder of the paper is structured as follows: In Sect. 2, we discuss related work. Section 3 outlines our approach to employ autoencoders for session-based job recommendation. Section 4 describes the baseline approaches, datasets, evaluation protocol and performance metrics. Section 5 elaborates on the results of our experiments. Finally, in Sect. 6, we conclude the paper and provide an outlook on our plans for future work.

2 Related work

At present, we identify two lines of research that are related to our work: (1) job recommender systems and (2) session-based recommender systems.

Job recommender systems Job recommender systems address a particular recommendation problem, in that a company might want to hire only a few candidates, while classic recommender systems typically recommend items that are relevant for a large number of users (Kenthapadi et al. 2017). There are two directions of the recommendation problem: One is to recommend jobs to a user given her user profile, while the other is to recommend candidates for a job posting. The directions of both problems can even be combined using a reciprocal recommender (Mine et al. 2013).

Research on recommending jobs to users has mostly focused on improving accuracy with methods like collaborative- and content-based filtering or hybrid combinations of both (Al-Otaibi and Ykhlef 2012; Zhang and Cheng 2016). One example of a hybrid job recommendation system that uses interaction data as well as content data is the work of Liu et al. (2017). Here, the recommendation problem corresponds to first searching for matching candidates for a given job and then recommending this job to these candidates. In another job recommender system presented in Hong et al. (2013), the authors propose to first cluster user profiles based on their characteristics and then to design separate recommendation strategies for each cluster.

In 2016, XING (a career-oriented social networking site based in Europe) organized a challenge for the ACM RecSys conference to build a job recommendation system (Abel et al. 2016) that recommends a list of job posts with which a user might interact in the upcoming week. The winning approach (Xiao et al. 2016) used a hierarchical learning-to-rank model to generate the recommendations, which captures semantic relevance, temporal characteristics of a user's profile information, the content of job postings and the complete log of user activities. The anonymized challenge dataset has since been employed, for instance, by Mishra and Reddy (2016), who built a gradient boosting classifier to predict if a given user will like a particular job posting. In 2017, XING organized another recommender challenge for the ACM RecSys conference (Abel et al. 2017). Here, the recommendation problem was turned into a search for suitable candidates when a new job posting is added to the system (i.e., the task constitutes a cold-start problem (Lacic et al. 2015)). The winning approach (Volkovs et al. 2017) spent considerable effort on feature engineering to train a gradient boosting algorithm, which determines the probability of whether or not a given candidate user profile is suited for a target job posting.

In our work, we employ the most recent version of the dataset provided by XING after the RecSys challenge 2017 to evaluate a range of approaches to provide job

recommendations in anonymous sessions. Besides, in our experiments, we use a proprietary dataset gathered from Studo Jobs, an Austrian student job portal, as well as a publicly available dataset from the job portal CareerBuilder.

Since in our work, we focus on session-based job recommendations, in the next paragraph, we summarize related work on session-based recommender systems.

Session-based recommender systems Most recommender systems require a user preference history in the form of explicit or implicit user interactions. Based on the user preference history, a user profile is created, which is the basis for approaches such as matrix factorization (Koren et al. 2009). However, it is not always possible to create such user profiles, e.g., to protect the privacy of users or due to inadequate resources. As a remedy, session-based recommender systems (Hidasi et al. 2015) have been proposed, which model a user's actions within a session, i.e., a short period when the user is actively interacting with the system. A simple approach toward session-based recommendation is to recommend similar items using item–item similarity as proposed by Sarwar et al. (2001). Hidasi and Tikk (2016) propose a general factorization framework that models a session using the average of the component latent item representations. Shani et al. (2005) use Markov decision processes to compute recommendations that incorporate the transition probability between items. Jannach and Ludewig (2017) use co-occurrence patterns as a basis for session-based recommendations. They report comparable and often even a superior performance of a heuristics-based nearest neighbor method (KNN) to generate recommendations in a session-based setting in comparison with competitive, state-of-the-art methods based on neural architectures. Hence, in our work, we also use two KNN-based methods, i.e., sequential session-based KNN and vector multiplication session-based KNN (Ludewig and Jannach 2018) as baseline algorithms due to their good performance and scalability as reported in related works (Jannach and Ludewig 2017; Kamehkhosh et al. 2017; Ludewig and Jannach 2018).

In general, applying neural networks in session-based recommendation systems has gained much attention in recent years. For instance, recent work (Tuan and Phuong 2017; Yuan et al. 2019) uses convolutional networks to produce session-based item recommendations. Song et al. (2016) proposed a neural architecture that combines both long-term and short-term temporal user preferences. They model these preferences through different long short-term memory (LSTM) networks in a stepwise manner. In this vein, Lin et al. (2018) introduce STAMP (short-term attention/memory priority) that simultaneously incorporates a user's general interest (i.e., long-term memory) and current interest (i.e., short-term memory). Wu et al. (2018) present an architecture for session-based recommendations that is based on graph neural networks. Here, using an attention network, each session is also represented by a session user's global preference and their current interest. The authors of Li et al. (2017) propose NARM (neural attentive recommendation machine), which uses an attention mechanism in a hybrid encoder to model the sequential behavior of a user and to extract the user's main purpose from the current session. As the authors show, this approach is specifically well suited to model long sessions.

Out of the different neural architectures, recurrent neural networks have become particularly popular for the task at hand (Chatzis et al. 2017; Hidasi et al. 2015; Smirnova and Vasile 2017). In the earlier mentioned work of Hidasi

et al. (2015), the authors showed that a recurrent neural network (RNN)-based approach can model variable-length session data. Other related papers on sequential data either improve the original algorithm (Hidasi and Karatzoglou 2018; Tan et al. 2016) or extend it by capturing additional information such as context (Twardowski 2016) or attention (Li et al. 2017). In later work, Hidasi et al. (2016) introduce an architecture (i.e., pRNN) that combines multiple RNNs to model sessions via clicks as well as via features of the clicked items such as content information. Here, each RNN handles a particular feature, such as the clicked item's textual representation. The authors show that, given the optimal training strategy, pRNN architectures can result in higher performance compared to feature-less session models. Due to its ability to incorporate content features of job postings in its model in addition to interactions within sessions, in our work, we use pRNN as a baseline approach as we also take into account content features of job postings as well as interactions.

In our work, we employ autoencoders, a type of neural network that can reduce the dimensionality of data (Kramer 1991), to infer latent session representations and to generate recommendations. Specifically, we propose to employ a classic autoencoder (Kramer 1991), a denoising autoencoder (Vincent et al. 2008) and a variational autoencoder (Jordan et al. 1999) to model and encode sessions. In this vein, we find that collaborative denoising autoencoders (CDAE) (Wu et al. 2016) are related to our work. CDAE utilize a denoising autoencoder (Vincent et al. 2008) by adding a latent factor for each user to the input. A denoising autoencoder can learn representations that are robust to small, irrelevant changes in the input. In CDAE, the number of parameters grows linearly with the number of users and items, which makes it prone to overfitting (Liang et al. 2018). Also related to our work is neural collaborative filtering (He et al. 2017), where a neural architecture, which can learn any function from data, replaces the dot product between the latent user and item features. However, this model has a similar issue as CDAE and, thus, grows linearly with the number of sessions and jobs as the authors of Liang et al. (2018) describe.

Finally, with respect to evaluation, to the best of our knowledge, related work on evaluating session-based recommender systems with beyond-accuracy metrics such as system-based and session-based novelty, or coverage is scarce. Only in recent work, Ludewig and Jannach (2018) evaluate session-based recommender systems in light of coverage and popularity bias. With this work, we aim to contribute to this sparse line of research as we evaluate all approaches in this work with respect to system-based and session-based novelty as well as coverage, in addition to accuracy.

3 Approach

In this section, we describe our approach toward a session-based job recommender system using autoencoders. In Sect. 3.1, we first describe how we encode sessions with autoencoders. Then, in Sect. 3.2, we outline our method to model the input session vectors from interactions and content features. Finally, Sect. 3.3 details how we compute session-based job recommendations.

3.1 Encoding sessions using autoencoders

Autoencoders are a type of neural network, which were popularized by Kramer (1991) as a more effective method than principal component analysis (PCA) with respect to describing and reducing the dimensionality of data. Autoencoders are trained in an unsupervised manner where the network is trying to reconstruct the input by passing the information to the output layer through a bottleneck architecture. For our work, we employ three variants of autoencoder architectures to represent a session: (1) a classical autoencoder (AE), (2) a denoising autoencoder (DAE) and (3) a variational autoencoder (VAE).

Autoencoder (AE) The simplest form of an autoencoder has only one hidden layer (i.e., the latent layer) between the input and output (Bengio et al. 2007). The latent layer takes the vector $x_s \in \mathbb{R}^D$, which represents the session and maps it to a latent representation $z_s \in \mathbb{R}^K$ using a mapping function:

$$z_s = h(x_s) = \sigma(W^T x_s + b)$$

where W is a $D \times K$ weight matrix, $b \in \mathbb{K}$ is an offset vector and σ is usually a non-linear activation function. Using z_s, the network provides a reconstructed vector $\hat{x_s} \in \mathbb{R}^D$, which is calculated as:

$$\hat{x_s} = \sigma(W' z_s + b')$$

By adding one or more layers between the input and latent layer, we create an *encoder* and, correspondingly, a *decoder* by doing the same between the latent and output layer, hence the name autoencoder. During inference, we use the output of the latent layer (i.e., the information bottleneck) to represent the latent session vector z_s.

In our experiments, for σ, we use rectified linear units (ReLU[7]) (Nair and Hinton 2010) activation function for all layers except the final output layer, where a sigmoid activation function is used. Furthermore, we use a $D_s - 256 - 100 - 256 - D_s$ network architecture,[8] where D_s is the dimension of the original vector representation of the session that is encoded using job interactions with or without the corresponding job content data. To train the network, we use RMSprop (Tieleman and Hinton 2012) and minimize the Kullback–Leibler divergence (Fischer and Igel 2012).

We also experimented with adding additional encoder/decoder layers as well as increasing the layer size (e.g., layers with a size of 1000) but did not see any major performance differences besides an increased training complexity. Both Adam and RMSProp are two of the most popular adaptive stochastic algorithms for training deep neural networks. In our work, we focused on RMSProp.

Denoising Autoencoder (DAE) As shown by Vincent et al. (2008), extending autoencoders by corrupting the input can show surprising advantages. The idea of a

[7] For an input x, $relu(x) = max(0, x)$.

[8] We also tested higher values for the dimension of the latent layer (e.g., layers with a size of 1000) as well as adding additional encoder/decoder layers, but did not find enough accuracy improvement that would justify the additional computation burden when calculating session similarities in real time.

denoising autoencoder is to learn representations that are robust to small, irrelevant changes in the input. Corrupting the input can be done on either one or multiple layers before we calculate the final output.

In our DAE model, we get a corrupted input $\hat{x}$ using the commonly employed additive Gaussian noise on the input layer with a probability of 0.5. Like earlier, we use the same $D_s - 256 - 100 - 256 - D_s$ architecture, ReLU and sigmoid activation functions, the RMSprop optimization algorithm and the Kullback-Leibler divergence as loss function.

Variational Autoencoder (VAE) Another approach to extract the latent representation z_s is to use variational inference (Jordan et al. 1999). For that, we approximate the intractable posterior distribution $p(z_s|x_s)$ with a simpler variational distribution $q_\phi(z_s|x_s)$, for which we assume an approximate Gaussian form with an approximately diagonal covariance:

$$\log q_\phi(z_s|x_s) = \log \mathcal{N}(z_s;\mu, \sigma^2 I)$$

where μ and σ^2 is the encoded output given the input vector representation x_s of a session. To be more precise, we use additional neural networks as probabilistic encoders and decoders. Most commonly, this is done using a multilayered perceptron (MLP). For the above-mentioned $q_\phi(z_s|x_s)$, we calculate:

$$\begin{aligned} \mu &= W_2\, h + b_2 \\ \log \sigma^2 &= W_3\, h + b_3 \\ h &= \text{ relu } (W_1\, x_s + b_1) \end{aligned}$$

where $\{W_1, W_2, W_3, b_1, b_2, b_3\}$ are weights and biases of the MLP and are part of variational parameters ϕ. While decoding, we sample the latent representation and produce a probability distribution $\pi(z_s)$ over all features from the input session vector x_s. As we deal with implicit data, to calculate the probabilities, we let $p_\theta(x_s|z_s)$ be a multivariate Bernoulli (Kingma and Welling 2013), whose probabilities in the MLP we calculate as:

$$\log p(x_s|z_s) = \sum_{i=1}^{D} x_{si} \log y_i + (1 - x_{si}) \cdot \log(1 - y_{si})$$
$$y_s = f_\theta(\, W_5 \text{ relu } (W_4\, z_s + b_4) + b_5\,)$$

where f_θ is an element-wise nonlinear activation function (i.e., in our case a sigmoid) and $\theta = \{W_4, W_5, b_4, b_5\}$ are weights and biases of the MLP.

The generative model parameters θ are learned jointly with variational parameters ϕ by optimizing the marginal likelihood of the data. The objective is thus to minimize the distance between the variational lower bound $\mathcal{L}(\theta, \phi, x)$ and a certain prior (Kingma and Welling 2013; Liang et al. 2018), which in case of VAEs is the Kullback–Leibler divergence (Fischer and Igel 2012) of $q_\phi(z_s|x_s)$ and $p(z_s|x_s)$. As we are sampling z_s from q_ϕ in the variational lower bound, in order to learn the model, we need to apply the reparametrization trick (Kingma and Welling 2013; Rezende et al. 2014) by sampling $\epsilon \sim \mathcal{N}(0, I_K)$ (also seen later in Fig. 2) and reparametrize

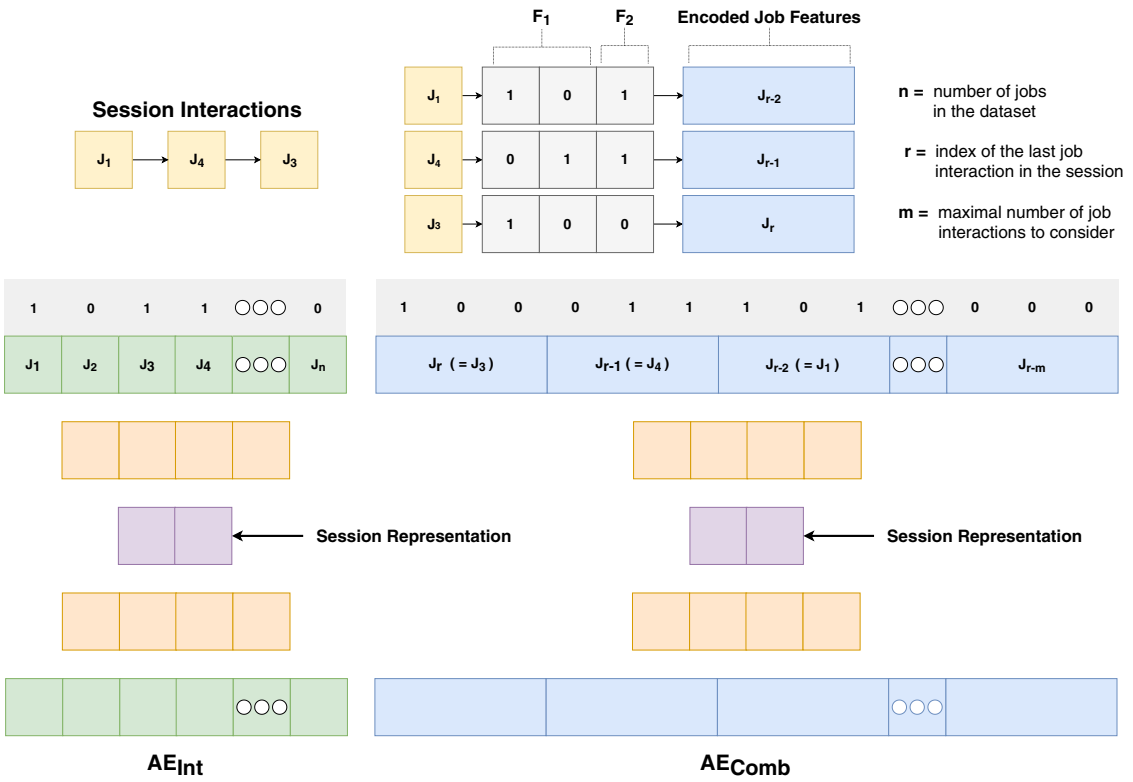


Fig. 1 Modeling session vectors. The input of the utilized autoencoder is a session representation, which can be binary-encoded using job interactions with or without the corresponding job content data. For example, a standard autoencoder that only considers interaction data (denoted as AE_{Int}) will expect a binary encoded vector with a dimension that equals the number of jobs in the underlying dataset. To combine this with job content data (denoted as AE_{Comb}), we use the most recent m job interactions within the session and generate a binary encoding of the job content features in descending order

$z_s = \mu_\phi(x_s) + \epsilon \odot \sigma_\phi(x_s)$. Hence, the gradient with respect to ϕ can be back-propagated through the sampled z_s.

In our experiments, we utilize the described VAE model with a similar architecture as previously mentioned: $D_s - 256 - 100 - 256 - D_s$ (i.e., the encoder and decoder MLPs are symmetrical). Furthermore, for all three autoencoder architectures, we experiment on additionally incorporating the self-attention mechanism (e.g., as Lin et al. 2017; Parikh et al. 2016; Vaswani et al. 2017 do in their work) on the encoder layer.

3.2 Modeling session vectors

The input for any of the three autoencoder variants is a binary-encoded representation of the session x_s. As shown in Fig. 1, we propose the following two variants of how to train the autoencoder models that will be used to infer the latent representation z_s.

Variant 1: Modeling from interactions We construct x_s by only using the job interaction data of a given session. In the remaining paper, we denote the three autoencoder models, which only use job interaction data as AE_{Int}, DAE_{Int} and VAE_{Int}. We create session vectors of size n_s, where n_s is the number of jobs in the

underlying dataset. Each job is then assigned an index in this vector. The interactions on the corresponding job indices are set to 1, while we set the rest to 0. One possible drawback of this approach is that due to the ephemeral nature of job postings, we would need to frequently retrain the utilized model in order to consider new jobs coming to the system (Matuszyk et al. 2015). Moreover, this will also impact the size of the input vector x_s, which will constantly be increasing with every new job.[9]

Variant 2: Modeling from interactions and content In order to mitigate the need to retrain the autoencoder models frequently, we also propose to leverage the content of job postings, with which anonymous users have interacted during a session (i.e., combine interaction data with content data). Given a set of content features $F = \{f_1, \ldots, f_l\}$, we first convert each job interaction in a session to a binary vector of size $n_j = \sum_{i=1}^{l} dist(f_i)$, where $dist(f_i)$ gives the number of distinct values of a job feature f_i. Each feature value is then assigned an index in this vector, and the existing feature values are set to 1, while the rest are 0. To create the session vector x_s, starting from the most recent job interaction, we concatenate the last m converted job interactions. In case the number of job interactions is less than m, x_s is right-padded with 0-filled job vectors, which results in x_s being of size $n_j \times m$. We denote the three autoencoder models that use the content features of job interactions as AE_{Comb}, DAE_{Comb} and VAE_{Comb}. Note also that we introduce the parameter m to end up with an input vector x_s that has a fixed length and a model that is less sensitive to new job postings that are added to the system.

3.3 Computing session-based job recommendations

We formulate the recommendation problem as follows: Given a target session s_t, in which there was an interaction with at least one job j_i from the set of available jobs $J = \{j_1, \ldots, j_n\}$, the task is to predict the next jobs this user will likely interact with. In order to compute recommendations, as shown in Fig. 2, we first extract the output z_s for the sessions that are available in the training set. During prediction time, for a given target session s_t, we proceed to infer its latent representation first to find the top-k similar past sessions. In order to reduce the computational burden and allow for efficient recommendation,[10] we extract a subset of all sessions, where the users have interacted with the last job in s_t. Using z_s, we compute the cosine similarity between the respective target and candidate session and use the top-k similar sessions to recommend jobs. Jobs are then ranked based on the following score:

$$sKNN(s_t, j_i) = \sum_{i=1}^{n} sim(s_t, s_i) \times 1_{s_i}(j_i)$$

[9] This effect can, however, be damped by removing obsolete job postings, but would still result in a constantly changing input dimension.

[10] The number of stored sessions can easily pass the million mark and cause for unnecessary calculations once a recommender system is running for a longer period.

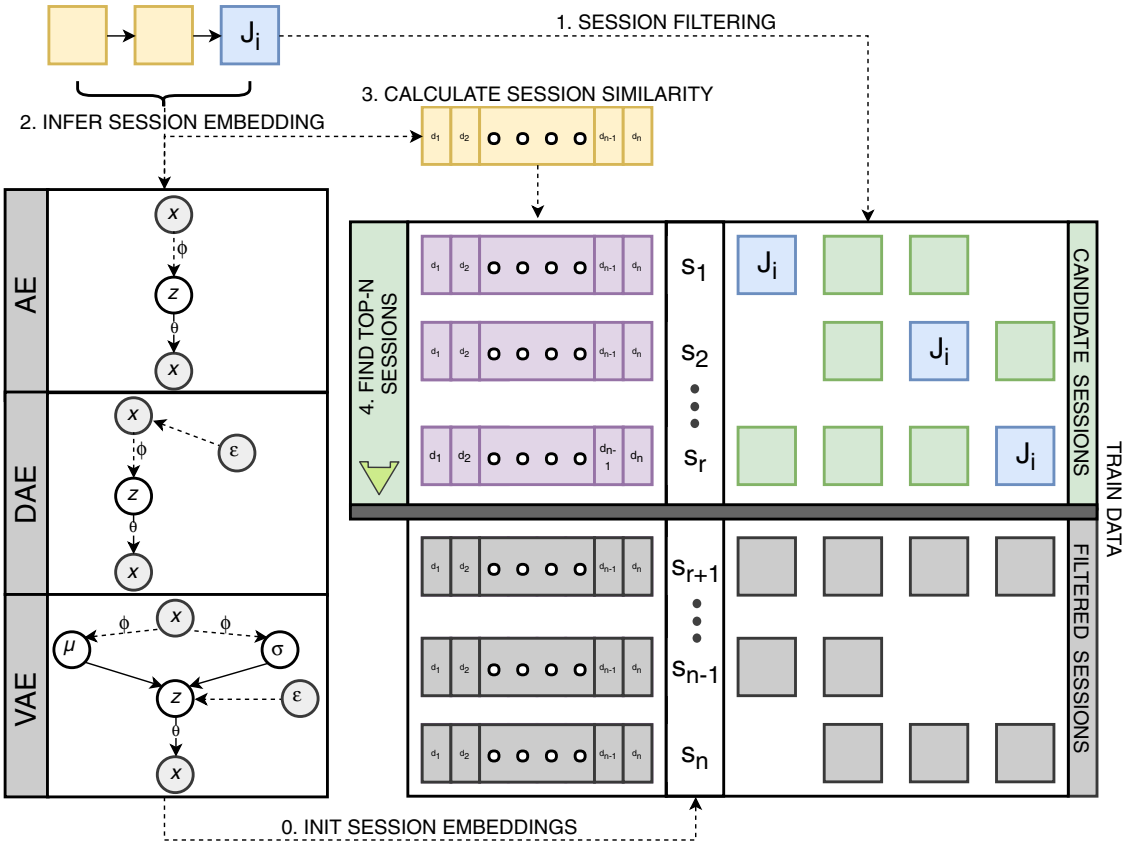


Fig. 2 Computing session-based job recommendations. Using the trained autoencoders, we infer latent representation for (1) sessions in the training data and (2) the current target session for which we recommend jobs. Jobs from the top-k similar candidate sessions (filtered by the currently interacted job posting) are recommended to the target session

where $1_{s_i}(j_i)$ is 1 if the candidate session s_i contains the job j_i and 0 otherwise (as in Bonnin and Jannach 2015; Jannach and Ludewig 2017).

4 Experimental setup

In this section, we present the baseline approaches and the datasets we used for this study. We outline the evaluation protocol and the performance measures, which we employed to compare all approaches. In our evaluation, we contribute to the limited amount of related work (e.g., like Ludewig and Jannach 2018) as we evaluate all approaches both concerning accuracy and beyond-accuracy measures (i.e., system-based and session-based novelty as well as coverage).

4.1 Baseline approaches

We utilize well-known baselines and compare our approach to the following state-of-the-art methods (Ludewig and Jannach 2018) for session-based recommendation:

POP A simple and yet often strong baseline for session-based recommendation is the popularity-based approach. As in Hidasi et al. (2015), the results are always the same top-k popular items from the training dataset.

iKNN The item-KNN approach recommends jobs that are similar to the actual job that is interacted with during the session. As in Hidasi et al. (2015), we use the cosine similarity and include regularization to avoid coincidental high similarities between rarely visited jobs.

BPR-MF One of the commonly used matrix factorization methods for implicit feedback is Bayesian personalized ranking (Rendle et al. 2009). As in Hidasi et al. (2015), we use the average of job feature vectors of the jobs that had occurred in the current session as the user feature vector to apply it directly to generate a session-based recommendation. That is, similarities of the feature vectors are averaged between a candidate job and the jobs of the current session.

Bayes Following the Bayesian rule, we calculate the conditional probability of a job x_i being clicked based on the previous r interactions of the current session s:

$$P(x_i|x_{s_1}, \ldots, x_{s_r}) = \frac{\prod_{j=1}^{r} P(x_{s_j}|x_i) \times P(x_i)}{\prod_{j=1}^{r} P(x_{s_j})}$$

This approach is, from a computational perspective, inexpensive to calculate and run in an online setting.

GRU4Rec Recently, Hidasi et al. (2015) showed that recurrent neural networks are excellent models for data generated in anonymous sessions. GRU4Rec combines gated recurrent units with a session-parallel mini-batch training process, and it incorporates a ranking-based loss function. For our study, we use the most recent improvement in GRU4Rec (Hidasi and Karatzoglou 2018). This GRU4Rec version employs a new class of loss functions tied together with an improved sampling strategy.

pRNN Another recent advancement of Hidasi et al. (2016) shows how to incorporate item features into the representation of neural networks. They propose several different architectures based on GRU units and ways to train them. We use a parallel architecture with simultaneous training for our experiments. This approach utilizes both a one-hot encoding of the current item interaction and an item representation as inputs for the subnets. The trained model uses the TOP1 loss function as defined in Hidasi et al. (2015).

sKNN Recent research has shown that computationally simple nearest-neighbor methods can be effective for session-based recommendation Jannach and Ludewig (2017). The session-based KNN approach first determines the k most similar past sessions in the training data. Sessions are encoded as binary vectors of the item space, and a set of k nearest sessions is retrieved for the current session using cosine similarity. The final job score is calculated by aggregating the session similarity over all the sessions that contain the candidate job.

V-sKNN Vector multiplication session-based KNN (V-sKNN) is a variant of sKNN that considers the order of the elements in a session. The idea here is to create a real-valued vector by putting more weight on recent interactions, where

Table 1 Statistics of the datasets Studo, RecSys Challenge 2017 (i.e., RecSys17) and CareerBuilder12

Dataset	# Interactions	# Sessions	# Jobs	Sparsity (%)
Studo	191,259	26,785	1111	99.36
RecSys17	55,380	16,322	15,686	99.98
CareerBuilder12	661,910	120,147	197,590	99.99

While Studo has more sessions and job interactions, the RecSys Challenge 2017 dataset has more job postings that can be recommended. CareerBuilder12 is the largest dataset, but also has the highest sparsity

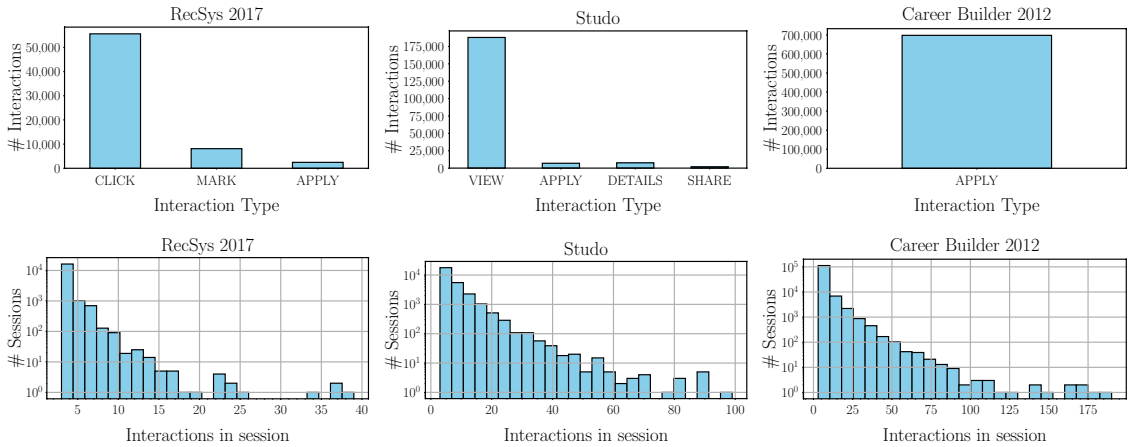


Fig. 3 Number of interactions based on the interaction type (top) and the distribution of session sizes (bottom) is shown for the RecSys Challenge 2017 (left) and Studo (middle) and CareerBuilder12 datasets. Overall, the distribution of interaction types is similar between the datasets where the *click*, *view* and *apply* interactions mostly dominate

only the very last element of the session obtains a value of "1" (Ludewig and Jannach 2018). For this, a linear decay function is used that depends on the position of an element within the session.

S-sKNN Sequential session-based KNN (S-sKNN) puts more weight on elements that appear later in the session in a similar way as V-sKNN (Ludewig and Jannach 2018). This effect is, however, achieved by giving more weight to neighboring sessions which contain recent items of the current session.

4.2 Datasets

For this study, we employ three different datasets from the job domain. The first dataset, *Studo*, is a proprietary dataset collected from the online platform Studo Jobs, a job-seeking service for university students. The second dataset *RecSys17* is the latest version of the data provided by XING after the RecSys Challenge

Table 2 Binary-encoded content features of our three datasets

Studo		RecSys17		CareerBuilder12	
Content feature	Encoding	Content feature	Encoding	Content feature	Encoding
Job state†	10	Region†	17	State†	55
Job country‡	1	Country‡	4	Requirement topic	20
Job begins now	1	Is payed	2	Title topic	20
Job effort	1	Career level	6	Description topic	20
Job language	1	Industry Id††	23		
Job discipline††	40	Discipline Id††	22		
Employment type‡‡	6	Employment‡‡	5		

For Studo, concatenating all job features results in a job vector with a dimensionality of 60. For the RecSys17 dataset, this results in a job vector with a dimension of 79. For the CareerBuilder12 dataset, this results in 115 dimensionality vectors. We also put the same annotation on content features, which have a similar meaning in both datasets. The *Job Discipline* feature is the only one in Studo, which represents a combination of the *Discipline Id* and *Industry Id* features from the RecSys17 dataset

2017 (Abel et al. 2017). The third dataset *CareerBuilder12* is from an open Kaggle competition, called Job Recommendation Challenge,[11] provided by the online employment Web site CareerBuilder. The statistics of all three datasets are given in Table 1. As seen, all datasets have a high sparsity: 99.36% for Studo, 99.98% for RecSys17 and 99.99% for CareerBuilder12. Studo contains a higher number of sessions when compared to RecSys17 but has a much smaller number of available jobs that can we can recommend. CareerBuilder12 is the largest dataset of the three, but only contains job applications as interactions. In the next paragraphs, we describe the three datasets in more detail.

Studo The dataset contains job interactions from anonymous user sessions from a period of three months between September 2018 and December 2018. All job interactions in this dataset have an anonymous session id assigned to them. As seen in the top row of Fig. 3, the Studo dataset contains four interaction types, i.e., job *view*, show company *details*, *apply* and *share* job. As shown at the bottom row of Fig. 3, the log histogram of session sizes follows a skewed pattern, which means that most sessions have a small number of interactions. In particular, every session has 6.98 interactions on average and a median of 5 interactions.

Concerning content features reported in Table 2, in the Studo dataset, we utilize seven content features of job postings. The *Job State* determines 1 out of 9 Austrian federal states. *Job Country* indicates whether the job is in Austria or some other country. The *Job Begins Now* feature specifies whether the job candidate can start immediately working on the advertised position. We relate this feature to the *Is Payed* feature from the RecSys17 dataset as companies typically pay for job postings to be shown if they urgently need candidates. Studo's *Job State* is similar to the *Region* feature from RecSys17, the same holds true for Studo's *Employment Type*,

[11] https://www.kaggle.com/c/job-recommendation.

which can be related to the *Employment* feature from RecSys17. *Job Effort* indicates whether the concrete working hours are specified; otherwise, the default working hours are assumed. The *Job Language* feature specifies whether the job requires the usage of either the German or English language. Furthermore, a job posting can also be described by a subset of 40 different *Job Discipline* labels and a subset of 6 different *Employment Type* labels. The *Job Discipline* feature can actually be regarded as a combination of the *Discipline Id* and *Industry Id* features from the RecSys17 dataset. As described in Sect. 3.2, we use all content features from the Studo dataset to create a binary-encoded job vector with a dimensionality of 60. Finally, we have 77.7% uniquely encoded job vectors, which consist, on average, of 11.8% assigned feature values.

RecSys17 The dataset contains six different interaction types that were performed on the job items. For this study, we only keep the *click*, *bookmark* and *apply* interactions (as seen on the top of Fig. 3). We remove the *delete recommendation* and *recruiter interest* interactions as these are irrelevant in our setting. Moreover, we discard *impression* interactions as they are created when XING shows a job to a user. As stated by Bianchi et al. (2017), an impression does not imply that the user has interacted with the job. The dataset consists of interactions from a period of three months (from November 6, 2016, until February 3, 2017). We manually partition the interaction data of the RecSys dataset into sessions using a 30-minute idle threshold (as in Quadrana et al. 2017). The resulting sessions have, on average, 3.62 interactions per session and a median of 3 interactions.

Also, the RecSys17 dataset contains content features about the job postings, such as career level or type of employment. From this set, we select seven features as content-based input for our approaches and discarded the numeric IDs of title and tags, since those would lead to very big encodings. The chosen features closely resemble the features that are present in the Studo dataset. More specifically, from RecSys17, we use the following features, as shown in Table 2: *Region*, *Employment*, *Is Payed*, *Discipline Id*, *Career Level*, *Industry Id* and *Country*. The *Region* content feature is a categorical feature with 17 possible value, like the *Employment* feature with 5 values. The *Is Payed* content feature indicates if the posting is a paid for by a company. The *Discipline Id* is a categorical feature with 22 different values that represent disciplines such as consulting or human resources. The categorical feature *Career Level* can take 7 values, *Industry Id* represents industries such as finance, and *Country* denotes the code of the country in which the job is offered.[12] Overall, we end up with a job vector that has dimensionality of 79. We find that only 33.58% of the job vectors are unique and have on average 8.86% assigned feature values.

CareerBuilder12 The dataset contains job applications from a period of almost three months. No other interaction types are present in this dataset. The sessions are created via a time-based split of 30 min. Due to the nature of job applications, most sessions contain very few interactions. The interactions in the dataset happen over 13 weeks. Thus, similar to the other two datasets, it happens over almost three

[12] https://www.recsyschallenge.com/2017.

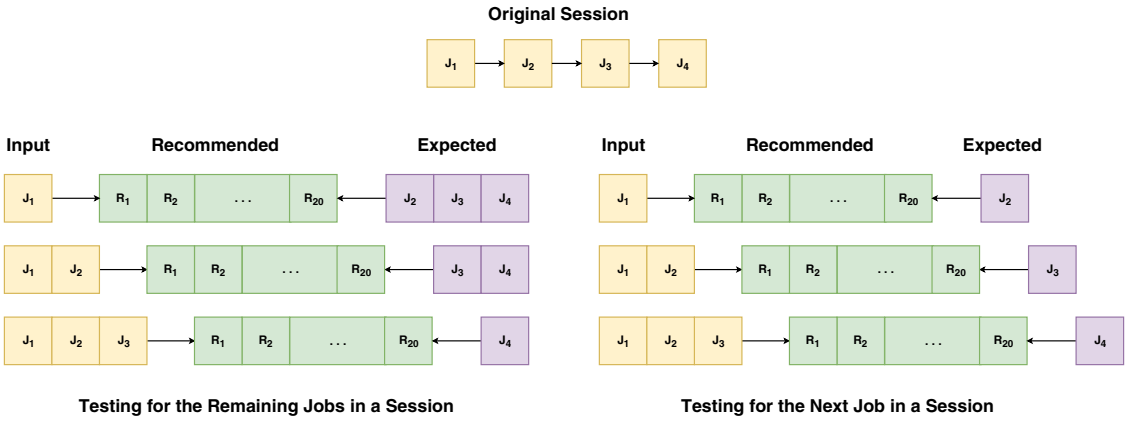


Fig. 4 Our evaluation protocol for one exemplary session consisting of four jobs. We distinguish between (1) comparing the recommended jobs with the remainder of the interactions in a session (left) and (2) comparing the recommended jobs with the next job interaction (right)

months, i.e., from April 2012 to June 2012. The sessions have, on average, 5.64 job applications per session, whereas the median is 4 applications per session.

Regarding content features, the CareerBuilder12 dataset contains textual descriptions of the jobs as well as categorical data for the location. From the content data, the 55 different states are used in the form of one-hot encodings. Since in our work, we utilize categorical job features as input to our models, we additionally inferred categorical topics for each of the 3 textual features (i.e., title, description and requirements). That is, for every textual feature, we trained a separate latent Dirichlet allocation (LDA) model from which we extracted 20 distinct topics. This procedure resulted in every job posting having a requirement, title and description topic assigned to them. Thus, the resulting feature vector of a job posting is of size 115. For this largest dataset, 13.46% of vectors are unique, and those vectors have only 2.58% assigned feature values.

4.3 Evaluation protocol

We employ a time-based split on all three datasets to create train and test sets. For this, we put the sessions from the last 14 days (i.e., 2 weeks) in the test set of the respective dataset and use the remaining sessions for training. For each set, we keep only sessions with a minimal number of 3 interactions.[13] Like (Quadrana et al. 2017), we filter items in the test set that do not belong to the train set as this enables a better comparison with model-based approaches (e.g., RNNs), which can only recommend items that have been used to train the model. In Studo, this procedure results in 23, 738 sessions to train and 3047 to test the approaches. For the RecSys17 dataset, this results in 12, 712 sessions for training and 3610 sessions for testing. In

[13] We chose 3 for the minimum amount of interaction as it is the lowest median of interactions in a session across all three datasets, as reported in Sect. 4.2.

the case of the much larger CarrerBuilder12 dataset, the train set contains 108, 783 sessions, whereas the test set has 11, 364 sessions.

Training and testing the algorithms We first train all approaches on the respective training data. In order to evaluate the performance of the utilized session-based recommendation algorithms, for each session in the test data, we iteratively subsample its interactions. That is, after each session interaction, we recommend 20 jobs for the current target session state and compare the predictions with the remaining interactions. We start this procedure for every session after the first interaction and end before the last one. In this setting, as shown in Fig. 4, we explore two evaluation cases: comparing the recommended jobs with (1) the remainder of the interactions in the session and (2) with the next job interaction (i.e., next item prediction; same as in Hidasi and Karatzoglou 2018; Hidasi et al. 2015).

For our proposed method, that uses content features in combination with user interactions to encode the input for the autoencoders (i.e., as described in Sect. 3.2), we use the top 25 recent job interactions to infer the session representation. That is, we set the parameter $m = 25$ as more than 98% of all sessions in Studo, and almost all sessions in the RecSys17 dataset do not have more than 25 job interactions (i.e., as shown in Fig. 3).

Hyperparameter optimization To optimize hyperparameters, we further split the train sets by the same time-based split to generate validation sets. Thus, we use the last 2 weeks of the train set as a separate validation set and the remaining sessions to train our models. The resulting split for the Studo dataset is 19, 245 sessions in the validation train set and 3273 sessions in the validation test set. For the RecSys17 dataset, we have 8001 sessions in the validation train set and 2046 sessions in the validation test set. In case of CareerBuilder12, the validation train set contains 51, 717 sessions and the validation test set 10, 574 sessions. Note that some sessions did no longer have the minimal number of 3 interactions and were filtered out. As a consequence, the combination of the validation train and validation test set is smaller than the original train set. The results of the hyperparameter optimization step are described in Sect. 5.3.

4.4 Evaluation metrics

We quantify the recommendation performance of each approach concerning accuracy and beyond-accuracy metrics like system-based and session-based novelty. More specifically, in our study, we use the following performance measures:

Normalized Discounted Cumulative Gain (nDCG) nDCG is a ranking-dependent metric that measures how many jobs are predicted correctly. Also, it takes the position of the jobs in the recommended list into account (Parra and Sahebi 2013). It is calculated by dividing the DCG of the session's recommendations with the ideal DCG value, which is the highest possible DCG value that can be achieved if all the relevant jobs would be recommended in the correct order. The nDCG metric is based on the *Discounted Cumulative Gain (DCG@k)*, which is given by Parra and Sahebi (2013):

$$DCG@k = \sum_{i=1}^{k} \frac{2^{rel(i)} - 1}{log(1+i)}$$

where $rel(i)$ is a function that returns 1 if the recommended job at position i in the recommended list is relevant. nDCG@k is calculated as DCG@k divided by the ideal DCG value iDCG@k, which is the highest possible DCG value that can be achieved if all the relevant jobs would be recommended in the correct order. Over all the sessions, it is given by:

$$nDCG@k = \frac{1}{|S|} \sum_{s \in S} \left(\frac{DCG@k}{iDCG@k} \right)$$

Mean reciprocal rank (MRR) MRR is another metric for measuring the accuracy of recommendations and is given as the average of the reciprocal ranks of the first relevant job in the list of recommended jobs, i.e., 1 for the first position, $\frac{1}{2}$ for the second position, $\frac{1}{3}$ for the third position and so on. This means that a high MRR is achieved if relevant jobs occur at the beginning of the recommended jobs list (Voorhees 1999). Formally, it is given by Aggarwal (2016):

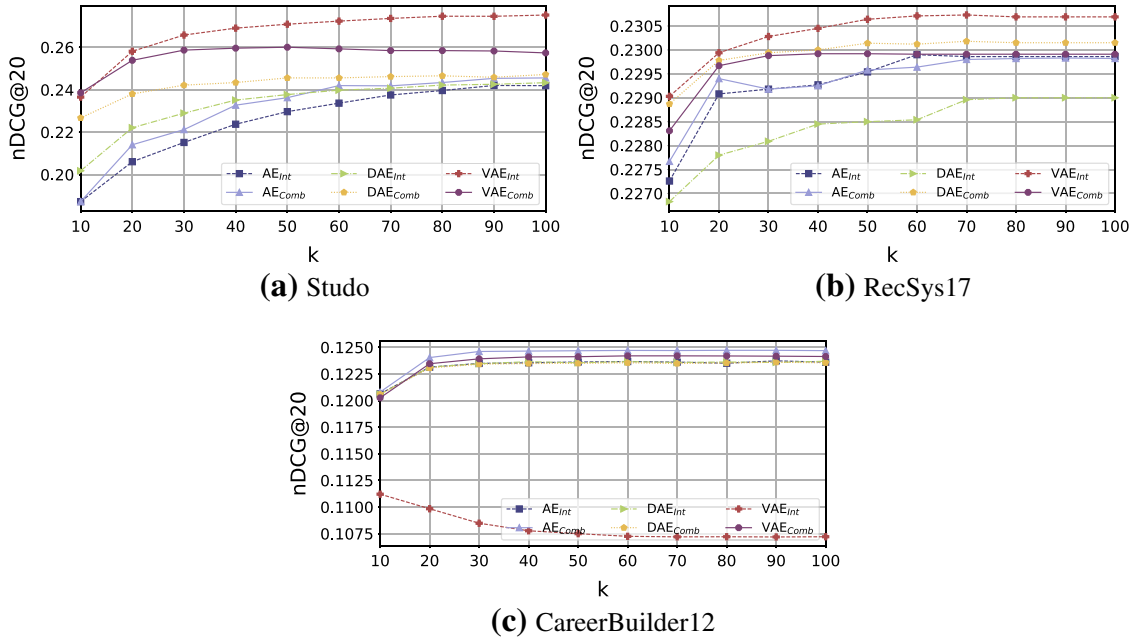
$$MRR@k = \frac{1}{|S|} \sum_{s \in S} \frac{1}{|H_s|} \sum_{H_j \in H_s} \frac{1}{rank(H_j, R_k)} \tag{1}$$

Here, H_s is the history of the current session s and $rank(H_j, R_k)$ is the position of the first relevant job H_j in the recommended job list R_k.

System-based novelty (EPC) System-based novelty denotes the ability of a recommender to introduce sessions to job postings that have not been (frequently) experienced before in the system. A recommendation that is accurate but not novel will include items that the session user enjoys, but (probably) already knows. Optimizing system-based novelty has been shown to have a positive, trust-building impact on user satisfaction (Pu et al. 2011). Moreover, system-based novelty is also an important metric for the job domain since applying to popular jobs may decrease a user's satisfaction due to high competition and less chance of getting hired (see, e.g., Kenthapadi et al. 2017). In our experiments, we measure the system-based novelty using the expected popularity complement (EPC) metric introduced by Vargas and Castells (2011). In contrast to solely popularity-based metrics (e.g., Zhou et al. 2010), EPC also accounts for the recommendation rank and the relevance for the current session. Thus, the system-based novelty $nov_{system}(R_k|s)$ for the recommendation list R_k of length k for session s is given by:

$$EPC@k = \frac{1}{|S|} \sum_{s \in S} \frac{1}{|R_k|} \sum_{R_i \in R_k} disc(i) p(rel|R_i, s)(1 - p(seen|R_i))$$

Here, $disc(i)$ is a discount factor to weight the recommendation rank i [i.e., $disc(i) = 1/log_2(i+1)$] and $p(rel|R_i, s)$ is 1 if the recommended job R_i is relevant for session s or 0 otherwise (i.e., only jobs that are in the current session history are taken



(a) Studo **(b)** RecSys17

(c) CareerBuilder12

Fig. 5 The figures show the influence of the neighborhood size k for picking top-k similar sessions when comparing the three autoencoder variations on both interaction data and combined data. We find that the recommendation accuracy converges when k is picked to be around 60 or more

into account). Finally, $p(seen|R_i)$ defines the probability that a recommended job R_i was already seen in the system, i.e., $p(seen|R_i) = log_2(pop_{R_i} + 1)/log_2(pop_{MAX} + 1)$.

Session-based novelty (EPD) In contrast to system-based novelty, session-based novelty incorporates the semantic content of jobs and represents how *surprising* or *unexpected* the recommendations are for a specific session history (Zhang et al. 2012). Given a distance function $d(H_i, H_j)$ that represents the dissimilarity between two jobs H_i and H_j, the session-based novelty is given as the average dissimilarity of all job pairs in the list of recommended jobs R_k and jobs in the current session history H_s (Zhou et al. 2010). In our experiments, we use the cosine similarity to measure the dissimilarity of two job postings using a raw job vector, which contains 1 if a session interacted with it and 0 otherwise. Again, we use the definition by Vargas and Castells (2011) that takes the recommendation rank as well as the relevance for the current session into account. Hence, we measure the session-based novelty $nov_{session}(R_k|s)$ for the recommendation list R_k of length k for session s by the expected profile distance (EPD) metric:

$$EPD@k = \frac{1}{|S|} \sum_{s \in S} \frac{1}{|R_k||H_s|} \sum_{R_i \in R_k} \sum_{H_j \in H_s} disc(i) p(rel|R_i, s) d(R_i, H_j)$$

Here, H_s is the current history of a session s and $disc(i)$ as well as $p(rel|R_i, s)$ are defined as for the EPC metric for measuring the system-based novelty.

Coverage With coverage (Adomavicius and Kwon 2012; Ludewig and Jannach 2018), we assess how many jobs a recommender approach can cover with its predictions. As such, we additionally report the job coverage of each evaluated algorithm. We define the coverage as the ratio between the jobs that have been recommended

and jobs that would be available for recommendation. Here, we make a distinction between coverage types and report the job coverage (1) on the full dataset, i.e., how many of all available jobs can we recommend, and (2) on the test dataset, i.e., how many of the jobs can we recommend that we expect to be interacted with during a session.

5 Results

In this section, we present our experimental results. We first compare the performance of the respective models when used in a k-nearest neighbor manner and then analyze the embedding space of the best-performing autoencoder model. After that, we show the best hyperparameter configurations used for the baseline approaches and then discuss the performance of our approach compared to these baselines.

5.1 Comparing the recommendation performances of autoencoders

We compare the recommendation performance of all three variants of autoencoders, i.e., AE, DAE, VAE, trained on interactions as well as on content. This results in six autoencoder variants in total. We train all autoencoder models for a maximum of 50 epochs or until the error on the validation test set converges. We made additional experiments and incorporated the self-attention mechanism on the encoder layer (e.g., as in Lin et al. 2017; Parikh et al. 2016; Vaswani et al. 2017). We did not find any major improvements, so we do not report the results of these 6 additional autoencoder models.

Figure 5 shows the results of the autoencoder comparison in terms of *nDCG*@20. We compare the results across different values for the neighborhood size k, ranging from 10 to 100. We find that VAE_{Int}, which only uses interactions to encode the input vector, outperforms all other approaches on the Studo and RecSys17 datasets. When combining interaction data with content features (i.e., AE_{Comb}, DAE_{Comb} and VAE_{Comb}), VAE_{Comb} performs the best on the Studo dataset and slightly worse than DAE_{Comb} on the RecSys17 dataset. For the CareerBuilder12 dataset, all approaches except the VAE_{Int} approach have a similar performance. Such accuracy performance for VAE_{Int} suggests that having a much larger item space can be problematic for the generative autoencoder variant. As the variational autoencoder outperforms the other approaches in the majority of the configurations, in the next step, we compare it to the baseline methods. Furthermore, we find that for all autoencoders, accuracy converges after $k = 60$. Thus, in Sect. 5.4, we report the results of VAE_{Int} and VAE_{Comb} using top-60 similar sessions for recommendation.

5.2 Embedding analysis

To better understand the autoencoder models' actual effectiveness, we employ the t-SNE algorithm (Maaten and Hinton 2008) to visualize the embedding spaces. The t-SNE method enables us to visualize high-dimensional data. It reduces the

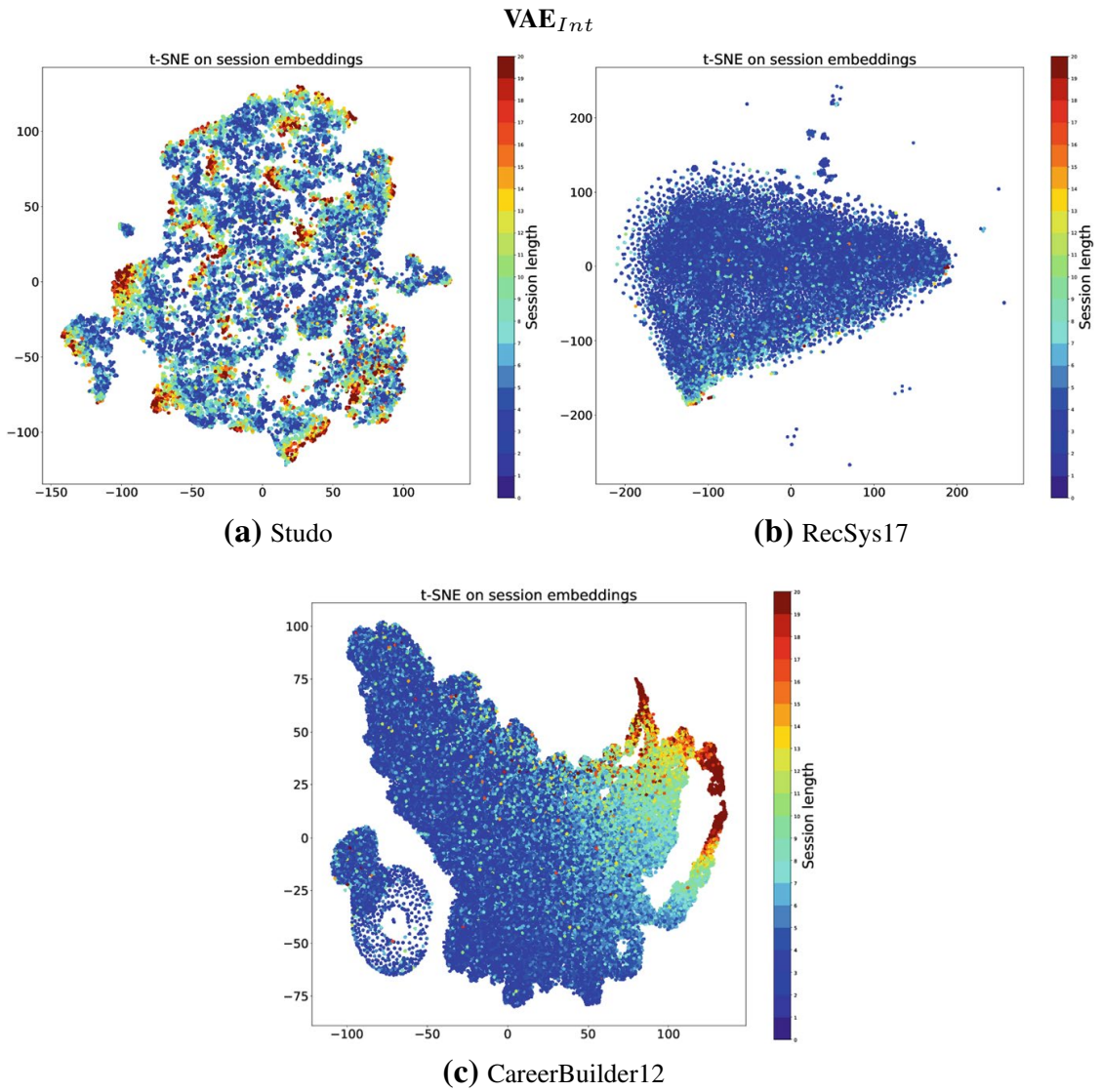
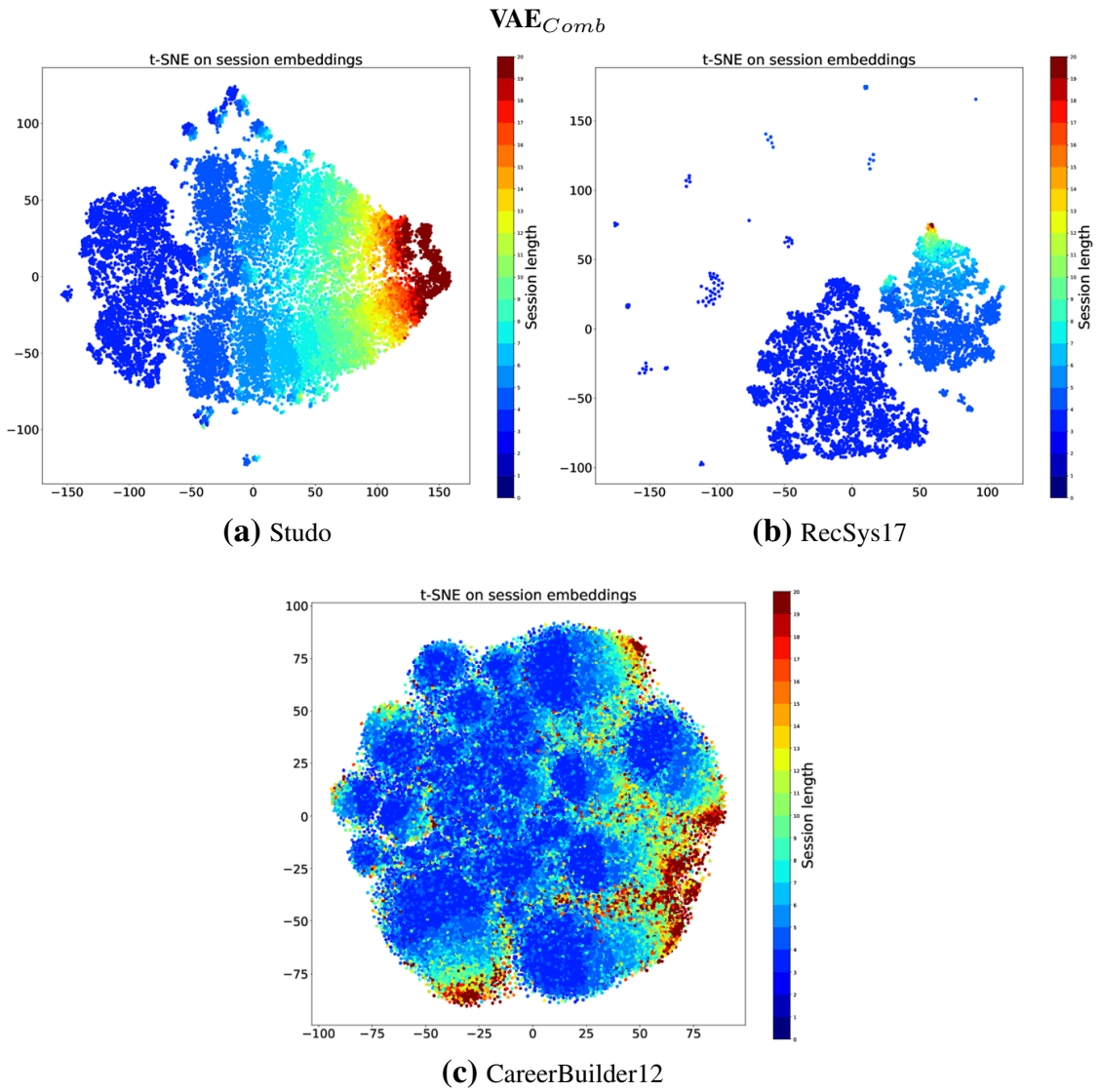


Fig. 6 The plots show t-SNE embeddings for latent session representations produced with the VAE autoencoder models trained on all three datasets using only interaction data. The colors of the sessions reflect the session length, where the same red color is used for sessions with 20 or more interactions

dimensionality of the latent session representations and lets us explore embeddings in a 2*D* space. In t-SNE plots, similar items are modeled by neighboring points with high probability. In our case, we expect similar sessions to form clusters of neighboring points in the 2*D* space.

Figures 6 and 7 show the variational autoencoder models as t-SNE plots for all three datasets, i.e., VAE$_{Int}$ trained on interactions and VAE$_{Comb}$ trained on interactions combined with job content (see "Appendix B" for a more detailed embedding analysis). In the case of the smallest dataset Studo, when we train the autoencoder only on interactions, more clusters are produced with sessions of different sizes close to each other (e.g., Fig. 6a). If the variational autoencoders are additionally trained on the job content, we can observe rainbow-colored shapes that are based on session length (e.g., as shown in Fig. 7a, b). In the larger CareerBuilder12 dataset, we end up with several sub-clusters that exhibit



(a) Studo **(b)** RecSys17

(c) CareerBuilder12

Fig. 7 The plots show t-SNE embeddings for latent session representations produced with the VAE autoencoder models trained on all three datasets using interaction data combined with the job content. The colors of the sessions reflect the session length, where the same red color is used for sessions with 20 or more interactions

this rainbow pattern. In other words, when we encode the input with content features, sessions of similar length tend to cluster. We attribute this to sessions of similar length having similar patterns of input vectors (e.g., many right-padded zeros for short sessions).

Next, we investigate the difference in recommendation accuracy between VAE$_{Int}$ and VAE$_{Comb}$ in light of the clustering patterns. The results suggest that when sessions cluster by similar size in the 2D space, as in the case of VAE$_{Comb}$ in the Studo and RecSys17 datasets and VAE$_{Int}$ in the CareerBuilder12 dataset, recommendation accuracy drops.

Table 3 Best performing hyperparameter settings for each evaluated baseline approach and dataset based on nDCG@20

Approach	Parameter	Studo	RecSys17	CareerBuilder12
BPR	$\lambda_{SESSION}$	0.25	0	0
	λ_{ITEM}	0.25	0	0
iKNN	λ	80	50	20
	α	0.75	0.75	0.75
sKNN	k	100	500	1000
	SAMPLING	Recent	Random	Random
	SIMILARITY	Cosine	Cosine	Jaccard
	POPULARITY BOOST	No	No	Yes
S-sKNN	k	100	500	1000
	SAMPLING	Recent	Random	Random
	SIMILARITY	Cosine	Jaccard	Cosine
	POPULARITY BOOST	No	No	Yes
V-sKNN	k	100	100	100
	SAMPLING	Recent	Random	Random
	SIMILARITY	Cosine	Cosine	Cosine
	POPULARITY BOOST	No	No	No
	WEIGHTING	Quadratic	Quadratic	Logarithmic
GRU4Rec	LOSS	top1-max	bpr-max-0.5	top1-max
	LAYERS	[100]	[100]	[1000]
	DROPOUT	0.2	0.2	0.2
	BATCH SIZE	32	32	32
pRNN	ACTIVATION	tanh	tanh	softmax
	LAYERS	[1000]	[100]	[1000]
	α	0.001	0.01	0.001
	BATCH SIZE	512	512	512

5.3 Hyperparameter optimization of the baseline approaches

We conducted a grid search on the hyperparameters for the baseline approaches using the validation set, i.e., two weeks of user interactions. As such, Table 3 reports on the best performing configurations for each approach and dataset in terms of recommendation accuracy (see "Appendix A" for more details).

BPR We performed a grid search that includes three different values for the regularization of session features $\lambda_{SESSION} \in \{0.0, 0.25, 0.5\}$ and the regularization of item feature $\lambda_{ITEM} \in \{0.0, 0.25, 0.5\}$.

iKNN For the iKNN approach, we evaluated the values for regularization (i.e., to avoid coincidental high similarities of rarely visited items) $\lambda \in \{20, 50, 80\}$ and the normalization factor for the support between two items $\alpha \in \{0.25, 0.5, 0.75\}$.

sKNN, S-sKNN and V-sKNN For all the sKNN variations that we utilize in this paper, we conducted a grid search for the parameter k (i.e., 100, 200, 500 or 1000),

Table 4 Prediction results ($k = 20$) of remaining jobs that will be subject to interaction within a session. (Color table online)

Studo						
	nDCG	MRR	EPC	EPD	Coverage (%)	
VAE_{Int}	.2724	.1636	.0214	.0174	70.12	100
VAE_{Comb}	.2593	.1597	.0198	.0162	82.27	100
sKNN	.2552	.1403	.0193	.0165	55.18	99.67
V-sKNN	.2766	.1679	.0209	.0178	60.67	100
S-sKNN	.2687	.1532	.0203	.0172	56.26	99.67
GRU4Rec	.2909	.1682	.0230	.0202	53.74	99.02
pRNN	.0895	.0500	.0060	.0058	20.70	29.97
Bayes	.1560	.0758	.0138	.0144	72.73	100
iKNN	.1718	.0864	.0153	.0158	62.65	99.67
BPR-MF	.0700	.0485	.0059	.0028	40.59	69.71
POP	.0501	.0276	.0014	.0048	1.80	2.61
RecSys Challenge 2017						
	nDCG	MRR	EPC	EPD	Coverage (%)	
VAE_{Int}	.2307	.1962	.0468	.0070	35.82	90.68
VAE_{Comb}	.2299	.1948	.0466	.0068	35.84	90.53
sKNN	.2180	.1782	.0129	.0033	36.33	91.03
V-sKNN	.1931	.1521	.0115	.0033	36.24	88.93
S-sKNN	.2221	.1829	.0131	.0034	36.91	91.43
GRU4Rec	.1040	.0811	.0065	.0024	46.63	73.45
pRNN	.1024	.0593	.0003	.0021	7.87	11.17
Bayes	.0446	.0336	.0031	.0019	28.56	61.62
iKNN	.0565	.0414	.0038	.0024	35.04	70.59
BPR-MF	.2530	.1900	.0086	.0028	76.58	93.79
POP	.2294	.2278	.0001	.0047	0.13	0.30
CareerBuilder 2012						
	nDCG	MRR	EPC	EPD	Coverage (%)	
VAE_{Int}	.1072	.0393	.0194	.0160	18.20	96.98
VAE_{Comb}	.1242	.0438	.0209	.0177	16.04	96.62
sKNN	.1406	.0454	.0161	.0146	13.77	92.45
V-sKNN	.1458	.0566	.0173	.0154	16.06	95.67
S-sKNN	.1428	.0462	.0166	.0150	14.61	95.03
GRU4Rec	.1415	.0554	.0172	.0159	20.46	83.60
pRNN	.0005	.0002	.0001	.0001	0.02	0.14
Bayes	.0842	.0383	.0094	.0077	11.87	78.53
iKNN	.1386	.0577	.0164	.0144	16.61	90.13
BPR-MF	.0005	.0001	.0001	.0001	78.75	95.36
POP	.0004	.0001	.0001	.0001	0.01	0.08

Table 4 (continued)

Coverage is reported for the ratio of recommended jobs compared to all jobs available in the data set (left) and jobs expected in the test set (right)

the sampling method of sessions (i.e., recent or random), the similarity function (i.e., cosine or Jaccard) and if popular items from neighboring sessions should be boosted. For V-sKNN, we also optimized the decay weighting function (i.e., division, logarithmic or quadratic).

GRU4Rec In the case of GRU4Rec, we experimented with two different loss functions {*top*1-*max*, *bpr-max*-0.5}, four variations of the number of GRU layers and their sizes {[100], [100, 100], [1000], [1000, 1000]}, a dropout applied to the hidden layer of {0.0, 0.2, 0.5} and batch sizes of {32, 128, 512}.

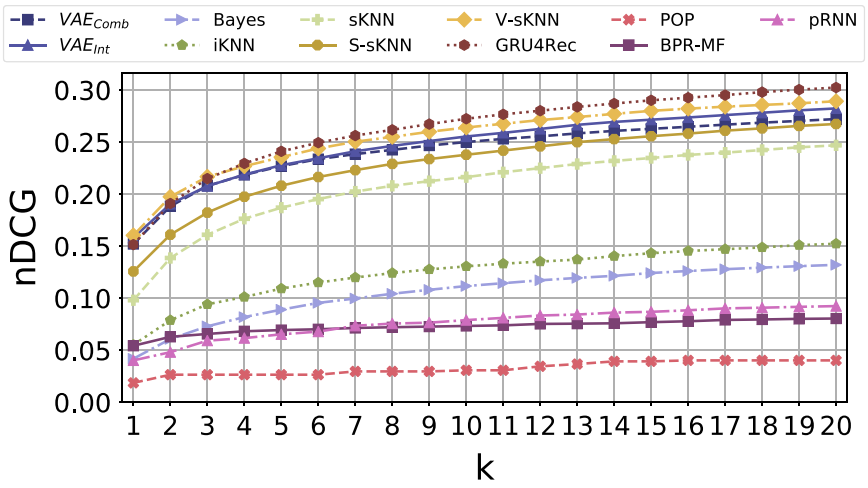
pRNN For the pRNN approach, we explored two activation functions {*softmax*, *tanh*} for the output layer, two sizes for the GRU layers {[100], [1000]}, a learning rate $\alpha \in \{0.01, 0.001\}$ and batch sizes of {32, 128, 512}. With respect to the batch size, however, due to the computational complexity of pRNN and the size of CareerBuilder12, we were only able to tune this hyperparameter for Studo and RecSys17. As we received the best results for a batch size of 512 for both datasets, we also used a batch size of 512 in case of CareerBuilder12.

5.4 Comparison with baseline approaches

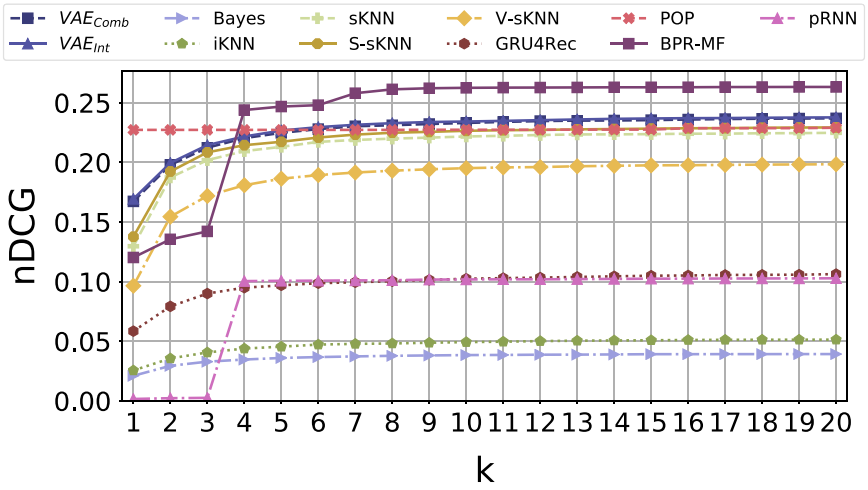
Table 4 shows the results of comparing VAE_{Int} and VAE_{Comb} with all baseline methods when we evaluate against the remaining jobs in the session. We report recommendation accuracy in terms of nDCG and MRR, as well as a system-based novelty (EPC), session-based novelty (EPD) and coverage. In the case of the next job prediction problem, in Figs. 8 and 9, we show nDCG and EPC results for different values of k (i.e., number of recommended jobs).

Accuracy (nDCG & MRR) On all datasets, the *sKNN*-based approaches achieve high accuracy in terms of nDCG and MRR, as shown in Table 4. In terms of both nDCG and MRR, VAE_{Int} performs second best in RecSys17, while it performs third best in Studo. For the Studo dataset, *GRU4Rec* has the highest accuracy for both metrics. In the RecSys17 dataset, *BPR-MF* performs best concerning nDCG, while *POP* performs best in terms of MRR. In CareerBuilder12, *V-sKNN* achieves the highest nDCG, while *iKNN* achieves the highest MRR. In this dataset, VAE_{Int} achieves medium performance, which we attribute to the ample item space and sparsity of CareerBuilder12. The VAE_{Comb} method, however, results in a higher recommendation accuracy, while training the model is much less expensive.

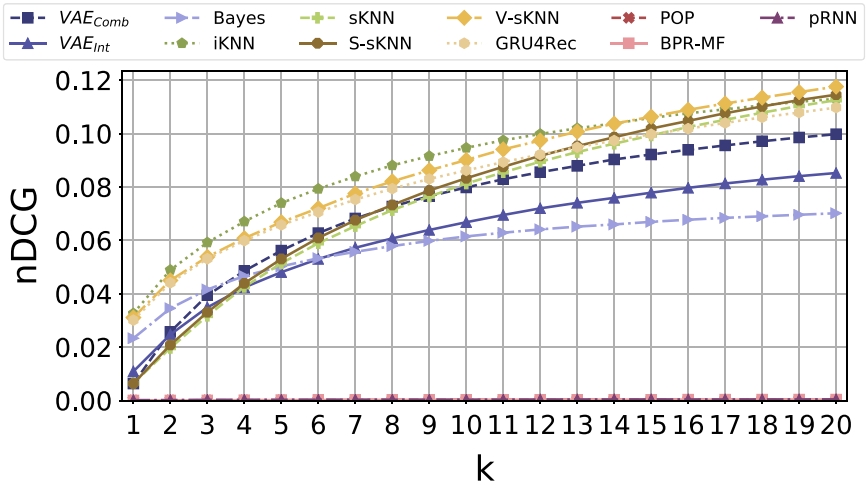
While the performance of *sKNN*-based approaches is rather stable, several baselines algorithms, namely *POP*, *BPR-MF*, *iKNN*, *Bayes*, *GRU4Rec* and *pRNN*, show



(a) Studo

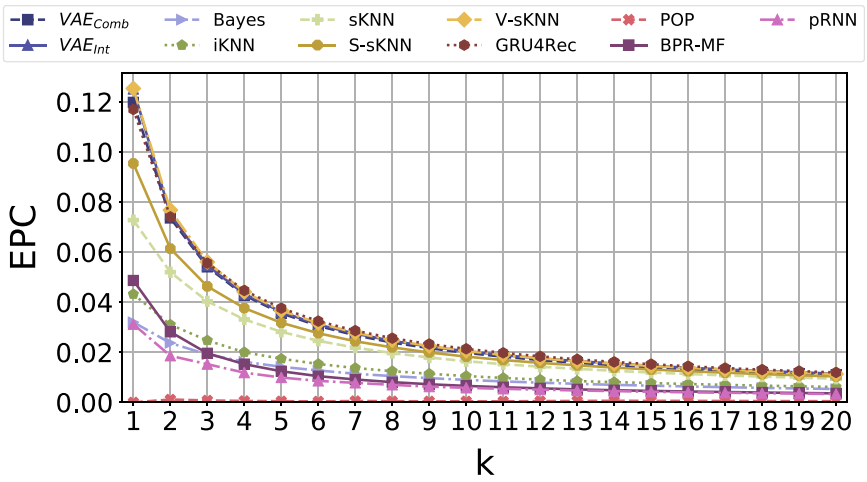


(b) RecSys Challenge 2017

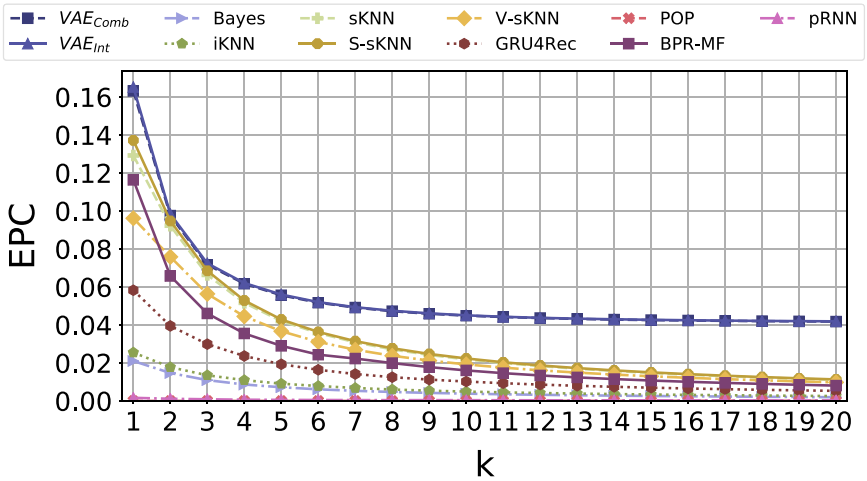


(c) CareerBuilder 2012

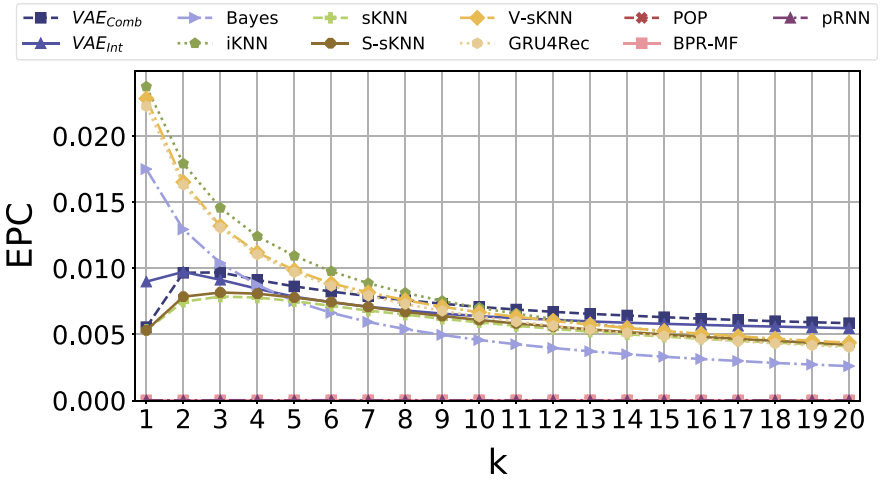
Fig. 8 nDCG results for different recommendation list sizes (i.e., values of k) when predicting the next job in the session. On all three datasets, both our proposed *VAE* approaches achieve competitive results concerning accuracy (i.e., nDCG) metrics



(a) Studo



(b) RecSys Challenge 2017



(c) CareerBuilder 2012

Fig. 9 EPC results for different recommendation list sizes (i.e., values of k) when predicting the next job in the session. On all three datasets, both our proposed *VAE* approaches achieve good results concerning beyond-accuracy (i.e., EPC) metrics

notable differences among the datasets. First, the *Bayes* approach establishes itself as a competitive baseline in the Studo dataset, whereas it results in a poor performance for the two larger datasets (i.e., RecSys17 and CareerBuilder12). In fact, for the RecSys17 dataset, it results in the worst performance. Hence, when the domain has a small number of items, it can be reasonable to employ such a simple and computationally inexpensive method.

Second, the accuracy of *POP* in the RecSys17 dataset is noteworthy.[14] The reason for this is that in the RecSys17 dataset, the most popular job from the train set was also the one with the highest number of interactions in the test set (i.e., around 21.5%). However, this approach will likely not result in high user satisfaction, just by predicting the same items repeatedly. Moreover, the *BPR-MF* performs best in terms of nDCG in the RecSys17 dataset, but it has the second worst performance in the other two datasets. Also, *GRU4Rec* performs worse for the RecSys17 dataset when compared to Studo and CareerBuilder12. We attribute this to bias toward popularity (Ludewig and Jannach 2018). The performance of *GRU4Rec* is low, while the performance of *BPR-MF* is high in the RecSys17 dataset. The *pRNN* method performs low on all three datasets, but its recommendation accuracy is especially weak on the CareerBuilder12 dataset. Finally, the performance of the *iKNN* differs among all three datasets. While it has the highest MRR for the CareerBuilder12 dataset, the performance in the RecSys17 dataset is the second lowest for both accuracy metrics.

For the next job prediction problem shown in Fig. 8, in all three datasets, all approaches show a similar accuracy performance. The results confirm the presence of bias toward popular items in the RecSys17 dataset as the popularity approach outperforms the other algorithms until $k = 3$, after which *BPR-MF* becomes the best performing approach. We also attribute the sudden increase in the nDCG values for *BPR-MF* and *pRNN* at the recommendation list of length 4 to this popularity bias in the dataset. A closer inspection revealed that both approaches often recommend highly popular items from the train set at the beginning of the recommendation list. The top-1 (i.e., most popular) item that is shared between the train and test set is also the one which gets recommended most frequently as the fourth item in the recommendation list of *BPR-MF* and *pRNN*. Besides that, for all values of k (i.e., the number of recommended jobs), the session-based KNN approaches and *GRU4Rec* achieve competitive accuracy values.

System-based novelty (EPC) As shown in Table 4, both *VAE* approaches achieve top results in terms of EPC for all three datasets. VAE_{Int} performs best on the RecSys17 dataset, while VAE_{Comb} outperforms all approaches in the CareerBuilder12 dataset. In the Studo dataset, VAE_{Int} achieves second best to *GRU4Rec*. Especially in the RecSys17 dataset, the difference in novelty is considerably high when compared to other baselines. For the baselines, the *sKNN* approaches and *GRU4Rec* both exhibit a good performance concerning the novelty of the recommended jobs. The *pRNN* method, as well as *POP* and *BPR-MF*, produces recommendations that have the lowest system-based novelty.

[14] Quadrana et al. (2017) report that their popularity approach outperforms session-based RNN (Hidasi et al. 2015) in the XING dataset used in the ACM RecSys Challenge 2016.

Table 5 Summary of the rankings of the session-based algorithms evaluated in the job domain. (Color table online)

	Accuracy	Beyond Accuracy	Coverage
VAE_{Int}	++	++	++
VAE_{Comb}	+	++	++
sKNN	+	o	+
V-sKNN	++	+	++
S-sKNN	++	+	+
GRU4Rec	++	+	+
pRNN	- -	- -	- -
Bayes	- -	- -	o
iKNN	o	-	+
BPR-MF	-	- -	++
POP	- -	- -	- -

"++" indicates best, "+" good, "o" average, "-" low and "- -" the worst ranking with respect to (1) accuracy (i.e., nDCG and MRR), (2) beyond-accuracy (i.e., EPC and EPD) and (3) coverage

In Fig. 9, we see that both our proposed *VAE* approaches outperform all others in the CareerBuilder12 dataset after $k = 9$. The *sKNN* baselines, as well as *GRU4Rec*, show a better novelty performance for a smaller number of recommended jobs.

Session-based novelty (EPD) As depicted in Table 4, both *VAE* approaches provide the best session-based novelty for the RecSys17 and CareerBuilder12 datasets and are competitive in the Studo dataset. The VAE_{Comb} method generates the most *surprising* recommendations in the largest dataset (i.e., CareerBuilder12) and *GRU4Rec* in the smallest dataset (i.e., Studo). In all cases, the *sKNN*-based approaches are a competitive baseline. We can observe the most notable difference between accuracy and EPD, however, in the CareerBuilder12 dataset, where the *VAE* approaches result in a rather average accuracy while performing very well concerning session-based novelty. Overall, the results indicate that both *VAE* approaches are suitable for cases when we aim to generate novel session-based recommendations.

Coverage In Table 4, we report the percentage of jobs, which were recommended and are a part of (1) all jobs available in the dataset (i.e., the complete item catalog), and (2) the jobs that we know anonymous session users will interact within the test set (i.e., the expected item catalog).

In terms of the coverage of all possible job postings, VAE_{Comb} performs best in the Studo dataset. *BPR-MF* covers at the most the entire item catalog in the RecSys17 and CareerBuilder12 dataset. Concerning the coverage of items in the test set (i.e., expected items), the session-based KNN approaches achieve almost perfect coverage in the Studo dataset. Only in the case of the RecSys17 dataset, the

BPR-MF baseline has an even higher coverage. As expected, the *POP* baseline results in the worst coverage. While this baseline has high accuracy values in the RecSys17 dataset (due to the popularity bias inherent in this dataset), it effectively covers only a small fraction of jobs in the system. It also has to be noted that the *pRNN* baseline always has the second-worst coverage. As the available item catalog grows, the coverage drops, which suggests that the trained model focuses on a specific (i.e., relatively small) set of items, which explains the worse performance in the largest dataset (i.e., CareerBuilder12).

5.5 Performance overview

To provide a better overview of the performance of the different session-based job recommendation approaches, we summarize all results in Table 5 with respect to three metric categories. That is, we report the performance on accuracy (i.e., *nDCG* and *MRR*), beyond accuracy (i.e., *EPC* and *EPD*) and coverage (of the whole dataset and the test set). For every approach, we assign a rank (i.e., from 1 to 11) for the particular metric in a dataset. We then aggregate these rankings across all three metric categories and datasets. The final rankings are then normalized and assigned into five performance buckets (i.e., from worst "- -" to best "++"; see "Appendix C" for the calculation steps).

Concerning accuracy, the best performance is achieved by *V-sKNN*, our VAE_{Int} variant, *S-sKNN* and *GRU4Rec*. This is then followed by VAE_{Comb} and *sKNN*. All other baselines achieve worse accuracy. For the beyond accuracy metric category, both of our *VAE* variants achieve the best performance. This is followed by *GRU4Rec* and the *sKNN* variants. A similar observation can be made for the metric category coverage. Here, however, *BPR-MF* also shows the best, *iKNN* good and the simple *Bayes* baseline medium coverage. Noteworthy is also the ranking score of the VAE_{Comb}, as with our proposed method it is possible to train the autoencoder models faster (i.e., even with a large item space) and without the need to frequently retrain the utilized model to consider new jobs coming to the system. The *pRNN* approach did not achieve a good rank in any metric category. The same is true for *POP*.

6 Conclusion and future work

In this work, we addressed the problem of providing job recommendations in an anonymous, online session setting. In three datasets, i.e., Studo, RecSys17 and CareerBuilder12, we evaluated the efficacy of using different autoencoder architectures to produce session-based job recommendations. Specifically, we utilized autoencoders to infer latent session representations, which are used in a k-nearest neighbor manner to recommend jobs within a session. We evaluated two types of

input for the autoencoders: (1) interactions with job postings within browsing sessions and (2) a combination of interactions with job postings and content features extracted from these job postings.

We found that variational autoencoders trained on interaction and content data, and used in a *k*-nearest neighbor manner, led to very good results in terms of accuracy compared to other autoencoder variants. A visual analysis of the embedding spaces with t-SNE revealed that we could attribute a lower accuracy performance when similar-sized sessions form clusters in the 2D space. Although this was mostly the case for autoencoders trained on content features, in practice, however, such an approach has the advantage of fixed size vectors, which means retraining is needed less often. Consequently, depending on the application scenario, one can decide which input for the variational autoencoder to take, i.e., to balance frequent retraining and accuracy.

Furthermore, we evaluated all autoencoder and baseline approaches with respect to beyond-accuracy metrics, i.e., system-based and session-based novelty as well as coverage, in two settings: Firstly, we compared the recommendation performance of the approaches on all remaining interactions within a session, and secondly, we predicted the next job interaction in the session. We find that our proposed variational autoencoder methods can outperform state-of-the-art approaches for sessions-based recommender systems with respect to system-based and session-based novelty. Besides, the session-based KNN approaches are a competitive baseline for the variational autoencoder methods with respect to accuracy and coverage.

For future work, we aim to explore the use of generative variational autoencoder models to directly recommend jobs from the reconstructed session vector (e.g., in a similar way as in Liang et al. 2018). Other ideas for future work include investigating all approaches used in this study in an online evaluation. We plan to conduct an online study to ask users how satisfied and surprised they are with job recommendations generated by autoencoders. Also, we plan to evaluate the accuracy in an A/B test to conclude whether a higher system-based and session-based novelty in a session-based offline setting leads to higher user satisfaction. Additionally, we also plan to directly optimize for the beyond-accuracy metrics by incorporating re-ranking techniques (e.g., maximum marginal relevance Carbonell and Goldstein 1998). These evaluations are planned to be carried out in the Talto [15] career platform. In summary, we hope that the approach presented in this paper will attract further research on the effectiveness of dimensionality reduction techniques

[15] Talto (https://talto.com) is the successor of the jobs platform in Studo (http://www.studo).

for session-based job recommender systems and the effect of such methods on beyond-accuracy metrics such as system-based and session-based novelty as well as coverage.

Limitations Our work has several limitations. So far, we only focused on autoencoders to infer the latent representation of the anonymous user session. While autoencoders are a popular choice to reduce the dimensionality of data, other deep neural networks such as restricted Boltzmann machines (Nguyen et al. 2013), deep belief networks (Srivastava and Salakhutdinov 2012) or convolutional neural networks (Shen et al. 2014) could also serve well for this task. Furthermore, additional metadata information about jobs (e.g., textual content of job postings) could potentially enhance recommendations, which we did not tackle due to the unavailability of such data in all datasets. So far, we did not compare the approaches used in this study concerning computational performance, like the authors of (Ludewig and Jannach 2018) did. Moreover, in this work, we did not investigate how to model repeated interactions on the same job postings. Although this is implicitly considered by the autoencoder variants that combine interactions with job content features, such actions are not taken into account by the autoencoders that solely rely on interaction data. Also, in this work, we extracted the candidate sessions based on the last job interaction, which is a limitation of our work. For the evaluation procedure, we used a single time-based split for our experiments. One approach to assess the robustness of our results would be to apply a sliding window approach to generate splits with varying lengths. However, the size of the Studo and RecSys17 datasets is limited, which makes such an approach infeasible. For the larger CareerBuilder dataset, a sliding-window-based evaluation approach could be applied to test the robustness of the method. Due to computational constraints, for the present work, we used the same time-based split as for the Studo and the RecSys17 datasets. We leave the exploration of more splits to future work.

Another limitation is that one of the datasets we used for our study, the Studo dataset, is proprietary, and due to the terms of service of Moshbit, the owner of Studo, it cannot be made available for others at this point.

Acknowledgements Open access funding provided by Graz University of Technology. This work is supported by the Know-Center, the Institute of Interactive Systems and Data Science (ISDS) of Graz University of Technology and Moshbit. We thank Moshbit for granting access to their dataset in Studo Jobs. We thank Simone Kopeinik, Dieter Theiler, Tomislav Duricic and Leon Fadjevic for their feedback on this manuscript.

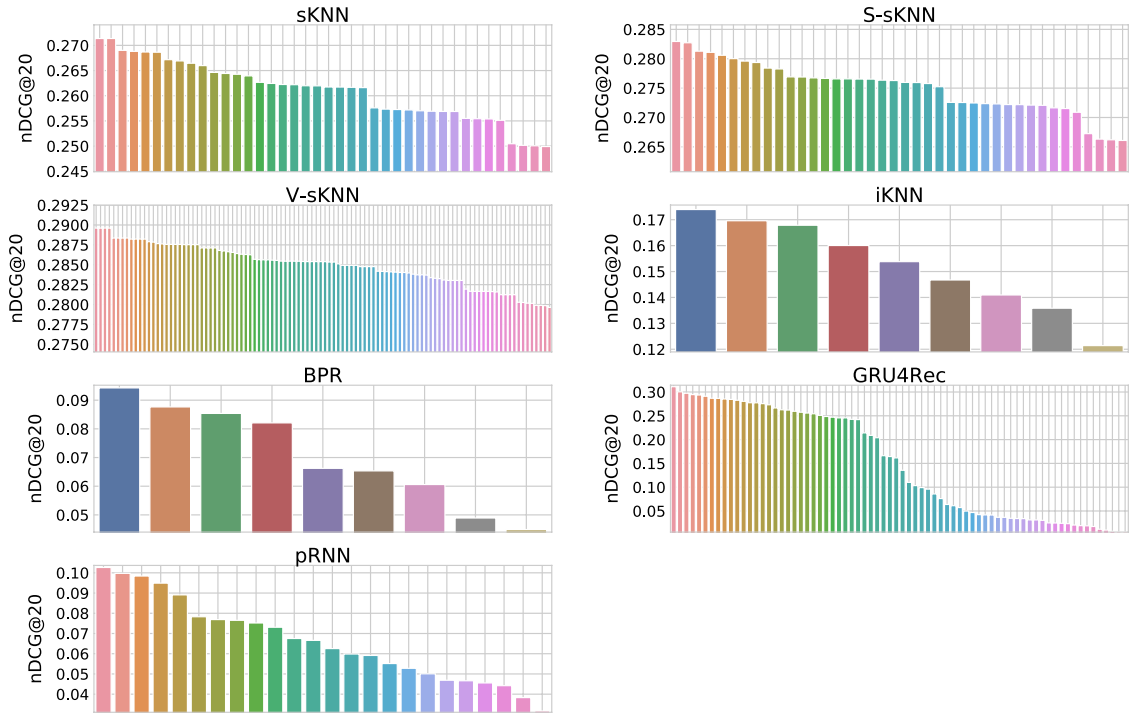


Fig. 10 Accuracy results for the different hyperparameters of the baseline approaches on the Studo dataset

Appendices

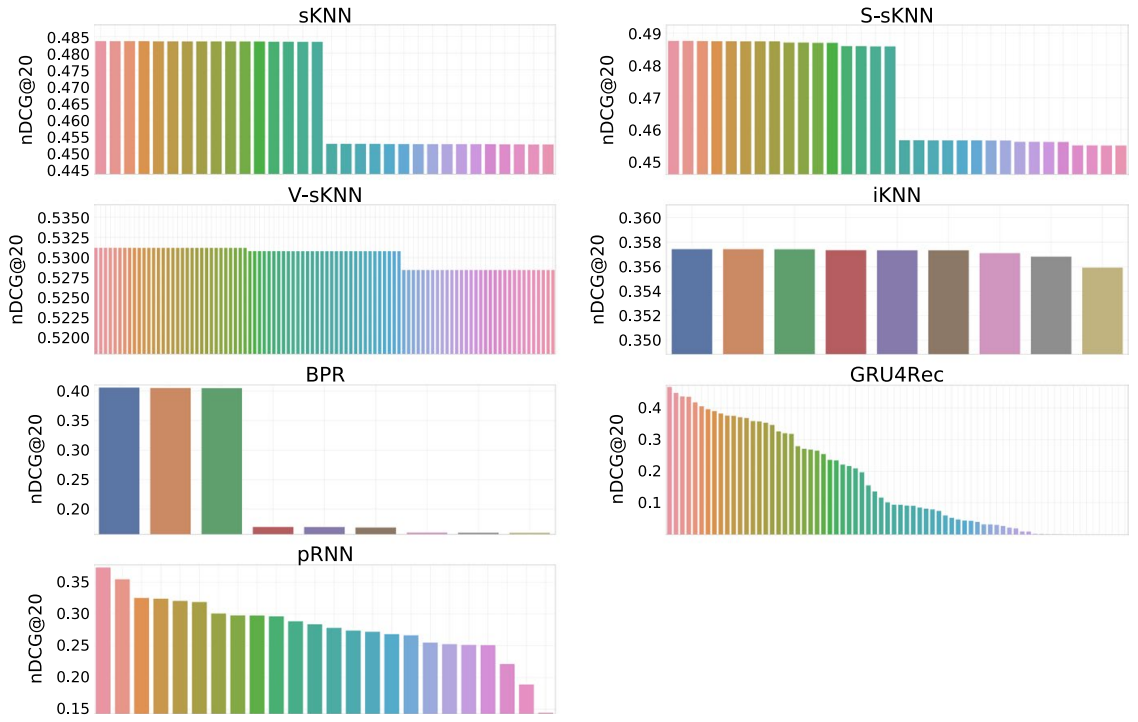


Fig. 11 Accuracy results for the different hyperparameters of the baseline approaches on the RecSys17 dataset

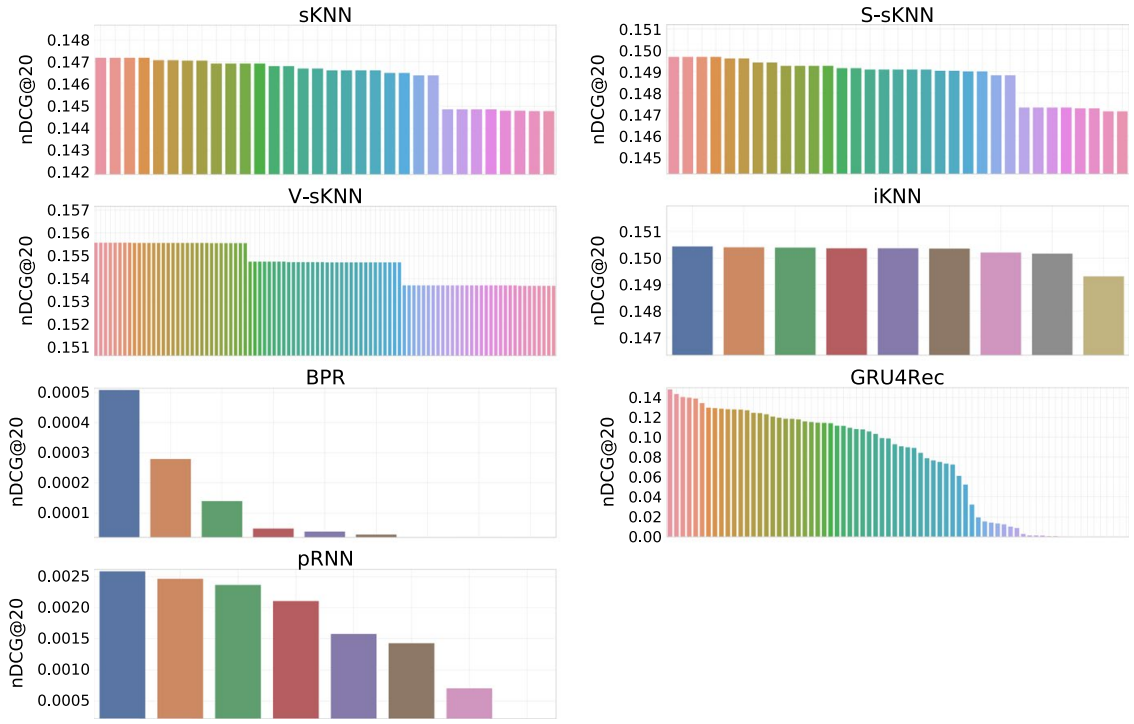


Fig. 12 Accuracy results for the different hyperparameters of the baseline approaches on the CareerBuilder12 dataset

Hyperparameter optimization results

In this section, we report the distribution of the accuracy results achieved by optimizing the hyperparameters for the baseline approaches in Sect. 5.3. For each baseline approach, we pick those hyperparameters which showed the best performance with respect to nDCG@20. As such, Fig. 10 shows the differences between the evaluated baseline configurations on the Studo dataset. Respectively, Fig. 11 depicts the results for the RecSys17 and Fig. 12 for the CareerBuilder12 dataset.

Autoencoder embedding analysis

Figure 13 shows all autoencoder models as t-SNE plots for the Studo dataset, i.e., AE_{Int}, DAE_{Int} and VAE_{Int} trained on interactions and AE_{Comb}, DAE_{Comb} and VAE_{Comb} trained on the combination of interactions and job content. The same is reported for RecSys17 in Fig. 14 and CareerBuilder12 in Fig. 15.

The results indicate that both denoising autoencoders and variational autoencoders tend to produce more session clusters than a classic autoencoder, which creates more of a linear pattern of neighboring sessions. In some cases, we can observe that both the classic and denoising autoencoder models produce shapes without clear structure and large dispersion (e.g., see Fig. 13d or 15b), which indicates that it is hard to find a clear neighborhood of similar sessions. For the smaller Studo dataset, if the autoencoders are solely trained on interactions, i.e., AE_{Int}, DAE_{Int} and VAE_{Int}, more clusters are produced with sessions of different

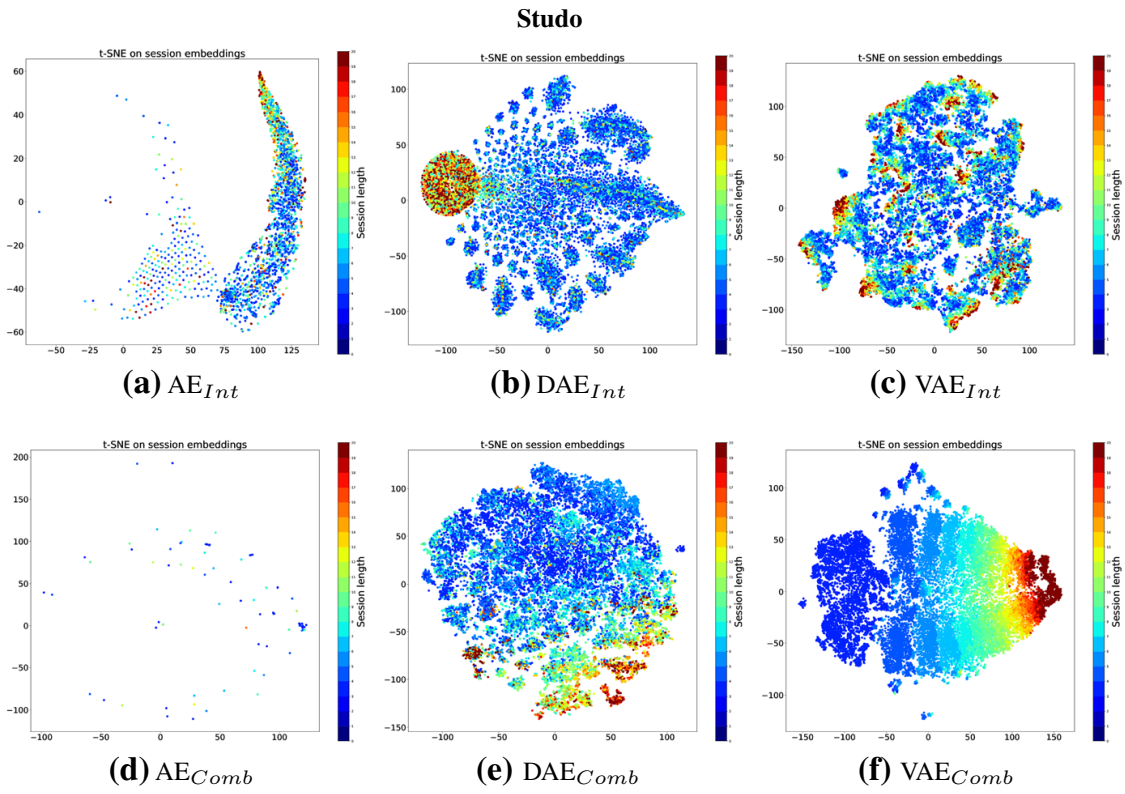


Fig. 13 t-SNE embeddings for latent session representations produced with the three autoencoder models trained on interaction and content data from the Studo dataset. Sessions are colored according to their length, where the same red color is used for sessions with 20 or more interactions

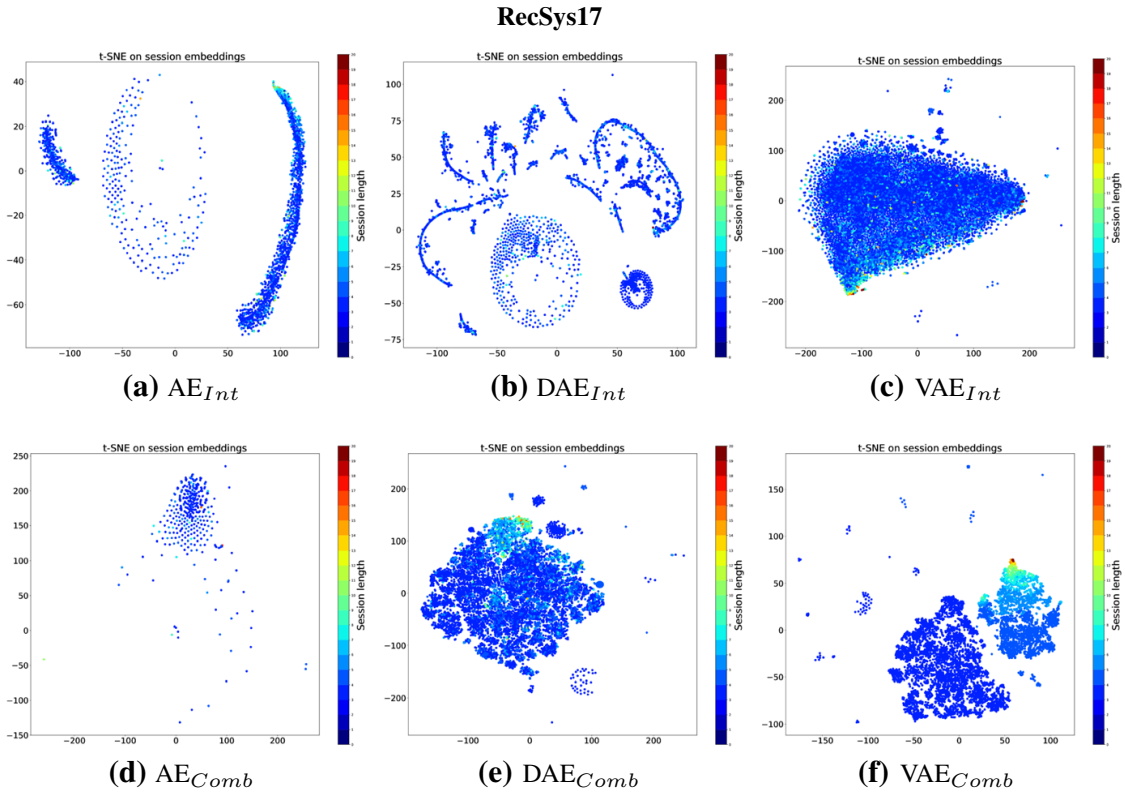


Fig. 14 t-SNE embeddings for latent session representations for the RecSys17 dataset

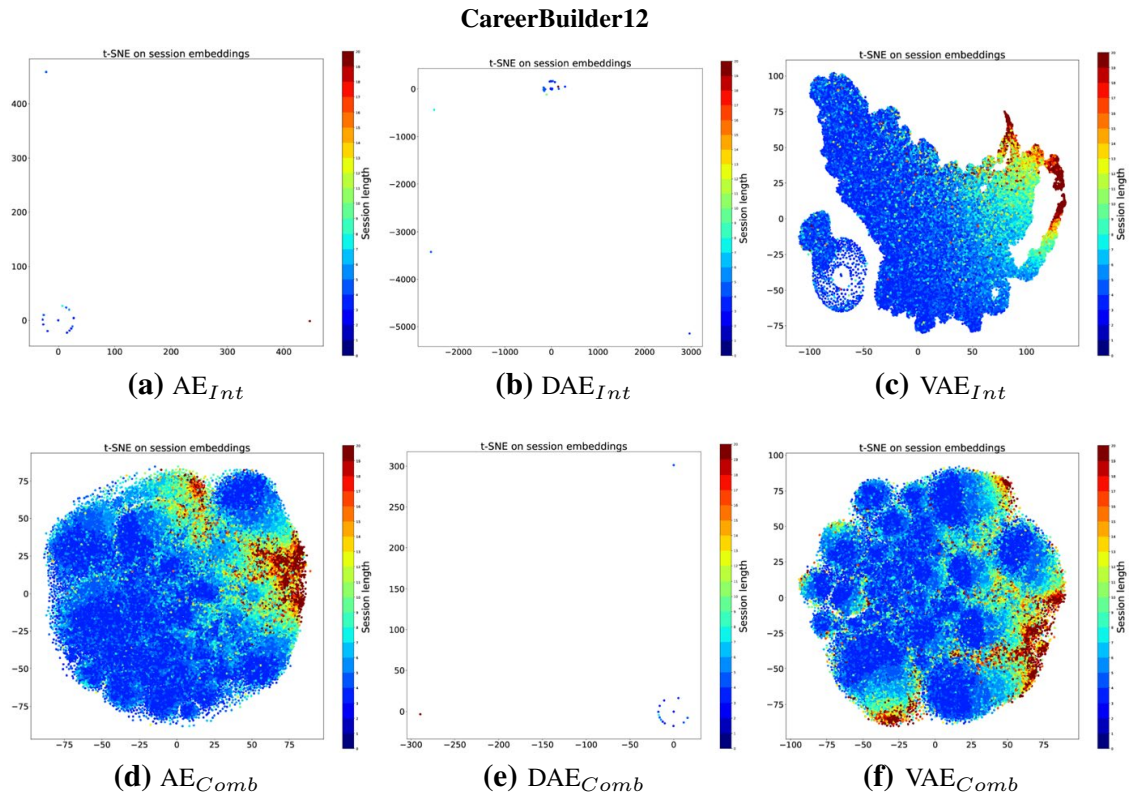


Fig. 15 t-SNE embeddings for latent session representations for the Careerbuilder12 dataset

sizes close to each other (e.g., Fig. 13b, c). Interestingly, if autoencoders are trained on content, we can observe rainbow-colored shapes that are based on session length (e.g., as shown in Fig. 13e, f). In case of a larger dataset like CareerBuilder12, we end up with several sub-clusters that exhibit this rainbow pattern. This shows that when we encode the input with content features, sessions of similar length tend to cluster. We attribute this to sessions of similar length having similar patterns of input vectors (e.g., many right-padded zeros for short sessions).

Aggregation of rankings

In Sect. 5.5, we report the aggregated performance of the different approaches. For this, in Table 6 we first rank the results from each dataset (i.e., based on Table 4). We then sum the rankings for each dataset (i.e., Studo, RecSys17 and CareerBuilder12) for the accuracy metrics (i.e., nDCG and MRR), the beyond-accuracy metrics (i.e., EPC and EPD) and both coverage, respectively. The aggregated rankings are outlined in Table 7. The rankings are then normalized with the equation

Table 6 Ranking of the results per metric and dataset, which are derived from numerical results. (Color table online)

Studo						
	nDCG	MRR	EPC	EPD	Coverage (%)	
VAE_{Int}	3	3	2	3	3	1
VAE_{Comb}	5	4	5	6	1	1
sKNN	6	6	6	5	7	5
V-sKNN	2	2	3	2	5	1
S-sKNN	4	5	4	4	6	5
GRU4Rec	1	1	1	1	8	8
pRNN	9	9	9	9	10	10
Bayes	8	8	8	8	2	1
iKNN	7	7	7	7	4	5
BPR-MF	10	10	10	11	9	9
POP	11	11	11	10	11	11
RecSys Challenge 2017						
	nDCG	MRR	EPC	EPD	Coverage (%)	
VAE_{Int}	2	2	1	1	7	4
VAE_{Comb}	3	3	2	2	6	5
sKNN	6	6	4	5	4	3
V-sKNN	7	7	5	5	5	6
S-sKNN	5	5	3	4	3	2
GRU4Rec	8	8	7	8	2	7
pRNN	9	9	10	10	10	10
Bayes	11	11	9	11	9	9
iKNN	10	10	8	8	8	8
BPR-MF	1	4	6	7	1	1
POP	4	1	11	3	11	11
CareerBuilder 2012						
	nDCG	MRR	EPC	EPD	Coverage (%)	
VAE_{Int}	7	7	2	2	3	1
VAE_{Comb}	6	6	1	1	6	2
sKNN	4	5	7	6	8	6
V-sKNN	1	2	3	4	5	3
S-sKNN	2	4	5	5	7	5
GRU4Rec	3	3	4	3	2	8
pRNN	9	9	9	9	10	10
Bayes	8	8	8	8	9	9
iKNN	5	1	6	7	4	7
BPR-MF	9	10	9	9	1	4
POP	11	10	9	9	11	11

Coloring is according to the rank within each dataset

Table 7 Aggregated rankings across the three different datasets and per metric type (i.e., accuracy, beyond accuracy and coverage)

	Accuracy		Beyond Accuracy		Coverage	
	Aggregated	Normalized	Aggregated	Normalized	Aggregated	Normalized
VAE_{Int}	24	0.1176	11	0.0217	19	0.0208
VAE_{Comb}	27	0.2059	17	0.1522	21	0.0625
sKNN	33	0.3824	33	0.5000	33	0.3125
V-sKNN	21	0.0294	22	0.2609	25	0.1458
S-sKNN	25	0.1471	25	0.3261	28	0.2083
GRU4Rec	24	0.1176	24	0.3043	35	0.3542
pRNN	54	1.0000	56	1.0000	60	0.8750
Bayes	54	1.0000	52	0.9130	39	0.4375
iKNN	40	0.5882	43	0.7174	36	0.3750
BPR-MF	44	0.7059	52	0.9130	25	0.1458
POP	48	0.8235	53	0.9348	66	1.0000

Results are then normalized by a min-max scaling

$Norm(x) = \frac{x-min+1}{max-min+1}$, where *min* is the lowest aggregated rank and *max* is the highest aggregated rank. Thus, lower results are considered better, while the worst results receive the value 1. The results are then put into five buckets according to their values. A double plus (i.e., ++) is assigned to values between 0.0 and 0.2, while values between 0.2 and 0.4 get assigned a single plus (i.e., +), followed by *o* (i.e., 0.4 until 0.6), − (i.e., 0.6 until 0.8) and for the worst results a −− (i.e., 0.8 until 1).

References

Abel, F.: We know where you should work next summer: job recommendations. In: Proceedings of the 9th ACM Conference on Recommender Systems, pp. 230–230 (2015)

Abel, F., Benczúr, A., Kohlsdorf, D., Larson, M., Pálovics, R.: Recsys challenge 2016: job recommendations. In: Proceedings of the 10th ACM Conference on Recommender Systems, pp. 425–426. ACM (2016)

Abel, F., Deldjoo, Y., Elahi, M., Kohlsdorf, D.: Recsys challenge 2017: offline and online evaluation. In: Proceedings of the Eleventh ACM Conference on Recommender Systems, pp. 372–373 (2017)

Adomavicius, G., Kwon, Y.: Improving aggregate recommendation diversity using ranking-based techniques. IEEE Trans. Knowl. Data Eng. **24**(5), 896–911 (2012)

Aggarwal, C.C.: Evaluating recommender systems. In: Recommender Systems, pp. 225–254. Springer (2016)

Al-Otaibi, S.T., Ykhlef, M.: A survey of job recommender systems. Int. J. Phys. Sci. **7**(29), 5127–5142 (2012)

Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Advances in Neural Information Processing Systems, pp. 153–160 (2007)

Bianchi, M., Cesaro, F., Ciceri, F., Dagrada, M., Gasparin, A., Grattarola, D., Inajjar, I., Metelli, A.M., Cella, L.: Content-based approaches for cold-start job recommendations. In: Proceedings of the Recommender Systems Challenge 2017, p. 6. ACM (2017)

Bonnin, G., Jannach, D.: Automated generation of music playlists: survey and experiments. ACM Comput. Surv. (CSUR) **47**(2), 26 (2015)

Carbonell, J., Goldstein, J.: The use of mmr, diversity-based reranking for reordering documents and producing summaries. In: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 335–336 (1998)

Chatzis, S.P., Christodoulou, P., Andreou, A.S.: Recurrent latent variable networks for session-based recommendation. In: Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems, pp. 38–45. ACM (2017)
Fischer, A., Igel, C.: An introduction to restricted Boltzmann machines. In: Iberoamerican Congress on Pattern Recognition, pp. 14–36. Springer (2012)
He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S.: Neural collaborative filtering. In: Proceedings of the 26th International Conference on World Wide Web, pp. 173–182. International World Wide Web Conferences Steering Committee (2017)
Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. (TOIS) **22**(1), 5–53 (2004)
Hidasi, B., Karatzoglou, A.: Recurrent neural networks with top-k gains for session-based recommendations. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, pp. 843–852. ACM (2018)
Hidasi, B., Karatzoglou, A., Baltrunas, L., Tikk, D.: Session-based recommendations with recurrent neural networks. arXiv preprint arXiv:1511.06939 (2015)
Hidasi, B., Quadrana, M., Karatzoglou, A., Tikk, D.: Parallel recurrent neural network architectures for feature-rich session-based recommendations. In: Proceedings of the 10th ACM Conference on Recommender Systems, pp. 241–248. ACM (2016)
Hidasi, B., Tikk, D.: General factorization framework for context-aware recommendations. Data Min. Knowl. Discov. **30**(2), 342–371 (2016)
Hong, W., Zheng, S., Wang, H., Shi, J.: A job recommender system based on user clustering. JCP **8**(8), 1960–1967 (2013)
Jannach, D., Ludewig, M.: When recurrent neural networks meet the neighborhood for session-based recommendation. In: Proceedings of the Eleventh ACM Conference on Recommender Systems, pp. 306–310 (2017)
Jordan, M.I., Ghahramani, Z., Jaakkola, T.S., Saul, L.K.: An introduction to variational methods for graphical models. Mach. Learn. **37**(2), 183–233 (1999)
Kamehkhosh, I., Jannach, D., Ludewig, M.: A comparison of frequent pattern techniques and a deep learning method for session-based recommendation. In: RecTemp@ RecSys, pp. 50–56 (2017)
Kenthapadi, K., Le, B., Venkataraman, G.: Personalized job recommendation system at linkedin: practical challenges and lessons learned. In: Proceedings of the Eleventh ACM Conference on Recommender Systems, pp. 346–347 (2017)
Kenthapadi, K., Le, B., Venkataraman, G.: Personalized job recommendation system at linkedin: practical challenges and lessons learned. In: Proceedings of the 11th ACM Conference on Recommender Systems, pp. 346–347 (2017)
Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint. arXiv:1312.6114 (2013)
Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
Kramer, M.A.: Nonlinear principal component analysis using autoassociative neural networks. AIChE J. **37**(2), 233–243 (1991)
Lacic, E., Kowald, D., Traub, M., Luzhnica, G., Simon, J., Lex, E.: Tackling cold-start users in recommender systems with indoor positioning systems. In: Poster Proceedings of the 9th ACM Conference on Recommender Systems (2015)
Lacic, E., Reiter-Haas, M., Duricic, T., Slawicek, V., Lex, E.: 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, pp. 496–500. ACM (2019)
Li, J., Ren, P., Chen, Z., Ren, Z., Lian, T., Ma, J.: Neural attentive session-based recommendation. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 1419–1428. ACM (2017)
Liang, D., Krishnan, R.G., Hoffman, M.D., Jebara, T.: Variational autoencoders for collaborative filtering. arXiv preprint. arXiv:1802.05814 (2018)
Lin, Z., Feng, M., Santos, C.N.d., Yu, M., Xiang, B., Zhou, B., Bengio, Y.: A structured self-attentive sentence embedding. arXiv preprint. arXiv:1703.03130 (2017)
Liu, Q., Zeng, Y., Mokhosi, R., Zhang, H.: Stamp: short-term attention/memory priority model for session-based recommendation. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1831–1839. ACM (2018)
Liu, R., Rong, W., Ouyang, Y., Xiong, Z.: A hierarchical similarity based job recommendation service framework for university students. Front. Comput. Sci. **11**(5), 912–922 (2017)

Ludewig, M., Jannach, D.: Evaluation of session-based recommendation algorithms. User Model. User-Adap. Inter. **28**(4–5), 331–390 (2018)
Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. J. Mach. Learn. Res. **9**(Nov), 2579–2605 (2008)
Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., Frey, B.: Adversarial autoencoders. arXiv preprint. arXiv:1511.05644 (2015)
Matuszyk, P., Vinagre, J., Spiliopoulou, M., Jorge, A.M., Gama, J.: Forgetting methods for incremental matrix factorization in recommender systems. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 947–953. ACM (2015)
McNee, S.M., Riedl, J., Konstan, J.A.: Being accurate is not enough: how accuracy metrics have hurt recommender systems. In: ACM CHI'06 (2006)
Mine, T., Kakuta, T., Ono, A.: Reciprocal recommendation for job matching with bidirectional feedback. In: 2013 Second IIAI International Conference on Advanced Applied Informatics, pp. 39–44. IEEE (2013)
Mishra, S.K., Reddy, M.: A bottom-up approach to job recommendation system. In: Proceedings of the Recommender Systems Challenge, p. 4. ACM (2016)
Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on International Conference on Machine Learning, pp. 807–814 (2010)
Nguyen, T.D., Tran, T., Phung, D., Venkatesh, S.: Learning sparse latent representation and distance metric for image retrieval. In: 2013 IEEE International Conference on Multimedia and Expo (ICME), pp. 1–6. IEEE (2013)
Parikh, A., Täckström, O., Das, D., Uszkoreit, J.: A decomposable attention model for natural language inference. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp. 2249–2255 (2016)
Parra, D., Sahebi, S.: Recommender systems: sources of knowledge and evaluation metrics. In: Advanced Techniques in Web Intelligence-2: Web User Browsing Behaviour and Preference Analysis, pp. 149–175. Springer (2013)
Pu, P., Chen, L., Hu, R.: A user-centric evaluation framework for recommender systems. In: Proceedings of the fifth ACM conference on Recommender systems, pp. 157–164 (2011)
Quadrana, M., Karatzoglou, A., Hidasi, B., Cremonesi, P.: Personalizing session-based recommendations with hierarchical recurrent neural networks. In: Proceedings of the Eleventh ACM Conference on Recommender Systems, pp. 130–137. ACM (2017)
Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: Bpr: Bayesian personalized ranking from implicit feedback. In: Proceedings of the Twenty-fifth Conference on Uncertainty in Artificial Intelligence, pp. 452–461. AUAI Press (2009)
Rezende, D.J., Mohamed, S., Wierstra, D.: Stochastic backpropagation and approximate inference in deep generative models. arXiv preprint. arXiv:1401.4082 (2014)
Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th International Conference on World Wide Web, pp. 285–295. ACM (2001)
Sedhain, S., Menon, A.K., Sanner, S., Xie, L.: Autorec: autoencoders meet collaborative filtering. In: Proceedings of the 24th International Conference on World Wide Web, pp. 111–112. ACM (2015)
Shani, G., Heckerman, D., Brafman, R.I.: An mdp-based recommender system. J. Mach. Learn. Res. **6**(Sep), 1265–1295 (2005)
Shen, Y., He, X., Gao, J., Deng, L., Mesnil, G.: A latent semantic model with convolutional-pooling structure for information retrieval. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, pp. 101–110. ACM (2014)
Siting, Z., Wenxing, H., Ning, Z., Fan, Y.: Job recommender systems: a survey. In: 2012 7th International Conference on Computer Science Education (ICCSE), pp. 920–924 (2012)
Smirnova, E., Vasile, F.: Contextual sequence modeling for recommendation with recurrent neural networks. arXiv preprint. arXiv:1706.07684 (2017)
Song, Y., Elkahky, A.M., He, X.: Multi-rate deep learning for temporal recommendation. In: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 909–912. ACM (2016)
Srivastava, N., Salakhutdinov, R.: Learning representations for multimodal data with deep belief nets. In: International Conference on Machine Learning Workshop, Vol. 79 (2012)
Strub, F., Gaudel, R., Mary, J.: Hybrid recommender system based on autoencoders. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, pp. 11–16. ACM (2016)

Tan, Y.K., Xu, X., Liu, Y.: Improved recurrent neural networks for session-based recommendations. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, pp. 17–22 (2016)
Theis, L., Shi, W., Cunningham, A., Huszár, F.: Lossy image compression with compressive autoencoders. arXiv preprint. arXiv:1703.00395 (2017)
Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop, coursera: neural networks for machine learning. University of Toronto, Technical Report (2012)
Tuan, T.X., Phuong, T.M.: 3d convolutional networks for session-based recommendation with content features. In: Proceedings of the Eleventh ACM Conference on Recommender Systems, pp. 138–146. ACM (2017)
Twardowski, B.: Modelling contextual information in session-aware recommender systems with neural networks. In: Proceedings of the 10th ACM Conference on Recommender Systems, pp. 273–276. ACM (2016)
Vargas, S., Castells, P.: Rank and relevance in novelty and diversity metrics for recommender systems. In: Proceedings of the Fifth ACM Conference on Recommender Systems, pp. 109–116. ACM (2011)
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th International Conference on Machine Learning, pp. 1096–1103. ACM (2008)
Volkovs, M., Yu, G.W., Poutanen, T.: Content-based neighbor models for cold start in recommender systems. In: Proceedings of the Recommender Systems Challenge 2017, p. 7. ACM (2017)
Voorhees, E.: Proceedings of the 8th Text Retrieval Conference. TREC-8 Question Answering Track Report, pp. 77–82 (1999)
Wu, S., Tang, Y., Zhu, Y., Wang, L., Xie, X., Tan, T.: Session-based recommendation with graph neural networks. arXiv preprint. arXiv:1811.00855 (2018)
Wu, S., Tang, Y., Zhu, Y., Wang, L., Xie, X., Tan, T.: Session-based recommendation with graph neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, pp. 346–353 (2019)
Wu, Y., DuBois, C., Zheng, A.X., Ester, M.: Collaborative denoising auto-encoders for top-n recommender systems. In: Proceedings of the Ninth ACM International Conference on Web Search and Data Mining, pp. 153–162. ACM (2016)
Xiao, W., Xu, X., Liang, K., Mao, J., Wang, J.: Job recommendation with hawkes process: an effective solution for recsys challenge 2016. In: Proceedings of the Recommender Systems Challenge, p. 11. ACM (2016)
Yuan, F., Karatzoglou, A., Arapakis, I., Jose, J.M., He, X.: A simple convolutional generative network for next item recommendation. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, pp. 582–590 (2019)
Zhang, C., Cheng, X.: An ensemble method for job recommender systems. In: Proceedings of the Recommender Systems Challenge, p. 2. ACM (2016)
Zhang, Y.C., Séaghdha, D.Ó., Quercia, D., Jambor, T.: Auralist: introducing serendipity into music recommendation. In: Proceedings of the Fifth ACM International Conference on Web Search and Data Mining, pp. 13–22. ACM (2012)
Zhou, T., Kuscsik, Z., Liu, J.G., Medo, M., Wakeling, J.R., Zhang, Y.C.: Solving the apparent diversity-accuracy dilemma of recommender systems. Proc. Natl. Acad. Sci. **107**(10), 4511–4515 (2010)

Emanuel Lacic is a Senior Researcher and Recommender Systems Architect in the Social Computing team at the Know-Center. He is a PhD student at Graz University of Technology and a former visiting researcher at the Computer Science department of the University of California, Los Angeles. He has an M.Sc. and B.Sc. in Software Engineering and Information Systems from the University of Zagreb. His research interests are in the fields of Recommender Systems, Deep Learning and Social Network Analysis.

Markus Reiter-Haas is a researcher at Moshbit GmbH and is responsible for the recommender system of the Talto career platform. He has a background in Computer Science at the Graz University of Technology with a focus on Knowledge Technologies. His master thesis tackled the evaluation of student job recommendations on the Talto predecessor Studo Jobs. His current research concentrates on creating low-dimensional embeddings for effective retrieval in the job domain.

Dominik Kowald is a post-doctoral researcher and deputy research area manager of the Social Computing team at the Know-Center. He has a Ph.D. (with hons), M.Sc. (with hons) and B.Sc. in Computer Science from Graz University of Technology. His research interests are in the fields of recommender systems, fairness and biases in algorithms, and computational social science, in which he has published more than 60 papers so far.

Manoj Reddy Dareddy is a Ph.D. Candidate in the Computer Science department at the University of California Los Angeles. His research interest is in recommender systems and applied machine learning. More specifically, Manoj works on emerging frontiers in personalization such as privacy and explainability. He received his Masters from the University of Michigan Ann Arbor and Bachelors from Carnegie Mellon University in Qatar.

Junghoo Cho is a professor in the Department of Computer Science at the University of California, Los Angeles. He received a Ph.D. degree in Computer Science from Stanford University and a B.S. degree in physics from Seoul National University. His research interest is in the theory and practice of learning, particularly in the area of language acquisition and understanding. He is a recipient of prestigious awards such as the 10-Year Best Paper Award at VLDB 2010, NSF CAREER Award or IBM Faculty Award.

Elisabeth Lex is an assistant professor and head of the Social Computing Lab at Graz University of Technology, Austria. She received a Ph.D. and an M.Sc. degree in Computer Science from Graz University of Technology. Her research interests are in the development of personalized recommender systems, in particular, algorithms based on psychological theory, as well as in computational social science, more specifically, using behavioral and network data to investigate human activity and social dynamics.

Affiliations

Emanuel Lacic[1] · Markus Reiter-Haas[2] · Dominik Kowald[1] · Manoj Reddy Dareddy[3] · Junghoo Cho[3] · Elisabeth Lex[4]

Emanuel Lacic
elacic@know-center.at

Markus Reiter-Haas
markus.reiter-haas@moshbit.com

Dominik Kowald
dkowald@know-center.at

Manoj Reddy Dareddy
mdareddy@cs.ucla.edu

Junghoo Cho
cho@cs.ucla.edu

[1] Know-Center GmbH, Graz, Austria

[2] Moshbit GmbH, Graz, Austria

[3] University of California, Los Angeles, USA

[4] Graz University of Technology, Graz, Austria

Chapter 4

Conclusions and Outlook

This thesis investigated the impact of utilizing different information sources, as such are nowadays available in many application domains. The gained insights have been further utilized to develop a scalable and customizable architecture suited for providing recommendations in a multi-domain environment. Building upon this, this thesis showed how to provide recommendations in real-time while still balancing the trade-off between the accuracy and runtime. Finally, in order to tackle the problem of improving on measures that go beyond accuracy, first an online user study was conducted to investigate the user acceptance of real-time recommendations. Following that, this thesis addressed anonymous session users and proposed a new method for providing real-time recommendations which improve on the recommendation performance with respect to beyond accuracy measures.

Thus, this chapter concludes this thesis by summarizing the achieved results and scientific contributions in Section 4.1, stating the limitations in Section 4.2 and discussing topics as well as open questions for future research in Section 4.3.

4.1 Results and Contribution

This section presents the main contributions of this thesis with respect to the research questions that were introduced in Chapter 1.

RQ1: How does combining different data sources and recommender approaches impact the robustness of recommendations?

Recommender systems often consider only user-item interactions and neglect to incorporate any additional contextual information. As such, the focus of the first research question has been on understanding the impact of utilizing different information sources, as well as how to combine them in order to provide more robust recommendations. This has resulted in the introduction of different content and network-based similarity features that can be applied to popular nearest-neighbor based methods. Such an approach further led to a better recommendation performance as, for example, the network structure that was derived from social interactions (e.g., likes or comments) was observed as a particularly beneficial feature. Within additional experimental setups it was presented how to combine the different information sources (e.g., social, location, or purchase data in case of a social e-commerce platform) and improve the quality of recommendations while being able to cover the whole user base with respect to relevant content. Furthermore, it was shown how such an approach can be especailly beneficial for users in a cold-start setting. Finally, with respect to the first research question, this thesis also contributes with SocRecM, a Java-based framework for providing real-time recommendations in the e-commerce domain that exploits heterogeneous information sources.

RQ2: How can we address customization, scalability and real-time performance across multiple recommender systems domains?

Most literature on applying recommender algorithms focuses only on scenarios within one specific application domain. As such, it has often been overlooked how to simultaneously support a diverse set of domains within the same recommender system. Thus, the second research question tackled the problem of building a recommender system for a multi-domain environment. This has resulted into a categorization of four different aspects that such a system needs to handle. First, service isolation has to be provided, as possible load spikes with respect to the number of recommendation requests of one domain should not interfere with the performance of another one. Second, the diversity of available information sources has to be taken into account, as these heavily differ between individual application settings.

Third, algorithm specific parameters should be customizable between the individual domains. This not only fosters reproducibility of the utilized approach, but also makes it possible to use the best performing approach for each domain instead of relying on a single suboptimal solution. Fourth, fault tolerance needs to be ensured via mechanisms like dynamic horizontal scaling, as domains do not necessary share the same requirements with respect to runtime, data throughput and the number of recommendation requests. To put everything together, this thesis improved on the previously mentioned SocRecM framework with respect to the second research question. As such, it also contributes with ScaR, a scalable recommendation framework that is applicable for a multi-domain setting. To show its applicability, this thesis further showed how ScaR can be adapted for various domains like Tourism, Music, Venues, E-Commerce, TEL or Jobs, just to name a few.

RQ3: How can we balance the trade-off between accuracy and runtime in real-time recommender systems?

One important aspect of recommender systems when applied in an online setting is to provide the recommendations in real-time. To achieve this, the most common approach is to calculate recommendations in a batch-wise offline manner and cache the results to be fetched at a later point in time. However, such an approach can potentially miss the current context of the user's real-time demand. Thus, the third research question has looked into how to balance the open problem of showing accurate recommendations while considering a user's real-time interest. The experimental results presented in this thesis have shown that search engine technology can be leveraged to generate recommendations under real-time constraints. Moreover, the reported scalability experiments have shown that under an exponentially growing workload (i.e., incoming recommendation requests), the average runtime performance of a utilized approach will continue to increase. But by adapting the recommendation algorithms to be suited for a scalable architecture, it is possible to counter this effect and run either the desired approaches or multiple combinations of them while achieving the expected accuracy in addition to guarantying the targeted real-time performance. In case of needing to restrict the usage of computing resources, this thesis further showed that the runtime performance can be additionally improved on an algorithmic level. For instance, this can be done via

greedy pre-filtering in nearest-neighbor methods or by utilizing neural embeddings that present a condensed form of a user's preference.

RQ4: How can we improve real-time recommendations beyond accuracy?

Most work that focus on improving recommendation algorithms follow the assumption that a higher offline accuracy performance directly translates into a better user acceptance when applied in an online setting. But the research community has recognized that other factors like the human need for variety and discovery may play a crucial role in how satisfied a user will be with the displayed recommendations. As such, the fourth research question tackled the problem of going beyond accuracy improvements in order to better understand the true utility of real-time recommendations. For this, this thesis first contributed with an online user study which showed how neural embeddings, which can improve the recommendation performance on offline measures like diversity, serendipity, and novelty, lead to a better user acceptance in an online setting. This, however, heavily depends on the location context where the recommendations are displayed and further strengthens the argument that conducting an offline experiment that focuses on improving accuracy is not enough. After that, this thesis specifically addressed the problem of providing real-time recommendations to anonymous user sessions. This resulted in a novel approach that utilizes different autoencoder architectures to extract the latent representation of a particular session. In an extensive comparison study, using the session embeddings in a nearest-neighbor manner achieved comparable accuracy results with state-of-the-art session-based recommendation approaches but managed to provide a better performance with respect to beyond accuracy measures.

4.2 Limitations

The author of this thesis recognizes several limitations of the present work and discusses them in this section.

Information sources. This thesis has considered various information sources in order to improve the robustness of recommendations. However, it did not go into

comparing a single source of information across different domains, rather it focused on the data that is available within each domain separately. This opens the path for future work to investigate the effectiveness of sharing the same source of information across domains that usually do not have anything in common, especially with respect to current literature on cross-domain recommender systems. Furthermore, the gained insights mostly resulted from simulating the user behavior within an offline experimental setup that uses only a specific snapshot of the available data. This could be alleviated with multiple data snapshots or by conducting an online user study. The later approach is usually much harder to set up, but tracking the user acceptance within a longer timeframe would result in a better understanding of the true impact of utilizing different information sources.

Comparison with other frameworks. Over the past decade, there have been several recommender frameworks reported in the literature. These include LensKit, LightFM, MyMediaLite, LibRec, RankSys or Cornac, just to name a few. To extend this list, one of the contributions of this thesis are two different recommender frameworks, namely SocRecM and ScaR. Although this was not the focus of the thesis, one limitation is that a comparison with other recommender frameworks was not done. This inherently lies in the fact that the presented research insights are more related to SaaS-based recommender systems (e.g., like the one from IBM Watson) which are close-sourced with respect to handling challenges like scalability or real-time recommendation performance. Nevertheless, this paves the way for future work to focus on adapting and extending the existing literature on framework comparisons with respect to requirements that need to considered while providing recommendations that are in an online real-time setting.

Architectural choice. While this thesis did investigate how to address the design decisions of customization, scalability and real-time performance, it did so by proposing only a single architectural solution. For instance, the Apache Solr search engine has been leveraged to ensure the real-time performance of the implemented recommendation algorithms. But one could, for instance, use other software technologies suited for handling data in real-time, like Elasticsearch or Sphinx. Moreover, this thesis proposed an architecture that is based on microservices. For future work, one could for example look into making a more in depth comparison with other

architectural design decisions, like the ones in a lambda architecture.

Algorithmic comparisons. Almost simultaneously with the evolution of the field of NLP, the research interest in recommender systems seems to have significantly increased over the past eight years. Although within this thesis, extensive algorithmic experiments were conducted, one limitation could be the choice of algorithms that were compared against in the reported research results. Even though this always depends on the specific research question and experimental scenario that is tackled, nowadays, there are a multitude of different recommendation algorithms that report state-of-the-art performance. Be it simple neighborhood based methods, computationally intensive matrix factorization algorithms or even more complex deep and wide neural network architectures, the choice of algorithms is abundant. This also reflects the current trend of revisiting the impact of recommendation algorithms. Correspondingly, the research community has identified important topics like the reproducibility of research results or algorithmic performance on measures other than accuracy.

4.3 Topics for Future Research

To conclude, this thesis states some additional potential topics for future work and open questions that could be a natural extension of the presented contributions.

Tackling biases. One important topic that future work on recommender systems in general need to focus on is how to handle biases. For example, polarization in online information [Kopeinik et al., 2019] is something that a recommender system could easily reinforce. This is in accordance to recent work [Baeza-Yates, 2020, Abdollahpouri et al., 2021] which looks into the impact of different biases like activity, algorithmic, cognitive or popularity bias. As all of them actually form a vicious cycle, further research can be directed into extending the notion of beyond accuracy evaluation with different methods on identifying and removing biases from the utilized information sources.

Incorporating privacy. Another possible extension of this thesis is to focus on topics related to recommender privacy. With the rise of different legal frameworks

like the GDPR [Krebs et al., 2019], recommender systems that are applied in various domains need to ensure that a user's personal information is not exposed. Especially in a multi-domain setting, this would result in severe privacy issues. Recent work has started to go into that direction with approaches like meta-learning [Finn et al., 2017] or privacy budgets [Muellner et al., 2021] and show that the topic of privacy will need to be investigated in much more detail.

Approximating online performance. For most researchers, the gold standard to evaluate the true utility of a recommender system is by conducting an AB test. As setting this up is hard, recent work goes into the direction of utilizing counterfactual estimators to estimate how a new recommendation policy for bandits would perform [Saito and Joachims, 2021] or model online interaction patterns by using simulators [McInerney et al., 2021]. Overall, the problems of approximating the online performance (e.g., CTR) of a recommender algorithm is still something the research community struggles with and would need to be investigated further.

Providing session-aware real-time recommendations. Session-aware recommendations, which leverage information about past user sessions, have recently gained increased interest. This opens up further research possibilities on how to efficiently adapt them for online recommendations which need to be provided in real-time. To build up on this, topics on how to handle shifts in a user's short-term interest with respect to, for example, providing reminders or considering popularity trends come into play [Jannach et al., 2017]. Actually, as stated by Latifi et al. [Latifi et al., 2021], there is a huge potential for more sophisticated approaches which combine both, the short-term and long-term preferences of a given user.

The incorporation of additional information sources, providing real-time recommendations as well as improving their prediction quality beyond just accuracy, will continue to be the focus of future research. Especially by considering the nowadays increased adoption of content personalization in a myriad of different domains and application scenarios. The author of this thesis hopes that the research community will benefit from the presented contributions and believes that the abovementioned open questions incentivize future research on real-time recommenders in multi-domain settings.

Bibliography

[Abdollahpouri et al., 2021] Abdollahpouri, H., Mansoury, M., Burke, R., Mobasher, B., and Malthouse, E. (2021). User-centered evaluation of popularity bias in recommender systems. In *Proceedings of the 29th ACM Conference on User Modeling, Adaptation and Personalization*, pages 119–129.

[Adomavicius and Tuzhilin, 2008] Adomavicius, G. and Tuzhilin, A. (2008). Context-aware recommender systems. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pages 335–336. ACM.

[Agarwal et al., 2010] Agarwal, D., Chen, B.-C., and Elango, P. (2010). Fast online learning through offline initialization for time-sensitive recommendation. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 703–712.

[Al-Ghossein et al., 2021] Al-Ghossein, M., Abdessalem, T., and Barre, A. (2021). A survey on stream-based recommender systems. *ACM Computing Surveys (CSUR)*, 54(5):1–36.

[Amatriain, 2013] Amatriain, X. (2013). Big & personal: Data and models behind netflix recommendations. In *Proc. of BigMine '13*.

[Baeza-Yates, 2020] Baeza-Yates, R. (2020). Bias in search and recommender systems. In *Fourteenth ACM Conference on Recommender Systems*, pages 2–2.

[Balabanović and Shoham, 1997] Balabanović, M. and Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Communication of ACM*, 40(3):66–72.

[Beel et al., 2016] Beel, J., Breitinger, C., Langer, S., Lommatzsch, A., and Gipp, B. (2016). Towards reproducibility in recommender-systems research. *User modeling and user-adapted interaction*, 26(1):69–101.

[Beel et al., 2013] Beel, J., Genzmehr, M., Langer, S., Nürnberger, A., and Gipp, B. (2013). A comparative analysis of offline and online evaluations and discussion of research paper recommender system evaluation. In *Proceedings of the international workshop on reproducibility and replication in recommender systems evaluation*, pages 7–14.

[Belém et al., 2013] Belém, F., Santos, R., Almeida, J., and Gonçalves, M. (2013). Topic diversity in tag recommendation. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 141–148.

[Berndsen et al., 2020] Berndsen, J., Smyth, B., and Lawlor, A. (2020). Fit to run: Personalised recommendations for marathon training. In *Fourteenth ACM Conference on Recommender Systems*, pages 480–485.

[Bischoff, 2012] Bischoff, K. (2012). We love rock 'n' roll: Analyzing and predicting friendship links in last.fm. In *Proceedings of the 4th Annual ACM Web Science Conference*, WebSci '12, pages 47–56. ACM.

[Blei et al., 2003] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022.

[Bonab et al., 2021] Bonab, H., Aliannejadi, M., Vardasbi, A., Kanoulas, E., and Allan, J. (2021). Cross-market product recommendation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 110–119.

[Cantador et al., 2015] Cantador, I., Fernández-Tobías, I., Berkovsky, S., and Cremonesi, P. (2015). Cross-domain recommender systems. In *Recommender Systems Handbook*. Springer.

[Chan et al., 2013] Chan, S., Stone, T., Szeto, K. P., and Chan, K. H. (2013). Predictionio: a distributed machine learning server for practical software development. In *Proc. of CIKM '13*.

[Chandramouli et al., 2011] Chandramouli, B., Levandoski, J. J., Eldawy, A., and Mokbel, M. F. (2011). Streamrec: A real-time recommender system. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 1243–1246.

[Chen et al., 2013] Chen, C., Yin, H., Yao, J., and Cui, B. (2013). Terec: A temporal recommender system over tweet stream. *Proceedings of the VLDB Endowment*, 6(12):1254–1257.

[Dacrema et al., 2019] Dacrema, M. F., Cremonesi, P., and Jannach, D. (2019). Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 101–109.

[Dai et al., 2014] Dai, C., Qian, F., Jiang, W., Wang, Z., and Wu, Z. (2014). A personalized recommendation system for netease dating site. *Proc. VLDB Endow.*, 7(13):1760–1765.

[Das et al., 2007] Das, A. S., Datar, M., Garg, A., and Rajaram, S. (2007). Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280.

[Davidson et al., 2010] Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., Gupta, S., He, Y., Lambert, M., Livingston, B., et al. (2010). The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296.

[de Souza Pereira Moreira et al., 2021] de Souza Pereira Moreira, G., Rabhi, S., Lee, J. M., Ak, R., and Oldridge, E. (2021). Transformers4rec: Bridging the gap between nlp and sequential/session-based recommendation. In *Fifteenth ACM Conference on Recommender Systems*, pages 143–153.

[Diaz-Aviles et al., 2012] Diaz-Aviles, E., Drumond, L., Schmidt-Thieme, L., and Nejdl, W. (2012). Real-time top-n recommendation in social streams. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 59–66. ACM.

[Duricic et al., 2018] Duricic, T., Lacic, E., Kowald, D., and Lex, E. (2018). Trust-based collaborative filtering: tackling the cold start problem using regular equiv-

alence. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 446–450. ACM.

[Eksombatchai et al., 2018] Eksombatchai, C., Jindal, P., Liu, J. Z., Liu, Y., Sharma, R., Sugnet, C., Ulrich, M., and Leskovec, J. (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*, pages 1775–1784.

[Ekstrand et al., 2011] Ekstrand, M. D., Ludwig, M., Konstan, J. A., and Riedl, J. T. (2011). Rethinking the recommender research ecosystem: reproducibility, openness, and lenskit. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 133–140. ACM.

[Elahi et al., 2013] Elahi, M., Braunhofer, M., Ricci, F., and Tkalcic, M. (2013). Personality-based active learning for collaborative filtering recommender systems. In *Congress of the Italian Association for Artificial Intelligence*, pages 360–371. Springer.

[Elkahky et al., 2015] Elkahky, A. M., Song, Y., and He, X. (2015). A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *Proceedings of the 24th International Conference on World Wide Web*, pages 278–288.

[Essinger et al., 2021] Essinger, S., Huber, D., and Tang, D. (2021). Air: Personalized product recommender system for nike's digital transformation. In *Fifteenth ACM Conference on Recommender Systems*, pages 530–532.

[Feng and Wang, 2012] Feng, W. and Wang, J. (2012). Incorporating heterogeneous information for personalized tag recommendation in social tagging systems. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 1276–1284. ACM.

[Finn et al., 2017] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR.

[Freno, 2017] Freno, A. (2017). Practical lessons from developing a large-scale recommender system at zalando. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 251–259.

[Gao et al., 2013] Gao, S., Luo, H., Chen, D., Li, S., Gallinari, P., and Guo, J. (2013). Cross-domain recommendation via cluster-level latent factor model. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 161–176. Springer.

[Garcin et al., 2014] Garcin, F., Faltings, B., Donatsch, O., Alazzawi, A., Bruttin, C., and Huber, A. (2014). Offline and online evaluation of news recommender systems at swissinfo. ch. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 169–176.

[Ge et al., 2010] Ge, M., Delgado-Battenfeld, C., and Jannach, D. (2010). Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 257–260. ACM.

[Georgiev and Nakov, 2013] Georgiev, K. and Nakov, P. (2013). A non-iid framework for collaborative filtering with restricted boltzmann machines. In *International Conference on Machine Learning*, pages 1148–1156.

[Gomez-Uribe and Hunt, 2015a] Gomez-Uribe, C. A. and Hunt, N. (2015a). The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4):13:1–13:19.

[Gomez-Uribe and Hunt, 2015b] Gomez-Uribe, C. A. and Hunt, N. (2015b). The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19.

[Gunawardana and Shani, 2009] Gunawardana, A. and Shani, G. (2009). A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research*, 10(12).

[Guo et al., 2011] Guo, S., Wang, M., and Leskovec, J. (2011). The role of social networks in online shopping: Information passing, price of trust, and consumer choice. In *Proc.*, EC '11, pages 157–166. ACM.

[Gupta et al., 2013] Gupta, P., Goel, A., Lin, J., Sharma, A., Wang, D., and Zadeh, R. (2013). Wtf: The who to follow service at twitter. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 505–514, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Gupta et al., 2014] Gupta, P., Satuluri, V., Grewal, A., Gurumurthy, S., Zhabiuk, V., Li, Q., and Lin, J. (2014). Real-time twitter recommendation: Online motif detection in large dynamic graphs. *Proc. VLDB Endow.*, 7(13):1379–1380.

[Han and Yamana, 2017] Han, J. and Yamana, H. (2017). A survey on recommendation methods beyond accuracy. *IEICE TRANSACTIONS on Information and Systems*, 100(12):2931–2944.

[He et al., 2017] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee.

[Herlocker et al., 2004] Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53.

[Hidasi and Karatzoglou, 2018] Hidasi, B. and Karatzoglou, A. (2018). Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 843–852.

[Hidasi et al., 2015] Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. (2015). Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*.

[Hu et al., 2008] Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. Ieee.

[Huang et al., 2015] Huang, Y., Cui, B., Zhang, W., Jiang, J., and Xu, Y. (2015). Tencentrec: Real-time stream recommendation in practice. In *Proceedings of the*

2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15, pages 227–238.

[Im and Hars, 2007] Im, I. and Hars, A. (2007). Does a one-size recommendation system fit all? the effectiveness of collaborative filtering based recommendation systems across different domains and search modes. *ACM Transactions on Information Systems (TOIS)*, 26(1):4–es.

[Jamali and Ester, 2010] Jamali, M. and Ester, M. (2010). A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, pages 135–142. ACM.

[Jannach et al., 2017] Jannach, D., Ludewig, M., and Lerche, L. (2017). Session-based item recommendation in e-commerce: on short-term intents, reminders, trends and discounts. *User Modeling and User-Adapted Interaction*, 27(3):351–392.

[Jannach et al., 2012] Jannach, D., Zanker, M., Ge, M., and Gröning, M. (2012). Recommender systems in computer science and information systems–a landscape of research. In *International conference on electronic commerce and web technologies*, pages 76–87. Springer.

[Jäschke et al., 2007] Jäschke, R., Marinho, L., Hotho, A., Schmidt-Thieme, L., and Stumme, G. (2007). Tag recommendations in folksonomies. In *European conference on principles of data mining and knowledge discovery*, pages 506–514. Springer.

[Kang et al., 2017] Kang, W.-C., Fang, C., Wang, Z., and McAuley, J. (2017). Visually-aware fashion recommendation and design with generative image models. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 207–216. IEEE.

[Kang and McAuley, 2018] Kang, W.-C. and McAuley, J. (2018). Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206. IEEE.

[Kersbergen and Schelter, 2021] Kersbergen, B. and Schelter, S. (2021). Learnings from a retail recommendation system on billions of interactions at bol. com. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2447–2452. IEEE.

[Kontaxis et al., 2012] Kontaxis, G., Polychronakis, M., and Markatos, E. P. (2012). Minimizing information disclosure to third parties in social login platforms. *International Journal of Information Security*, 11(5):321–332.

[Kopeinik et al., 2019] Kopeinik, S., Lex, E., Kowald, D., Albert, D., and Seitlinger, P. (2019). A real-life school study of confirmation bias and polarisation in information behaviour. In *European Conference on Technology Enhanced Learning*, pages 409–422. Springer.

[Koren et al., 2009] Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

[Kowald et al., 2018] Kowald, D., Lacic, E., Theiler, D., and Lex, E. (2018). Afelrec: A recommender system for providing learning resource recommendations in social learning environments. *Social Interaction-Based Recommender Systems (SIR'2018) Workshop co-located with Conference on Information and Knowledge Management (CIKM'2018) conference.*

[Krebs et al., 2019] Krebs, L. M., Alvarado Rodriguez, O. L., Dewitte, P., Ausloos, J., Geerts, D., Naudts, L., and Verbert, K. (2019). Tell me what you know: Gdpr implications on designing transparency and accountability for news recommender systems. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–6.

[Lacic, 2016] Lacic, E. (2016). Real-time recommendations in a multi-domain environment. In *Extended Proceedings at Doctoral Consortium of the 27th ACM Conference on Hypertext and Social Media (HT'16).*

[Lacic, 2017] Lacic, E. (2017). "real-time recommendations in a multi-domain environment" by emanuel lacic with prateek jain as coordinator. *SIGWEB Newsl.*, (Autumn):3:1–3:2.

[Lacic et al., 2015a] Lacic, E., Kowald, D., Eberhard, L., Trattner, C., Parra, D., and Marinho, L. (2015a). Utilizing online social network and location-based data to recommend products and categories in online marketplaces. In *Mining, Modeling, and Recommending 'Things' in Social Media*, pages 96–115. Springer.

[Lacic et al., 2017] Lacic, E., Kowald, D., and Lex, E. (2017). Tailoring recommendations for a multi-domain environment. *Workshop on Intelligent Recommender Systems by Knowledge Transfer & Learning (RecSysKTL'2017) co-located with the 11th ACM Conference on Recommender Systems (RecSys'2017).*

[Lacic et al., 2018a] Lacic, E., Kowald, D., and Lex, E. (2018a). Neighborhood troubles: On the value of user pre-filtering to speed up and enhance recommendations. In *International Workshop on Entity Retrieval (EYRE'18) co-located with the 27th International Conference on Information and Knowledge Management (CIKM'18).*

[Lacic et al., 2014a] Lacic, E., Kowald, D., Parra, D., Kahr, M., and Trattner, C. (2014a). Towards a scalable social recommender engine for online marketplaces: The case of apache solr. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*, WWW Companion '14, pages 817–822. International World Wide Web Conferences Steering Committee.

[Lacic et al., 2018b] Lacic, E., Kowald, D., Reiter-Haas, M., Slawicek, V., and Lex, E. (2018b). Beyond accuracy optimization: On the value of item embeddings for student job recommendations. *Proceedings of the 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).*

[Lacic et al., 2014b] Lacic, E., Kowald, D., Seitlinger, P., Trattner, C., and Parra, D. (2014b). Recommending items in social tagging systems using tag and time information. *Proceedings of the 1st International Workshop on Social Personalisation co-located with the 25th ACM Conference on Hypertext and Social Media (HT'14).*

[Lacic et al., 2019a] Lacic, E., Kowald, D., Theiler, D., Traub, M., Kuffer, L., Lindstaedt, S., and Lex, E. (2019a). Evaluating tag recommendations for e-book annotation using a semantic similarity metric. *Proceedings of the REVEAL Workshop co-located with ACM Conference on Recommender Systems (RecSys'2019).*

[Lacic et al., 2014c] Lacic, E., Kowald, D., and Trattner, C. (2014c). Socrecm: A scalable social recommender engine for online marketplaces. In *Proceedings of the 25th ACM Conference on Hypertext and Social Media*, HT '14, pages 308–310.

[Lacic et al., 2015b] Lacic, E., Kowald, D., Traub, M., Luzhnica, G., Simon, J., and Lex, E. (2015b). Tackling cold-start users in recommender systems with indoor positioning systems. *Proceedings of the 9th ACM Conference on Recommender Systems (RecSys'15).*

[Lacic et al., 2019b] Lacic, E., Reiter-Haas, M., Duricic, T., Slawicek, V., and Lex, E. (2019b). 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*, pages 496–500.

[Lacic et al., 2020] Lacic, E., Reiter-Haas, M., Kowald, D., Dareddy, M. R., Cho, J., and Lex, E. (2020). Using autoencoders for session-based job recommendations. *User Modeling and User-Adapted Interaction*, 30(4):617–658.

[Lacic et al., 2016] Lacic, E., Traub, M., Kowald, D., Kahr, M., and Lex, E. (2016). Need help? recommending social care institutions. *Workshop on Recommender Systems and Big Data Analytics (RSBDA'2016) co-location with i-KNOW'2016.*

[Lacic et al., 2015c] Lacic, E., Traub, M., Kowald, D., and Lex, E. (2015c). Scar: Towards a real-time recommender framework following the microservices architecture. *Workshop on Large Scale Recommender Systems (LSRS'2015) co-located with the 9th ACM Conference on Recommender Systems (RecSys'2015)*, 15.

[Latifi et al., 2021] Latifi, S., Mauro, N., and Jannach, D. (2021). Session-aware recommendation: A surprising quest for the state-of-the-art. *Information Sciences*, 573:291–315.

[Lewis, 2021] Lewis, L. (2021). Infographic: What happens in an internet minute 2020. *AllAccess 25/04/2022. Available at:*

https://www.allaccess.com/merge/archive/32972/infographic-what-happens-in-an-internet-minute.

[Li et al., 2017] Li, J., Ren, P., Chen, Z., Ren, Z., Lian, T., and Ma, J. (2017). Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1419–1428. ACM.

[Liang et al., 2018] Liang, D., Krishnan, R. G., Hoffman, M. D., and Jebara, T. (2018). Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 world wide web conference*, pages 689–698.

[Liu et al., 2017a] Liu, D. C., Rogers, S., Shiau, R., Kislyuk, D., Ma, K. C., Zhong, Z., Liu, J., and Jing, Y. (2017a). Related pins at pinterest: The evolution of a real-world recommender system. In *Proceedings of the 26th international conference on world wide web companion*, pages 583–592.

[Liu et al., 2017b] Liu, Q., Wu, S., and Wang, L. (2017b). Deepstyle: Learning user preferences for visual recommendation. In *Proceedings of the 40th international acm sigir conference on research and development in information retrieval*, pages 841–844.

[Loni et al., 2014] Loni, B., Shi, Y., Larson, M., and Hanjalic, A. (2014). Cross-domain collaborative filtering with factorization machines. In *ECIR'14*, pages 656–661. Springer.

[Low et al., 2012] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. (2012). Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8).

[Lu et al., 2020] Lu, F., Dumitrache, A., and Graus, D. (2020). Beyond optimizing for clicks: Incorporating editorial values in news recommendation. In *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization*, pages 145–153.

[Ludewig and Jannach, 2018] Ludewig, M. and Jannach, D. (2018). Evaluation of session-based recommendation algorithms. *User Modeling and User-Adapted Interaction*, 28(4-5):331–390.

[Ma et al., 2011] Ma, H., Zhou, D., Liu, C., Lyu, M. R., and King, I. (2011). Recommender systems with social regularization. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 287–296. ACM.

[Maksai et al., 2015] Maksai, A., Garcin, F., and Faltings, B. (2015). Predicting online performance of news recommender systems through richer evaluation metrics. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 179–186.

[Matthes et al., 2020] Matthes, J., Karsay, K., Schmuck, D., and Stevic, A. (2020). "too much to handle": Impact of mobile social networking sites on information overload, depressive symptoms, and well-being. *Computers in Human Behavior*, 105:106217.

[McInerney et al., 2021] McInerney, J., Elahi, E., Basilico, J., Raimond, Y., and Jebara, T. (2021). Accordion: A trainable simulator for long-term interactive systems. In *Fifteenth ACM Conference on Recommender Systems*, pages 102–113.

[McNee et al., 2006] McNee, S. M., Riedl, J., and Konstan, J. A. (2006). Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 1097–1101. ACM.

[Mnih and Salakhutdinov, 2008] Mnih, A. and Salakhutdinov, R. R. (2008). Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264.

[Muellner et al., 2021] Muellner, P., Kowald, D., and Lex, E. (2021). Robustness of meta matrix factorization against strict privacy constraints. In *European Conference on Information Retrieval*, pages 107–119. Springer.

[Nilashi et al., 2016] Nilashi, M., Jannach, D., bin Ibrahim, O., Esfahani, M. D., and Ahmadi, H. (2016). Recommendation quality, transparency, and website quality for trust-building in recommendation agents. *Electronic Commerce Research and Applications*, 19:70–84.

[Ning and Karypis, 2011] Ning, X. and Karypis, G. (2011). Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pages 497–506. IEEE.

[Parra and Sahebi, 2013] Parra, D. and Sahebi, S. (2013). Recommender systems : Sources of knowledge and evaluation metrics. In *Advanced Techniques in Web Intelligence-2: Web User Browsing Behaviour and Preference Analysis*, pages 149–175. Springer-Verlag.

[Patro et al., 2020] Patro, G. K., Chakraborty, A., Banerjee, A., and Ganguly, N. (2020). Towards safety and sustainability: Designing local recommendations for post-pandemic world. In *Fourteenth ACM Conference on Recommender Systems*, pages 358–367.

[Rana and Jain, 2015] Rana, C. and Jain, S. K. (2015). A study of the dynamic features of recommender systems. *Artificial Intelligence Review*, 43(1):141–153.

[Reiter-Haas et al., 2017] Reiter-Haas, M., Slawicek, V., and Lacic, E. (2017). Studo jobs: Enriching data with predicted job labels. *Workshop on Recommender Systems and Social Network Analysis (RS-SNA'2017) co-located with i-KNOW'2017*.

[Rendle et al., 2009] Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2009). Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press.

[Rendle and Schmidt-Thieme, 2008] Rendle, S. and Schmidt-Thieme, L. (2008). Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 251–258.

[Resnick et al., 1994] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM.

[Roitero et al., 2020] Roitero, K., Carterette, B., Mehrotra, R., and Lalmas, M. (2020). Leveraging behavioral heterogeneity across markets for cross-market train-

ing of recommender systems. In *Companion Proceedings of the Web Conference 2020*, pages 694–702.

[Ronen et al., 2013] Ronen, R., Koenigstein, N., Ziklik, E., Sitruk, M., Yaari, R., and Haiby-Weiss, N. (2013). Sage: Recommender engine as a cloud service. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 475–476.

[Russom et al., 2011] Russom, P. et al. (2011). Big data analytics. *TDWI best practices report, fourth quarter*, 19(4):1–34.

[Saberian and Basilico, 2021] Saberian, M. and Basilico, J. (2021). Recsysops: Best practices for operating a large-scale recommender system. In *Fifteenth ACM Conference on Recommender Systems*, pages 590–591.

[Saha et al., 2015] Saha, B., Shah, H., Seth, S., Vijayaraghavan, G., Murthy, A., and Curino, C. (2015). Apache tez: A unifying framework for modeling and building data processing applications. In *Proc. of SIGMOD '15*.

[Sahebi and Brusilovsky, 2015] Sahebi, S. and Brusilovsky, P. (2015). It takes two to tango: An exploration of domain pairs for cross-domain collaborative filtering. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 131–138.

[Sahebi and Walker, 2014] Sahebi, S. and Walker, T. (2014). Content-based cross-domain recommendations using segmented models. In *CBRecSys@ RecSys*, pages 57–64.

[Said and Bellogín, 2014] Said, A. and Bellogín, A. (2014). Rival: a toolkit to foster reproducibility in recommender system evaluation. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 371–372.

[Saito and Joachims, 2021] Saito, Y. and Joachims, T. (2021). Counterfactual learning and evaluation for recommender systems: Foundations, implementations, and recent advances. In *Fifteenth ACM Conference on Recommender Systems*, pages 828–830.

[Salakhutdinov and Mnih, 2008] Salakhutdinov, R. and Mnih, A. (2008). Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 880–887. ACM.

[Sarwar et al., 2001] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM.

[Sarwat et al., 2013] Sarwat, M., Avery, J., and Mokbel, M. F. (2013). Recdb in action: Recommendation made easy in relational databases. *Proc. VLDB Endow.*, 6(12):1242–1245.

[Schafer et al., 2007] Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. (2007). The adaptive web. chapter Collaborative Filtering Recommender Systems, pages 291–324. Springer-Verlag.

[Shani and Gunawardana, 2011] Shani, G. and Gunawardana, A. (2011). Evaluating recommendation systems. In Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, pages 257–297. Springer US.

[Shani et al., 2005] Shani, G., Heckerman, D., and Brafman, R. I. (2005). An mdp-based recommender system. *Journal of Machine Learning Research*, 6(Sep):1265–1295.

[Shi et al., 2010] Shi, Y., Larson, M., and Hanjalic, A. (2010). List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 269–272.

[Smirnova and Vasile, 2017] Smirnova, E. and Vasile, F. (2017). Contextual sequence modeling for recommendation with recurrent neural networks. In *Proceedings of the 2nd workshop on deep learning for recommender systems*.

[Smith and Linden, 2017] Smith, B. and Linden, G. (2017). Two decades of recommender systems at amazon. com. *Ieee internet computing*, 21(3):12–18.

[Steck, 2013] Steck, H. (2013). Evaluation of recommendations: rating-prediction and ranking. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 213–220.

[Sun et al., 2019] Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., and Jiang, P. (2019). Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1441–1450.

[Trattner et al., 2014] Trattner, C., Parra, D., Eberhard, L., and Wen, X. (2014). Who will trade with whom?: Predicting buyer-seller interactions in online trading platforms through social networks. In *Proc.*, WWW '14, pages 387–388. ACM.

[Traub et al., 2015] Traub, M., Kowald, D., Lacic, E., Schoen, P., Supp, G., and Lex, E. (2015). Smart booking without looking: Providing hotel recommendations in the triprebel portal. In *i-Know 2015: 15th International Conference on Knowledge Technologies and Data-driven Business*.

[Truong et al., 2021] Truong, Q.-T., Salah, A., and Lauw, H. (2021). Multi-modal recommender systems: Hands-on exploration. In *Fifteenth ACM Conference on Recommender Systems*, pages 834–837.

[Twardowski, 2016] Twardowski, B. (2016). Modelling contextual information in session-aware recommender systems with neural networks. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 273–276. ACM.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

[Walunj and Sadafale, 2013] Walunj, S. G. and Sadafale, K. (2013). An online recommendation system for e-commerce based on apache mahout framework. In *Proceedings of the 2013 annual conference on Computers and people research*, pages 153–158. ACM.

[Wang et al., 2015] Wang, H., Wang, N., and Yeung, D.-Y. (2015). Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1235–1244.

[Wu et al., 2016] Wu, Y., DuBois, C., Zheng, A. X., and Ester, M. (2016). Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of*

the Ninth ACM International Conference on Web Search and Data Mining, pages 153–162. ACM.

[Yao et al., 2015] Yao, W., He, J., Wang, H., Zhang, Y., and Cao, J. (2015). Collaborative topic ranking: Leveraging item meta-data for sparsity reduction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

[Zaharia et al., 2010] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., Stoica, I., et al. (2010). Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95.

[Zhang et al., 2019] Zhang, S., Tay, Y., Yao, L., Sun, A., and An, J. (2019). Next item recommendation with self-attentive metric learning. In *Thirty-Third AAAI Conference on Artificial Intelligence*, volume 9.

[Zhang and Pennacchiotti, 2013] Zhang, Y. and Pennacchiotti, M. (2013). Predicting purchase behaviors from social media. In *Proc.*, WWW '13, pages 1521–1532.

[Zhang et al., 2012] Zhang, Y. C., Séaghdha, D. Ó., Quercia, D., and Jambor, T. (2012). Auralist: introducing serendipity into music recommendation. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 13–22. ACM.

[Zheng et al., 2016] Zheng, Y., Tang, B., Ding, W., and Zhou, H. (2016). A neural autoregressive approach to collaborative filtering. In *International Conference on Machine Learning*, pages 764–773. PMLR.