Algorithms in E-recruitment Systems

Corné de Ruijt

Printed by: REPRO VU Print Services | `https://repro-vu.business.site`
Cover design by: Timo van Veldhoven

The research in this dissertation is supported by a Public-Private Partnership between the VU Amsterdam and Endouble.

VRIJE UNIVERSITEIT

ALGORITHMS IN E-RECRUITMENT SYSTEMS

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor of Philosophy aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. J.J.G. Geurts,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Bètawetenschappen
op vrijdag 20 mei 2022 om 9.45 uur
in een bijeenkomst van de universiteit,
De Boelelaan 1105

door

Corné Adrianus Maria de Ruijt

geboren te Delft

promotor:	prof.dr. S. Bhulai
copromotor:	prof.dr. G.M. Koole
promotiecommissie:	prof.dr. M.C.M. de Gunst
	prof.dr.ir. A.P. de Vries
	prof.dr. E. Kanoulas
	dr. C.C.S. Liem
	dr. S.T. Mol

Aan mijn vader

Acknowledgements

I would hereby like to thank the many people who have, directly or indirectly, contributed to this dissertation.

First of all, I would like to thank my promotor Sandjai Bhulai. The road to this dissertation has, like any dissertation, taken many turns. But without your patience in letting me find my own research edge, and your critical feedback on my work, this dissertation would not have been possible. I also would like to thank my co-promotor, Ger Koole, and Bram Gorissen, who was my co-promotor during the first year of my promotion.

I would also like to share my gratitude to the (former) members of the Analytics and Optimization research group at the Vrije Universiteit. A year of Corona solitude only made me realize how much I appreciate the countless conversations and the many fun moments we had. Needless to say, I hope there are many pizza calzones and Alpen trips yet to come.

I also owe my thanks to my former colleagues at Endouble. Here, I would especially like to mention Leon Willemsens and Jan Peter Tulp, the former directors of Endouble. I can only say that it is surprising what sending your kids to lessons of Krav Maga can lead to. I also owe my thanks to Robert Botman, Richard Jonkhof, Han Rusman, and Dimitrios Psarrou. You not only guided me through the difficult area between research and practice, but also your "datanthusiasm" kept me excited about this research, even on days when I was scheduled for lunch shift 2.

I owe a special thanks to Ton Sluiter. Ton has played an important role in this research, not only as an ambassador for data-driven recruitment, but also by building connections between researchers and organizations, from which this research has profited. I also would like to thank Jasper Hansman, Riecold

Veldkamp, and Sander Stuut, for the great conversations we had about recruitment analytics, and their help in gathering and interpreting datasets.

I would like to thank Vladimer Kobayashi. I greatly enjoyed our collaboration, and already miss your inexhaustible positivity. Also, I would like to thank Stefan Mol for his support and advice during the course of this work.

Of course, I would not have been able to write this dissertation without my family. You have always been able to give me the right push when I needed it, especially when my estimate of “almost done" was somewhat off. I would like to thank the members of my rock band Source. You’ve always provided the right type of distraction from my research. It is a cliché, but still, you rock! Last, I would like to thank the committee members for reading and reviewing my dissertation.

Contents

Chapter 1

Introduction

A personal anecdote

A few years back, I was invited to attend a discussion on the future role of artificial intelligence (AI) in recruitment. The other participants were mostly recruitment managers, responsible for managing the recruitment process at their firm, and two “AI experts”, myself included. Many participants already knew each other from past events, so little time was needed for everyone to introduce themselves. While waiting for the last participants to arrive, already some discussions started. One of the subjects was what e-recruitment (electronic recruitment) system each organization was using, and to what extent these fulfilled the needs of the recruitment department.

These e-recruitment systems enable job seekers to search and apply online to vacancies. They typically consist of an Applicant Tracking System (ATS), and a career website (also called recruitment website, or job portal). In its basic form, the ATS is a bookkeeping system for recruitment. It keeps track of what vacancies are available to receive applications, what applicant applied to what vacancy, and the current stage of the recruitment process each candidate is in. Examples of these stages include that the résumé and letter of motivation still need to be checked, or that the candidate has past the first interview. The ATS is what people from within the organization interact with. For example, managers of the department looking for new employees, also known as hiring managers, may use it to create new vacancies and keep track of applicants in existing vacancies. Recruitment managers may use it to track overall metrics of the recruitment process. On the contrary, the career website is what job seekers interact with. It allows them to search through vacancies and apply.

Here, there is a distinction between career websites and corporate career websites. While the former may publish vacancies from any organization, corporate career websites only publish vacancies from the organization itself. The corporate career website is also where, most often, job seekers can fill in the application form in order to apply. Given that this research was conducted in close collaboration with Endouble, a (at the time of writing) large vendor of corporate career websites in the Netherlands, I was eager to hear these recruitment managers discuss the pros and cons of e-recruitment systems.

Once all participants had arrived and everyone was introduced, the official program of the event started. The host of the event asked a number of yes-or-no questions about applied AI in recruitment, which the participants were asked to answer by raising their hands. These questions would then be followed by a more in-depth discussion. The question that struck me the most was when the recruitment managers were asked whether they were using AI in their recruitment process at that moment. To my surprise, the majority did not raise their hand, implying they were not using AI. After a brief discussion, it became clear that most recruitment managers had interpreted this question as "Are you currently using a machine learning model to automatically detect what candidates you should hire?". The recruitment managers had not noticed that, only a few moments earlier, they had been using a piece of AI to look up that new ATS vendor. That is, they had been using a search engine.

Search engines are used so often in daily life, that the impact they have is perhaps easily forgotten. However, search engines and recommender systems do have a large influence on how job seekers and recruiters search/advertise jobs. In many recruitment departments, you are likely to find a recruitment marketeer who can tell you about the importance of selecting the right keywords to make a vacancy findable via a search engine (also known as Search Engine Optimization, or SEO), or how to advertise this vacancy using social networks such as LinkedIn or Facebook. This recruiter may also tell you that you should republish a vacancy once in a while on a career website, such that the vacancy will again appear on top of the Search Engine Result Page (SERP). Also, he/she may tell you that the look and feel of the corporate career website influences who enters the applicant pool in the first place [28]. On the contrary, even though automated candidate selection is often hyped in the media (e.g., [100, 169]), using machine learning to solve the candidate selection problem remains to the adventurous few [212, p. 31].

I point out the importance of search engines here, as it often provides low-hanging fruit for optimizing the recruitment process. A similar argument can be made for job recommender systems. An example of such low-hanging fruit, is that we may change the order of a vacancy search result based on the number of applications each vacancy has received. Here, we would make use of the *position bias* in search engines and recommender systems.

Job seekers are more likely to click on vacancies on top of the SERP, as these are more likely to be evaluated by job seekers than vacancies at lower positions. Changing the order of vacancies allows us to promote vacancies with fewer applications. Such small adjustments may, under certain conditions, lead to a better spread of applicants over multiple similar vacancies, without so much reducing the probability of finding a suitable candidate [24]. Changing the SERP's order based on metrics such as on applicant counts is relatively easy to implement in, for example, current document database systems (e.g., [47]). Predicting how many applications a vacancy will receive will be the subject of Chapter 6.

What are these conditions under which we can change the order of vacancies? Obviously, vacancies should still satisfy the filters that a job seeker sets on the website, such as a location or job type filter. Though, even if vacancies satisfy these filters, the job seeker might still find one vacancy more relevant than another. Part of this dissertation will be about how to estimate this relevance. The relevance is often obscured in observed clicks. Click through rates of two identical vacancies may be different if one vacancy has been shown at higher positions in SERPs more often. Estimating relevance in the presence of the position bias is discussed in Chapter 3. Also, click probabilities may differ, depending on whether the job seeker has viewed the vacancy before. That the job seeker has viewed the vacancy before, however, may not be reflected in the data. Estimating whether different website sessions belong to the same user is one of the problems considered in Chapter 4.

That is not to say, that the candidate selection problem is not a problem worth studying. With the cost of turnover being somewhere between once or twice an employee's salary [37, pp. 88-89], avoiding a bad hire can lead to considerable cost savings. However, it is also a difficult problem. Both the features and the target variable come with considerable fuzziness, making it difficult to train models and interpret their results (e.g., see Chapter 5). And, as my anecdotal example illustrates, the novelty of the candidate selection problem may conceal other opportunities for applying AI in recruitment. Hence, apart from the scientific contributions in this dissertation, I also hope to broaden the perspective on how AI can be applied in the recruitment process.

About this dissertation

From the late 1990s, job seekers and organizations have increasingly adopted the internet as a means to find and market jobs. And, following Suvankulov [216], not without reason. Early studies on the usefulness of e-recruitment make an impressive list of benefits. These include cost reduction, increased speed of hiring, higher quality applicants, access to passive job seekers, larger geographical spread and therefore a larger audience, labor market insights, and so forth. However, Suvankulov at the same time composed an equally impressive list of the disadvantages. These include an overwhelming number of (sometimes unqualified) candidates, privacy problems, lack of personal touch, discrimination against those without access to the internet, the considerable time required sifting through applications, and recruitment websites being (too) difficult to use (the latter also follows from Maurer and Liu [170]).

Although some of these problems remain, there have been many improvements in information systems and information retrieval since then, including those in the e-recruitment domain. Also, the arrival of (professional) social networks, and the possibility to track users, has led to new insights on how job seekers and employers use the web. These can be used to remove hurdles that job seekers and recruiters may encounter when using e-recruitment systems.

In this dissertation, we will focus on recommender systems and search engines that may be encountered in e-recruitment systems. Most often, this is either a system that recommends vacancies to online job seekers, or the reverse, a system that recommends candidates to recruiters. Here, we consider the search engine as a specific case of a recommender system. That is, search engines base their search result on a search query, which is not required in recommender systems.

Recommender systems and search engines are concerned with estimating the utility that each item in the search result will have for the user. Following the terminology from click models, we will refer to this utility as the relevance of an item to a user. To estimate this relevance, the search engine/recommender system only has partial information. For example, it may have information about what items were clicked on in previous searches. We will consider several contexts in this dissertation, in which we estimate the relevance. These contexts are mostly determined by what data is available, and what definition of relevance is used.

This dissertation is structured as follows. Chapter 2 summarizes some key developments in job recommender systems from the past decade. We find that this literature could benefit from a more application-oriented view. Too often, job recommender systems are treated as just another recommender system. These contributions thereby neglect application dependent factors, such as the temporal and reciprocal nature of job recommendations, or attention to algorithm fairness. We also provide another taxonomy to classify job recommender systems, with the aim of branching the large class of hybrid job recommender systems found in the literature. Chapter 2 is based on de Ruijt and Bhulai [63].

Although the techniques discussed in Chapters 3 and 4 could be applied in e-recruitment systems, the chapters focus on a more general problem of estimating relevance from click data, given some bias in the data. These biases include the position bias and user censoring. In Chapter 3, we wish to estimate how relevant a job seeker will find the vacancy at some position t in a search result or recommendation. The difficulty of this task lies in that we do not observe this relevance, but only clicks/ skips (no click). These clicks/skips may not only reflect the relevancy, but may also depend on how job seekers search through vacancies. The literature suggests many models and estimation methods to estimate relevance in the presence of certain click biases. However, we find that many of these models can be estimated in a uniform way. To do so, we make use of the Expectation-Maximization algorithm on an Input/Output Hidden Markov Model. The method has been implemented in a `python` package called `gecasmo` [61]. Chapter 3 is based on de Ruijt and Bhulai [62].

Chapter 4 contains two contributions. First, we present a simulation model that can be used to simulate traffic on a (job) search engine or recommender system. Apart from simulating traffic, the simulation model also includes the option to censor which job seeker produced which session. Second, we apply the simulation model to the task of user clustering. That is, the task of identifying which internet session originated from which user. This question is present on corporate career websites, where there is typically no log-in. Hence, sessions from one user may only be uncovered using clustering techniques, or by using internet cookies. The result suggest that the censoring effect from cookies is relatively small. Therefore, most web statistics based on cookies are quite accurate, even if some censoring occurs. For clustering, we compared several variations of the (H)DBSCAN* algorithm to clusters users, without or by making limited use of cookie data. However, the results from using cookies to identify users vastly outperformed these clustering methods. Chapter 4 is based on de Ruijt and Bhulai [59, 58].

In Chapter 5, we consider the problem of matching job seekers and jobs, using their textual descriptions (the résumé and the vacancy). However, instead of using their semantic overlap to evaluate the quality of the match, we try to estimate how long the candidate will remain in the job described by the vacancy. To do so, we make use of the job durations that job seekers often indicate in the job history section of their résumé. We will refer to these job durations as *job tenures*. Although we do find that including time-related variables may improve matching job seekers to vacancies, the prediction problem is difficult. Job seekers indicate their previous job tenures at different levels of precision, and may use different definitions of a job. These factors may explain why more complex survival estimation methods, such as a random survival forest, performed only marginally better than simpler models such as a Kaplan-Meier estimate on this task. Chapter 5 is based on de Ruijt and Bhulai [64].

Chapter 6 compares various commonly used machine learning methods at the task of estimating the weekly number of applications per vacancy on a corporate career website. The predictions are mainly aimed at identifying vacancies that could have an applicant excess/shortage in the future. But also, they can be used to identify decision variables that may be used to influence job seeker behavior. Chapter 6 is based on de Ruijt et al. [65, 66].

Publications not contained in this dissertation
Anne Jonker, Corné de Ruijt, and Jornt de Gruijl. Bag & tag'em - A new Dutch stemmer. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 3868–3876, 2020.

Chapter 2

Patterns in Job Recommender System Literature

This chapter provides a review of the job recommender system (JRS) literature published in the past decade (2011-2021). Compared to previous literature reviews, we put more emphasis on contributions that incorporate the temporal and reciprocal nature of job recommendations. Previous studies on JRS suggest that taking such views into account in the design of the JRS can lead to improved model performance. Also, it may lead to a more uniform distribution of candidates over a set of similar jobs. We also consider the literature from the perspective of algorithm fairness. Here we find that this is rarely discussed in the literature, and if it is discussed, many authors wrongly assume that removing the discriminatory feature would be sufficient. With respect to the type of models used in JRS, authors frequently label their method as 'hybrid'. Unfortunately, they thereby obscure what these methods entail. Using existing recommender taxonomies, we split this large class of hybrids into subcategories that are easier to analyse. We further find that data availability, and in particular the availability of click data, has a large impact on the choice of method and validation. Last, although the generalizability of JRS across different datasets is infrequently considered, results suggest that error scores may vary across these datasets.

2.1 Introduction

From the start of the commercialization of the internet in the late 1980s, the question was raised of how this technology could be leveraged in employee recruitment. Even before the start of the world wide web, Vega [226] already proposed a system to match job seekers and jobs, that could *"be consulted by Minitel, using telephone number 3615 and selecting the LM/EMPLOI service"*. In other words, the service allowed job seekers to send text messages in the form of search queries or their digital résumé, over the telephone line, using a computer terminal called Minitel. The service would then compare the words in the query/résumé to a knowledge base that used a fixed job taxonomy to return a set of potentially interesting vacancies for the job seeker.

Although more than 30 years have passed since this early contribution, the usage of a fixed job taxonomy to extract information from a résumé including "branch of industry" (industry) and "qualification" (skill) using *"(a) dictionary specialized in the universe of employment"*, seems vaguely similar to LinkedIn's query processing method [146], that can be queried using your mobile phone. Of course, this is a very simplified view on reality. Vega's 200 simultaneous Minitel connections could not have served the currently close to 750 million LinkedIn users worldwide [154]. Nonetheless, the problem of recommending the right job to job seekers remains as pressing as it was more than 30 years ago.

In this chapter, we will provide an overview of the literature on job recommender systems (JRS) from the past decade (2011-2021). We will consider the different methods used in these systems, and consider these from a reciprocal, temporal, and ethical perspective. Also, we will consider the influence of data availability on the choice of method and validation. Furthermore, we put extra emphasis on branching the large class of hybrid recommender systems used in this application domain.

Our results suggest that JRS could benefit from a more application-oriented view. The reciprocal and temporal nature of JRS are infrequently discussed in the literature, while contributions that do consider these show considerable benefits. Fairness is also rarely considered in job recommender systems. If it is considered, authors too often wrongly conclude that removing discriminatory features from the data is sufficient. Recent scientific attention on fairness, including in candidate search engines, has introduced various metrics and algorithms to improve fairness. These could be used in the job recommender domain as well. In accordance with recommender system literature, deep language models have been more frequently applied in the domain of job recommender systems as well. However, what remains unknown is how well results on JRS generalize across different datasets. The only study that considers this question shows that error metrics my vary greatly over these datasets.

This chapter has the following structure. Section 2.2 discusses some earlier literature surveys. It also discusses the method used to obtain and select literature, and provides a brief discussion on some datasets and terminology we will frequently mention in this chapter. Section 2.3 discusses our findings, in which Section 2.3.1 classifies the job literature into the familiar recommender system taxonomy, while at the same time branching the large class of hybrid contributions. The remainder of Section 2.3 considers auxiliary topics such as the influence of data science competitions (Section 2.3.2), validation of JRS (Section 2.3.3), JRS taking a temporal and/or reciprocal view (Section 2.3.4), ethical considerations in JRS (Section 2.3.5), and considers contributions discussing job search and recommendation at LinkedIn (Section 2.3.6). Last, Section 2.4 draws a conclusion and discusses directions for further research.

2.2 Method and preliminaries

2.2.1 Previous surveys

In recent surveys on applications of recommender systems, job recommender systems, and recommender systems in e-recruitment, are often not included. In the well-cited review on applications of recommender systems, Lu et al. [159] do not mention the application area of e-recruitment. The same holds for the earlier review by Felfernig et al. [81]. Although most papers on neural networks in job recommender systems were published after 2018, a survey on (deep) neural networks in recommender systems, including a section on application areas, also neglects this application [15]. From the HR perspective, job search and recommendation are also not always mentioned as an application area, as opposed to candidate selection, while in the end these systems do determine who will be in the applicant pool in the first place [214].

One possible explanation could be that, from a technical perspective, the problem of job search and job recommendation is little different from a general information retrieval/recommendation task. Job seekers frequently use general-purpose search engines and online social networks to search for jobs (e.g., [114, 65, 132]). Also, many job recommender systems we will discuss in this chapter could as well be used in other application areas (and vice versa). Nonetheless, we will argue that factors such as the large amount of textual data, the reciprocal and temporal nature of vacancies, and that these systems deal with personal data, do indicate that a tailored approach is needed. The sheer volume of JRS contributions also make it clear that this application area should not be neglected.

Previous surveys on job recommender systems that consider JRS contributions before 2012 include Al-Otaibi and Ykhlef [7] and Siting et al. [210]. Especially the latter survey is very limited in scope. More recent is the survey on recommender systems in e-recruitment by Freire and de Castro [84]. Although our work has some overlap, we especially wish to address some of the limitations of the work by Freire and de Castro in this chapter.

Even though the work by Freire succeeds in collecting a substantial number of contributions in the JRS application domain, they seem to fail to classify these contributions. They thereby obscure the patterns in this literature. For example, approximately 20% of the contributions they discuss is labeled as hybrid, whereas another 33% is labeled as “other”. Although the reader would later find that the “other" category includes for 25% contributions using (deep) neural networks, this still leaves a large number of contributions with an unsatisfying label. As shown by Batmaz et al. [15], there is also a considerable development within the class of (deep) neural networks applied to recommender systems, which we also find in job recommender systems. This rich taxonomy of deep neural networks in JRS is not reflected in their paper.

The classification given by Freire and de Castro is understandable. Many contributions use mixtures of collaborative filtering and content-based techniques. These are often labeled by the contributions themselves as hybrids. However, these labels do not provide much insight into what these contributions actually entail. Furthermore, Freire and de Castro [84] focus solely on methods and validation, whereas we, among other subjects, will also take into consideration ethical concerns. We will also put emphasis on job recommender systems which, often successfully, take into account the reciprocal and temporal nature of job recommendations.

2.2.2 Preliminaries

Many contributions use one of the data sources made available through data science competitions. These datasets are often used for training and validating JRS. These datasets include the *RecSys 2016* and *RecSys 2017* competitions ([3], and [4] respectively), both using (different) datasets from the job board Xing [236]. Another dataset, the *CareerBuilder 2012* dataset, originates from the CareerBuilder Job Recommendation Challenge [122], which was hosted by CareerBuilder on Kaggle [34, 123]. All three datasets contain data with respect to candidate profiles, vacancies, and online interaction between the two. Another resource often used is the Occupation Information Network (*O*NET*) [183], an English-based job ontology that is frequently used in knowledge-based job recommender systems (see Section 2.3.1).

We will use the terms *vacancy*, *job posting*, and *job* somewhat interchangeably throughout this chapter to represent the *item* from the classic recommender system terminology. Likewise, job seekers represent the *users*. Although, as in the early paper by Vega [226], job seekers are still often described by their résumés, some e-recruitment systems consider other representations. E.g., job seekers can be represented by the social connections they have on professional social networks. When we speak of *résumés*, *CVs*, *user profiles*, or *job seeker profiles*, we assume these are synonyms and may contain additional information (such as social relations) beyond the self-description. Last, we will sometimes speak of “textbook” or “off-the-shelf”, by which we mean methods one can find in popular machine learning/pattern recognition textbooks such as Bishop [21], Flach [82], or Aggarwal [5] in the case of commonly used recommender systems.

2.2.3 Method

The following method was used to obtain the set of contributions discussed in this chapter. We used Google Scholar to search for contributions, published in a scientific journal or in conference proceedings. Here, we used the key phrases *job recommender systems*, *job recommendation*, and *job matching*. Additionally, we replaced “job” by “occupation” in these phrases. Also, a forward and backward search was applied. After reviewing, the titles and abstracts included 192 contributions.

Next, contributions on career trajectory recommendation were omitted. That is, recommender systems that only recommend a job type (such as “data scientist”), but not a specific occupation at a given organization at a specific time.

Table 2.1: Contributions per year per JRS category

		Method								
				Ensemble hybrid				*Monolithic hybrid*		
Year	Dataset	CBR	CF	Cascade	Feature aug.	Switching	Weighted	MM-SE	DNN	KB
2011	O							[185]		
2012	O	[128]								
2013	O			[160]	[70]		[26, 106]	[230]		
2014	O				[71, 69, 80, 97, 166]		[103]	[192]		
	CB12									[96]
2015	O	[45]								[130, 131]
2016	O	[73]		[93]	[141]	[205]	[153, 248]			[146]
	RS16		[6, 157]	[57, 107, 174, 184, 231, 235]		[247]	[193, 143, 254, 35]		[156]	
2017	O	[13]	[198, 142]	[24]			[222]	[43, 74, 206, 241]		[245, 246]
	CB12	[207]								
	RS17			[149, 227, 239, 204]	[94]		[144, 20]			[161]
2018	O	[115, 224]						[165, 56, 118, 237]	[196, 252]	[199]
	CB12									[8]
	RS16							[44]		
2019	O	[135]							[163, 18, 139, 180, 240]	[99, 168, 208, 201]
2020	O	[92, 48]							[19, 117, 197]	
	CB12	[177]								
	CB12, RS17, O		[136]							

Dataset abbreviations: Career Builder 2012 (CB12), RecSys 2016 (RS16), RecSys 2017 (RS17). 'O' implies another dataset was used in these papers.

Also papers on employee selection, recommender systems specifically for freelancers, and candidate recommender systems (i.e., recommending candidates for given vacancies) were filtered out. We further removed contributions without (some form of) empirical validation of their recommender system. The resulting set contained 87 contributions, which we will discuss in this chapter.

2.3 Results

2.3.1 Methods in job recommender systems

As discussed in Section 2.2.1, although the work by Freire and de Castro [84] is substantial, our critique lies in how they classify the different job recommender systems. As is common in the literature, these are split into Content-Based Recommender Systems (CBR), Collaborative Filtering (CF), Knowledge-based Recommender Systems (KB), and Hybrid Recommender Systems (HRS). However, given that so many contributions are hybrid, we will further make a distinction between *monolithic* and *ensemble* hybrid recommender systems [5, p. 200]. Also, we will split the class of ensemble hybrids, using the classification introduced by Burke [30]. Beyond monolithic and ensemble hybrids, Aggarwal [5] also mention a mixed design. However, we did not find such mixed hybrids in the JRS literature. Therefore, this design is excluded.

Content-based

Content-based recommender systems (CBRs) in the context of JRSs are models that, to construct a recommendation, only use a semantic similarity measure between the user profile and the set of available vacancies. In other words, the semantic similarity is used as a proxy for estimating the relevance of each vacancy to the job seeker. In CBRs, one creates a vector representation of the vacancy and user profile in an unsupervised way. I.e., the dimensions of these representations may not have an intuitive interpretation. Many authors use Bag of Words (BoW) with TF-IDF weighting [177, 128, 48, 73, 45], though also Latent Dirichlet Allocation is used [13], and the more recent word2vec [92, 224, 115]. Interestingly, CBR contributions have been relatively stable in the past 10 years, but also, they were not part of the top contributions during the 2016 and 2017 RecSys competitions (see Table 2.1).

Perhaps the main challenge in content-based JRS is that, in the words of Schmitt et al. [205], *"Job seekers and recruiters do not [always] speak the same language"*. I.e., job seekers and recruiters may use a different terminology to describe jobs, knowledge, or skills. As a result, two entities having the same meaning may end up with different vector representations, depending on whether they are described by the recruiter or by the job seeker. Unfortunately, this discrepancy is often neglected in content-based JRS.

Collaborative filtering

In collaborative filtering (CF), recommendations are based solely on behavioral data, typically stored in a user $\times$ items rating matrix. In the e-recruitment setting, like in the e-commerce setting, this matrix is usually filled with click behavior. For example, an element in the rating matrix may equal 1 if the job seeker (row) has clicked on a vacancy (column), and 0 otherwise. Though, in case such interaction data is missing, also the sequence of previous job occupations can be used to fill the rating matrix (e.g., [142]). The latter case does require jobs to be categorized, such that they can serve as discrete items in the rating matrix. For simplicity, we will refer to the entries in the rating matrix as ratings, irrespective of the exact type of behavioral feedback.

The literature commonly distinguishes between two types of CFs: memory-based CF and model-based CF [5]. In memory-based CF, recommendations are created using a K-nearest neighbor (KNN) type of approach. That is, for some user u, one tries to find either K users similar to u (user-based CF), or K items similar to the items u has already ranked (item-based CF). Here, the similarity is always based on the entries of the rating matrix. Contributions using textbook CF methods include Lee et al. [142], Reusens et al. [198], Ahmed et al. [6], and [156], where the latter two applied it to the RecSys 2016 dataset. Lacic et al. [136] compare several auto-encoders to encode user interactions, based on which similar users are determined.

Model-based CF tries to fill the missing values in the rating matrix using regression-based models, based solely on the rating matrix. However, likely due to the large amount of textual data, that can ideally function as features in such regression models, we are not aware of any studies using model-based CF in job recommender systems. On the other hand, although we classified regression models using textual data as hybrid, these models have a strong CF flavor. Even though textual features are used, their weights in the regression model are determined by behavioral feedback. Hence, even though these hybrids are not solely based on the rating matrix, still the resulting recommendation is largely determined by what vacancies similar job seekers have interacted with. Only the definition of 'similar' has changed, as it now partly includes features based on content.

Hybrids

As discussed earlier, hybrid recommender systems combine several models into one recommender system. Our aim here is to split this group into smaller, more similar methods. To do so, we follow the approach by Aggarwal [5, pp. 199-204], who split the hybrid recommender systems into those having a *monolithic*, or *ensemble* design. A monolithic design refers to hybrid recommender systems in which it is not possible to extract one component of the hybrid, and build recommendations based on this component independent of the other components in the hybrid, without altering the algorithm. On the contrary, ensemble methods do allow splitting the hybrid recommender system into at least two components that can (independently) be used for creating recommendations. I.e., the ensemble may consist of some models that by themselves may be monolithic.

Monolithic designs. In the class of monolithic designs, we again consider two classes. We will refer to the first class as *model-based methods on shallow embeddings* (MM-SE). These approaches commonly use off-the-shelf machine learning methods such as support vector machines [192], Naive Bayes [185], or Gradient Boosting [241, 237]. What unifies these contributions, is that the textual data is mapped to a vector space using linear or small order transformations. Although we will leave the definition of shallow or small order somewhat vague on purpose, these embeddings do have in common that this order is explicit in the method, whereas in deep embeddings this could be arbitrarily large. These shallow embeddings include TF-IDF-weighted document representations [237], representations based on a predefined job classification [241, 185], doc2vec [165], and a probabilistic stacked denoising auto-encoder [43].

Also graphical models are included in the MM-SE class of monolithic hybrids. Dave et al. [56] map jobs and skills to two matrices sharing the same latent space. Although the mapping is trained independently from O*NET, the resulting vector representations show considerable resemblance to the O*NET ontology. Jiang et al. [118], extract a large number of features from their dataset of the vacancies, job seeker profiles, and their interactions. This data is used to train a graphical model in which clicks are predicted based on estimates of whether a (job seeker, vacancy) pair is of some latent type z. They also make use of O*NET and use bin-counting [251, Ch. 5] to obtain numeric representations of some categorical features. Although the model provides an explicit click probability, as opposed to only predicting the appropriate ranking of items, the paper does mainly compare its approach with learning to rank models, such as AdaRank and RankBoost. From this comparison, the model is found to be favorable.

Three contributions model the job recommendation problem with a somewhat different objective function, though, we still label these as MM-SE. These include Dong et al. [74] and subsequent work [44]. Contrary to the approaches discussed so far, they consider the problem as a reinforcement learning problem. Given a successful or failed recommendation, a reward function is updated, taking as input the representation of a job seeker and a vacancy. To speed up computations, vacancies are represented in a binary tree. Wang et al. [230] not only consider what job should be recommended, but also when this recommendation should be made, based on estimations of when a job seeker will switch jobs.

Apart from shallow embeddings, also deep representations have become common strategies, accounting for approximately 50% of all contributions in 2019 and 2020 (see Table 2.1). Since this class consists of only deep neural networks (DNNs), we will also refer to this class by the label *DNN*. Many studies follow a similar approach as language models such as BERT [68] or ELMo [189], and extend on these embeddings by adding additional hierarchical attention networks [180, 196]. Also embeddings based on CNNs are common [139], or mixtures of the above approaches [163, 18, 19, 117, 240]. Siamese neural networks have also been used for this purpose [206].

The increasing usage of DNNs in recommender systems is not limited to recommending jobs, but holds for recommender systems in general [15]. Although it is too soon to draw firm conclusions, the absence of MM-SE contributions in the last two years is somewhat striking, giving that these were quite common in 2017 and 2018. Furthermore, some contributions do benchmark their proposed DNN with models that we may label as MM-SE. I.e., they compare their approach with similar approaches on word2vec or BoW document representations, and find the deep embeddings to be favorable [196, 180, 252]. Also, in some DNN contributions, the neural network on top of deep job/user/session embeddings is compared to other commonly used machine learning algorithms, such as gradient boosted decision trees, with the latter using the same job/user/session embedding as the proposed DNN. In such comparisons, DNNs are also found to outperform other machine learning models [163, 117].

To what extent this argument is completely satisfying remains somewhat unknown. If we view the architecture on top of initial job/user/session embeddings as automated feature engineering, then comparing these models with models without additional feature engineering is perhaps not completely fair. Moving away from the application of job recommender systems, DNNs also have not always been outperforming gradient boosting or ensemble techniques in recent data science competitions, in which participants put more effort in feature engineering [113].

Ensemble hybrids. Following Burke [30], we split ensemble hybrids into four classes. *Cascade hybrids* refine the recommendation given by another recommender system. These commonly include (gradient) boosting models [174, 149, 184, 231, 235, 227, 204], where in particular XGBoost[42] is popular. All these contributions were competitors in either the 2016 or 2017 RecSys competitions, with [235] and [227] being the winning contributions to the 2016 and 2017 competitions respectively. The problem has also been addressed from a learning to rank perspective, using LambdaMART [107]. Besides boosting, also refinements using common CBR or CF methods have been proposed [57, 239, 160]

Gui et al. [93] propose the so-called benefit-cost model, a re-ranking procedure for recommender systems that takes into account downside in recommender systems. They validate their approach on multiple recommender system applications, including that of creating job recommendations. To our knowledge, this contribution is the only contribution that raises the question of whether in some scenarios a recommendation should be given at all. Ill-fitted recommendations could hurt the trust job seekers have in the recommender system, making them decide not to use it [137].

Borisyuk et al. [24] refine an earlier job recommendation by predicting the number of applications to each vacancy, and pushing less popular vacancies to the top of the recommendation list. The idea behind this strategy is that, perhaps contrary to the e-commerce recommender setting, it can be hurtful to have too many applicants on a single vacancy. Recruiters will have to spend at least some time on each application to evaluate the fit between the applicant and the vacancy. Also, the result of this evaluation has to be communicated to the job seeker. Borisyuk et al. [24] also find that pushing less popular vacancies does increase the number of applications to those vacancies, hence causing a better spread of the applicants over the vacancy portfolio. However, this is conditioned on whether the less popular vacancies are sufficiently relevant.

Similar to cascade hybrids is the *feature augmentation* class, in which case the result of the previous recommender system in the sequence is not so much refined, but simply used as an input for the next model. The somewhat similar approaches by Diaby et al. [70, 71], Diaby and Viennet [69], and Malherbe et al. [166], Faliagka et al. [80], Gupta and Garg [97], Guo et al. [94] use such an approach. All these cases apply off-the-shelf methods on top of a CBR.

In *weighted hybrids*, the outputs of the separate models are combined using some (possibly non-linear) combination of the predicted scores. Commonly, CBR and model-based CF are combined in this way [193, 143, 144, 222, 103, 35, 20]. Others combine a large number of textual and behavioral features [254, 248, 26]. We consider these approaches ensemble hybrids, and not monolithic hybrids, as some of these features may be used directly for job recommendation by themselves. However, these methods are often comparable to methods classified as MM-SE. The output of several off-the-shelf machine learning methods has also been used for this purpose [153].

A common problem in CF is the cold-start problem. New users/items may not have given/received any behavioral feedback yet, making CF-based recommendations difficult. Although multiple hybrid approaches can be used to resolve this problem, perhaps the most direct approach is to use *switching hybrids.* In JRS, this most often implies that the recommender system uses CF by default. However, if an item or a user has insufficient data, the recommender system switches to CBR. Contributions using such approach with off-the-shelf CF and CBR methods include Zhang and Cheng [247], and Schmitt et al. [205].

Knowledge-based JRS

Although Aggarwal [5] define knowledge-based recommender systems as recommender systems having the conceptual goal to *"Give me recommendations based on my explicit specifications of the kind of content (attributes) I want"*, we will rather use the definition given by Freire and de Castro [84]. They define it as *"[Recommender systems] which rely on deep knowledge about the product domain (job) to figure out the best items (job vacancies) to recommend to a user (candidate)".* In job recommender systems, this often implies that both the job and candidate profiles are mapped to some predefined job ontology (i.e., the knowledge base), after which the two are matched.

A common strategy to generate job recommendations is then to compute the similarity between the candidate profile and vacancy in the ontology space [201, 130, 168, 131, 245, 246, 96, 208, 8, 161]. The overlap can be computed by, for example, the Jaccard index ([96]). Although the construction of such ontologies can take considerable effort, they have been used successfully in practice by for example LinkedIn [146], and Textkernel [219]. Nudging users to complete their profile by recommending skills from the ontology has also been shown to be a successful strategy to improve such recommendations [14]. Mapping jobs and candidate profiles to a shared ontology also provides a solution to the discrepancy between how job seekers and recruiters define jobs.

Another advantage of using job ontologies is that this also simplifies the implementation of keyword-based search engines and simplifies filtering. Gutiérrez et al. [99] consider such an approach, in which an interactive recommender system was built on top of the knowledge-based job recommender ELISE [77]. In this system, users are able to filter jobs based on travel distance or the type of contract. The recommender system also explains why some recommendation is (not) given, e.g., by indicating that a required skill is not mastered by the job seeker. Participants in the study indicated that the tool allowed, citing the authors, *"greater autonomy, a better understanding of needed competencies and potential location-based job mobility"*.

Reusens et al. [199] compare several CF approaches with a keyword search in terms of recall and reciprocity, both for job recommendation and candidate recommendation. With respect to job recommendations, although traditional CF approaches performed acceptably in terms of recall, they were inferior to keyword search in terms of reciprocity. The authors did find that using a 'reversed' rating matrix for job recommendation (that is, building a rating matrix on whether recruiters saved job seeker profiles for specific vacancies) improved reciprocity even beyond keyword search, though with a trade-off in terms of recall.

2.3.2 Competitions

Competitions have played an important role in the job recommendation literature. In these competitions, organizations share a dataset, an objective, and an error measure with respect to this objective. These are often shared via a competition platform, where data science teams can also enroll to the competition. In the case of job recommender competitions, the goal for each team is often to construct a job recommendation for a set of hold-out users. As already hinted in Table 2.1, to our knowledge, three competitions have had a considerable impact on the job recommender literature. 1) The 2012 Careerbuilder Kaggle competition [122], 2) the RecSys 2016 competition [3], and 3) the RecSys 2017 competition [4]. The latter two both used a dataset from the German job board Xing [236].

Besides being used for contributions to the competitions themselves, the datasets are also commonly used after completion of the competition. For example, to train and validate job recommender systems when no dataset is available. Given that approximately 32% of all job recommender system contributions use one of these datasets, they have had a considerable influence on the job recommender literature.

When considering the type of models proposed to the RecSys 2016 and 2017 competitions, we find that most were cascade or weighted hybrids. Gradient boosted trees were in particular successful [235, 227]. There is, however, no free lunch. All contributions show that a considerable time and effort was spent on constructing useful features to represent job seekers, jobs, and their interaction.

We will emphasize on two more observations with respect to competitions. First, the 2012 CareerBuilder dataset contains user interactions. Hence, it could be used to evaluate a collaborative or hybrid recommender system. However, it is frequently used to evaluate content-based recommender systems (see Table 2.1). Second, to our knowledge, only Lacic et al. [136] compare their proposed recommender system over multiple datasets: the Careerbuilder dataset, the 2017 RecSys dataset, and a private dataset originating from the student job portal Studo Jobs [215]. What is striking, is that the performance of the different models differs considerably across datasets, with no unanimous agreement across datasets and across error measures which model should be preferred. I.e., the results dependent on the dataset. However, the authors do find that from the set of models they consider, using variational auto-encoders to embed user sessions gave overall the best performance.

2.3.3 Validation

Validation when lacking job seeker - vacancy interaction data Many authors state in their introduction that there is a vast amount of online data available, commonly followed by a reference to the current LinkedIn user count [154]. Although this may be true in terms of vacancies and job seeker profiles, researchers do not always have access to interactions between the two. Naturally, this also impacts the type of methods and validation that is used.

In case interaction data is lacking, authors propose several strategies to still be able to evaluate their recommendations. One of these is to use one of the competition datasets for validation and training. As already discussed in Section 2.3.2, approximately 32% of all contributions use one of the competition datasets, but from Table 2.1, we also find that these were used after the competitions had finished.

Expert validation is also used frequently for validation. During such expert validation, the quality of recommendations is inquired by a group of 'experts'. These experts may be the researchers themselves, HR/recruitment experts, or sometimes students (e.g., [165]). Although authors rarely provide an argument for using expert validation, the choice could well be influenced by lacking interaction data. Approximately half of the contributions that use CBR or KB use expert validation ([128, 130, 45, 131, 13, 245, 246, 115, 224, 168, 208, 92]). As opposed to CF, CBR or KB do not (always) require interaction data to create recommendations.

Another way to obtain behavioral feedback when interaction data is lacking, is by using the previous N jobs in the job detail section of a résumé to predict the $N+1$-th job. This does come with the challenge of properly defining a job. E.g., some job seekers may consider all the different positions they occupied at one firm as separate jobs, whereas others may call this one job. I.e., this type of modeling requires clustering of the jobs, such that they are all approximately on the same level. Another challenge is that many details about the job may be missing [118], which complicates the job clustering.

Contributions that use the next job to train the model, but use expert validation for validation, include Diaby et al. [70], Diaby and Viennet [69], Malherbe et al. [166], Diaby et al. [71], and Lee et al. [141]. Contributions using the next job for both training and validation are Paparrizos et al. [185], Heap et al. [103], Domeniconi et al. [73], Lin et al. [153], Shalaby et al. [207], Lee et al. [142], and Dave et al. [56]. Predicting a job seekers next job is an unary classification problem. We only observe what jobs someone occupied, not those he/she rejected or was rejected for. Even though in such cases one could use different methods of negative sampling, all these contributions consider the jobs not switched to as negatives.

With respect to the best performing contributions during the RecSys 2016 and 2017 competitions, we find that none of these were CBRs. This may be because interaction data was given, hence, why not use it? However, also in other hybrid recommender systems, incorporating behavioral feedback is found to improve the recommender system, even if one has access to a well-defined and validated job ontology [146].

Validation with interactions: choices in negative sampling When interaction data is used in the form of clicks, the classification problem is also unary. I.e., if a vacancy in some position t in the job recommendation list was not clicked, we cannot be certain whether the job seeker did not find it relevant, or whether the vacancy was skipped.

In this case, common strategies for defining negatives include using shown but skipped items [107, 118, 139, 156, 174, 180, 184, 193, 197, 235, 241, 248, 254, 94], or picking negative samples at random (not per se uniform) [18, 239, 240, 143, 56]. If the recommender allows for sparse matrices, also all possible vacancy-user interactions can be used [144, 192, 141, 136, 26, 198, 6, 142, 157, 35, 20, 161, 204]. Less common strategies are replacing the job, but not the candidate or any further context, at random [252], or using vacancies of which the vacancy details were shown, but did not lead to an application [197, 196]. Others incorporate negative sampling into the estimation method itself [149, 19]. Some datasets include even more user actions besides clicks/skips, such as the deletion of stored vacancies from a profile [231, 227], or whether the candidate was hired/rejected [117]. These can also be used to define positive/negative instances.

2.3.4 On the temporal and reciprocal nature of job recommendations

In Borisyuk et al. [24], the authors note on interaction with vacancies posted at LinkedIn that *"Since users like to apply for new jobs, our [previous] job recommendation system tends to always show new jobs more often than old jobs"* This relationship is also found by [135]. Furthermore, the winning RecSys 2016 contribution explicitly takes into account the recency of items, something that contributions with similar methods (i.e., using XGBoost) did not.

Whether the claim by Borisyuk et al. (users like to apply to new jobs) holds is the question. The relationship could also be reversed: recent vacancies are shown more on top of the recommendation list, and therefore more likely to be clicked. Furthermore, given that vacancies with high demand will find a candidate faster, there might also be a survival bias. Vacancies with longer lead times are likely to be vacancies with low demand, as those with high demand were already removed from the platform (i.e., as they found a suitable candidate). Nonetheless, irrespectively of the causal chain, we would argue that the vacancy lead time should be taken more into account in job recommender systems, whereas we find that in most literature this aspect is currently neglected.

In one early contribution of job recommender systems [167], the authors represent the job recommendation problem as a reciprocal recommendation problem. They state that *"Theory shows that a good match between persons and jobs needs to consider both, the preferences of the recruiter and the preferences of the candidate"*. Although some contributions do take this reciprocal nature into account, and usually with success (e.g., [24, 199, 139]), most contributions consider the one-dimensional job seeker perspective, neglecting possible harm to the employer and/or job portal.

For example, the long tail problem in recommender systems [5, p. 33] also plays a role in job recommender systems and job search engines (e.g., [65]). There exists some extreme cases, in which this may lead to receiving over 1500 applications per vacancy, as was the case for jobs at the United Nations office in New York [25, p. 62]. However, the probability of hiring a candidate seems to be concave increasing in the size of the applicant pool [24]. I.e., growing the size of the applicant pool is likely to lead to diminishing returns. In case there is a large applicant pool for a single vacancy, this not only implies that the recruiter will have to sift through more résumés, but also has to reject more (possibly also well-suited) candidates. Rejecting many candidates may damage the employer's brand. Furthermore, as also shown by Borisyuk et al. [24], job seekers do apply to less popular, but still relevant vacancies, if these are pushed to the top of the recommendation list.

2.3.5 Ethical aspects in job recommender systems

In their literature review on applied machine learning in human resource management, Strohmeier and Piazza [214] state that *"Ethical and legal aspects are not broadly considered in current research, only equality of treatment and protection of privacy issues are discussed"*. Furthermore: *"A few of the research contributions address equality of treatment [...], mostly by excluding potential discriminatory features from mining"*. Unfortunately, in job recommender literature, equality of treatment is also rarely considered. When it is considered, also here authors come to the conclusion that simply excluding the discriminatory feature would be sufficient (e.g., [196]).

Excluding discriminatory features may seem logical from the perspective of preserving privacy, or when trying to eliminate human bias. But at the same time, this may actually hurt equality of treatment. To check whether the algorithm is discriminating against a certain group, one would require these discriminatory features. Following the argument made by Perez [188, Ch. 3-6], the problem of (gender) inequality is perhaps caused by the fact that the impact of HR/recruitment policies, and as we argue here, the choice for implementing a certain job recommender system, are not tested against specific groups. As a result, the chosen policy is often (in Perez' argument male-) biased.

One can easily find examples of how sensible features extracted from a résumé or vacancy may lead to inequality of treatment, which is also called *fairness* in the literature. Trivial examples includes work experience (age-biased), or first name (gender-biased), but many are also less trivial. These include how the type of words used in a vacancy text, or online (professional) social activity may be gender-biased ([17, Ch. 4], and [182, Ch. 6] resp.), or how commuting time may be related to welfare (especially if the office is located in, say, the city center of London) [182, Ch. 6]. Hence, these should make us aware that it is not an option to assume that if a (discriminatory) feature is not observed, it does not exist.

The recent increased scientific attention on algorithm fairness [12] will hope-fully turn this tide, as online job portals are more likely to come under scrutiny by algorithm audits. Chen et al. [41] perform such an audit on the candidate search engines of Monster, Indeed, and CareerBuilder, testing these search engines on gender bias. Although all three search engines are found to not allow for direct discrimination on inappropriate demographics (e.g., by allowing to filter on gender), the researchers do find indirect discrimination both in terms of individual and group fairness. Geyik et al. [88], compare several algorithms for ensuring group fairness in candidate recommender systems/search engines, with a focus on measures for *equal opportunity* and *demographic parity*, one of which has been implemented for LinkedIn.

Another ethical concern is the usage of so-called “People Aggregators", in which researchers or people from industry use a web crawler to obtain a substantial number of professional or personal profiles. Although this is sometimes with the user's consent (e.g., [70, 69, 80]), also in many cases this remains unclear. We did not find any work in which user profiles are explicitly anonymized before processing.

2.3.6 JRS at scale: notes from LinkedIn

Although many job recommender systems have been proposed in the literature, and the internet holds a considerable number of websites where job seekers can search for jobs, we do find it relevant to put some emphasis on contributions published by LinkedIn employees for a number of reasons. Even though many JRSs have been proposed in the literature, few of these dominate the (global) recruitment market [101]. Hence, we consider it likely that job seekers' perception will be biased towards the JRSs that these larger players are using. Furthermore, although many job seekers also use other well-known general-purpose search engines/social networks, for which the large-scale argument also holds, LinkedIn includes algorithms specifically designed for establishing professional matches. Also, LinkedIn has been quite transparent about the algorithms they use, and their respective performance, and is capable of testing these algorithms online via their A/B-testing platform XLNT [238].

LinkedIn's job recommender system is on a high level described in [127], while its search engine is described in [146]. To facilitate different information needs, LinkedIn encompasses multiple search indices. These are combined into one to facilitate federated search [9]. The job search engine itself is composed of a linear combination of several features, which can be grouped into four classes. First, a query-document semantic matcher. Second, an estimator for the searcher's skill with respect to the set of potential jobs. Third, an estimator for the quality of the document, independent of the query and/or searcher. And fourth, a collection of miscellaneous features such as textual, geographical and social features. LinkedIn's job recommender system consists of a GLMix model, which includes a large number of profile features from both the candidate, the job, their interactions, and the cosine similarity between user and job features in the same job domain [248].

LinkedIn's recommender system and search engine are, however, under continuous scrutiny. LinkedIn can change certain parts of its recommender system/search engine, and test these adjustments using XLNT. Many contributions published at LinkedIn, suggesting adjustment to the recommender system/search engine, therefore have an online evaluation (e.g., [24, 230, 88]).

From considering the different algorithms and their validation, a few observations can be made. Both the recommender system and search engine rely on the set of skills that users indicate on their profile. The algorithms thereby benefit from LinkedIn's policy to nudge users to complete their list of skills [14], or to provide endorsements to other users. Also, of all individual classes of features, the estimate of one's expertise seems to lead to the largest improvement in model performance. Given that endorsements from other users influence this estimate, the recommender system and search engine benefit from LinkedIn being a social network. Although LinkedIn strives towards personalized results, the algorithms are required to be scalable to meet latency requirements [127], which leads to some algorithms being competitive to, but not outperform, other models [248].

2.4 Conclusion

In this chapter, we have discussed the job recommender system (JRS) literature from several perspectives. These include the influence of data science competitions, the effect of data availability on the choice of method and validation, and ethical issues in job recommender systems. Furthermore, we branched the large class of hybrid recommender systems to obtain a better view on how these hybrid recommender systems differ. Both this multi-perspective view, and the new taxonomy of hybrid job recommender systems, have not been discussed by previous reviews on job recommender systems.

Application-oriented challenges in JRSs were already highlighted in early JRS contributions. Still, most literature does not take these into account. Contributions that do take different views on the JRS problem, however, do show that such views can have several benefits. These benefits may include improved model performance (temporal perspective), improved distribution of candidates over a set of homogeneous vacancies (reciprocal perspective), or ensuring algorithm fairness (ethical perspective). Currently, most attention goes out to how to represent the substantial amount of textual data from both candidate profiles and vacancies to create job recommendations. For this problem, especially deep representations have shown promising results. However, this focus may also create the illusion that this is the only perspective that is relevant.

Especially in terms of fairness, such a single perspective can be harmful. Although we are not aware of algorithm audits on job recommender systems, an audit on the candidate search engines of Indeed, Careerbuilder, and Monster did show significant results for both individual and group unfairness in terms of gender inequality. The increased scientific attention towards algorithm fairness, however, does provide algorithms and metrics that can be applied to measure and ensure algorithm fairness. Hence, there is a research opportunity to study how these can be applied to JRSs.

Many authors state in their introduction that there is a vast amount of data available in the form of vacancies and job seeker profiles. However, there is a clear split in the literature with regards to contributions having also access to online interaction data between these two. By online interaction data, we in particular mean clicks/skips on vacancies in the recommendation list. Interaction data can resolve the language inconsistency between job seekers and recruiters, which is especially troublesome in content-based and some knowledge-based JRSs. In case interaction data is missing, one common resort is to use one of the available datasets from JRS competitions. These are especially the CareerBuilder 2012, RecSys 2016, and RecSys 2017 competitions, which therefore have had a considerable influence on the JRS literature.

Comparing methods on different datasets is rarely done. This is unfortunate, as the (to our knowledge) only contribution that compares JRSs on different competition datasets shows that error metrics may differ substantially across these different datasets. This raises questions with respect to the generalizability of JRSs, when trained on one dataset.

Two additional questions with respect to competition datasets are why these datasets are so often used, even outside of the competitions, and why their (online) interaction data is sometimes not taken into account. Although there may be many valid reasons, we would hypothesize that it can be difficult to obtain interaction datasets. Organizations collecting such data may not always be part of research communities. Also, sharing such datasets may involve complications, both technical and legal, creating an extra hurdle for these organization to collaborate. Hence, while lacking (interaction) data, researchers resolve to using the competition datasets. Here, they prefer not to use the interaction data to create recommendations, to ensure their model can be generalized to datasets where no interaction data is available.

Chapter 3

A Flexible EM-approach to Estimate Clicks

Given the importance of search engines to find digital information, there has been much scientific attention on how users interact with search engines, and how such behavior can be modeled. Many models on user - search engine interaction, which in the literature are known as click models, come in the form of Dynamic Bayesian Networks. Although many authors have used the resemblance between the different click models to derive estimation procedures for these models, in particular in the form of Expectation-Maximization (EM), still deriving the expressions for EM may require a substantial amount of work, especially for the E-step. What we propose in this chapter, is that this derivation is often unnecessary. Many existing click models can, under certain assumptions, be optimized as they were Input-Output Hidden Markov Models (IO-HMMs). For these models, the forward-backward equations immediately provide this E-step. Although this approach does not improve algorithmic performance, and may even be more memory-intensive, it does simplify the implementation in, for example, Python. To arrive at this conclusion, we will present the Generalized Cascade Model (GCM), and show how this model can be estimated using the IO-HMM's EM framework. Also, we will provide two examples of how existing click models can be mapped to GCM. Our GCM approach to estimate click models has been implemented in the `gecasmo` Python package.

3.1 Introduction

In the last decade, many models have been proposed to explain and/or predict web user interaction with search engines [49]. Crucial to many of these so-called click models, is to explain the *position bias*. That is, that web users tend to be more likely to click on items at the top of a Search Engine Result Page (SERP), irrespectively of whether these items are the most relevant ones in the SERP. Many of these click models are Probabilistic Graphical Models (PGMs), in particular in the form of Dynamic Bayesian Networks (DBNs) [178, 49]. The choice of using DBNs to model user interaction follows from observations in early eye-tracking studies, which suggested that web users on average evaluate the items in the (SERP) sequentially in a top-down fashion [120]. But also the work of Craswell et al. [53] has been influential. This contribution showed that a sequential click model, the so-called *cascade model*, outperformed other click models that tried to explain the position bias. Although many improvements have been made to the cascade model, these improvements have kept this sequential view.

As the parameters of more complex click models than the cascade model do not have a closed-form maximum likelihood estimator per se, estimation is commonly done via expectation-maximization (EM). To our knowledge, most authors derived the E-step manually for their proposed click model in a similar fashion as suggested by [49, Ch. 4]. I.e., the expectation has to be derived explicitly, which may be different for different models. However, as we will argue in this chapter, such a derivation is often unnecessary. Under the umbrella of EM, many click models can be rewritten as if they were Input-Output Hidden Markov Models (IO-HMMs) [16]. For IO-HMMs, the solution to the EM algorithm is equivalent to the one obtained from deriving the EM algorithm explicitly for each of these click models. I.e., if we can use an IO-HMM for a click model, the IO-HMM directly provides an explicit expression for the E-step. It thereby makes a separate derivation of the E-step for each click model unnecessary.

This chapter contributes to the current click model literature in two ways. First, we introduce a new click model, which we name the Generalized Cascade Model (GCM). We show that the EM algorithm of GCM and its corresponding IO-HMM have the same solution. Furthermore, we show that GCM is a generalization of many commonly used click models, including the User Browser Model (UBM) and the Chapelle-Zang Model (CZM). Second, as an example, we provide explicit mappings from UBM and CZM to GCM, mappings that can be used in the gecasmo package [61] to estimate the parameters of these models. The package fully utilizes the solution equivalence between GCM and IO-HMM, and therefore only requires these mappings. I.e., contrary to existing click model estimation software using EM, gecasmo does not require programming the E-step or M-step explicitly for each click model.

This chapter is structured as follows. Section 3.2 gives an short overview of the click model literature and the methods used to estimate click models. Section 3.3 presents the GCM and shows how the EM algorithm of IO-HMMs can be used to estimate GCMs, and uses some properties of GCMs to simplify this estimation. Section 3.5 shows that UBM and CZM can be mapped to GCM, and provides an example of such a mapping for both click models, which can be used in the gecasmo package. Lastly, Section 3.6 gives a conclusion and discusses ideas for further research.

3.2 Related work

Probabilistic graphical click models As already discussed, a considerable number of click models have been proposed in the literature in the form of Dynamic Bayesian Networks (DBNs) [49], many of which were inspired by the early work by Craswell et al. [53]. Since the work of Crasswell et al., many alternative DBNs have been proposed, in particular to cope with the two main limitations of the cascade model. First, it cannot model more than one click on a SERP, and second, if no item is clicked, it assumes the web user evaluated all items.

The two perhaps most well-known alternatives to the cascade model are the *user browser model* (UBM) [76], and the *Chapelle-Zhang model* (CZM) [40], which we will discuss in more depth in Section 3.5.1. From UBM and CZM, many alternative click models have been derived [49, Ch. 8]. These include the partially sequential click model (PSCM), and its generalization, the time-aware click model (TACM) [228, 158]. Both models can be viewed as generalizations of UBM, in which the depth-first search assumption is replaced by a locally unidirectional examination assumption.

Unfortunately, CZM is in the literature more commonly known as 'the Dynamic Bayesian Network' model, hence, is also abbreviated as DBN. However, since the model is a strict subset of all DBNs, and to avoid confusion in our terminology, we deliberately renamed the model to CZM in this chapter. We do like to stress that the authors of the paper speak correctly of 'a dynamic

Bayesian network'; the name 'the dynamic Bayesian network' seems to have followed from other authors referring to the paper.

Distributed approach to click models More recently, recurrent neural networks (RNNs) have grown in popularity as alternatives to DBNs [22, 23, 67]. This approach is named by Borisov et al. [22] as the *distributed approach* to click modeling. The work of Borisov et al. [22] is in particular interesting, as it shows that these RNNs are able to outperform both UBM and CZM in terms of click perplexity, though they come at the cost of increased model complexity.

The latter may be problematic if one is interested in studying the optimal order of items in a SERP under a certain objective function, as was done by Balakrishnan and Kambhampati [11]. Given the large number of parameters in these RNNs, such analysis would quickly become intractable. Furthermore, although Borisov et al. [22] attempt to explain the features learned by the RNN using t-SNE [164], this analysis is post hoc. Hence, when drawing conclusions based on such analysis, one would be at risk of a narrative fallacy. We therefore argue that, even though a distributed approach may lead to better predictions, DBNs are still valuable in explaining web user behavior, and optimizing search engines accordingly.

Estimation of PGMs Click models that have no closed form maximum likelihood estimate are commonly estimated using Expectation-Maximization (EM) [49, Ch. 4]. EM has been implemented for some so-called 'basic click models'[49, Ch. 3], which include the earlier mentioned UBM and CZM, in a number of open source software packages. These include the `ClickModels` [51], and `PyClick`[194] projects.

Even though we focus on a typical textbook implementation of EM (e.g., [21, pp. 438-439]), other authors also consider alterations to EM, or alternatives to EM, in the context of click models. Especially when the number of parameters in the model grows large, alternatives to the classic EM implementation become more interesting to consider.

In one of the more complex click models, the *Task-Centric Model*, Zhang et al. [250] make some additional independence assumptions between latent variables in order to be able to use EM. However, the authors claim that these assumptions have little impact on the likelihood distribution. Subsequently, they propose alternative updates to speed up convergence. Wang et al. [229] allow for covariates in the model, while still relying on EM for estimation.

They propose a posterior regularized EM algorithm for click models to cope with noisy clicks and mis-ordered item pairs. Apart from EM, Zhu et al. [253] propose a click model named the *general click model*, which is solved using expectation-propagation. Zhang et al. [249] use a probit Bayesian inference approach, which can be applied to estimate various click models. Both [253] and [249] allow for covariates in the model.

3.3 On the relationship between GCM and IO-HMM

3.3.1 A brief recap of IO-HMM

We start by a brief recap of the Input-Output Hidden Markov Model (IO-HMM), as introduced by Bengio and Frasconi [16]. We will use $\mathcal{P}(X)$ to denote the set of all parents of some node X in a Dynamic Bayesian Network.

Definition 3.3.1. An Input-Output Hidden Markov Model (IO-HMM) is a Dynamic Bayesian Network consisting of observed states $\{\mathbf{x}_t, \mathbf{y}_t\}_{t=1,\ldots,T}$, and latent states $\{z_t\}_{t=1,\ldots,T}$. For $t > 1$ we have: $\mathcal{P}(z_t) = \{z_{t-1}, \mathbf{x}_t\}$, whereas $\mathcal{P}(z_1) = \{\mathbf{x}_t\}$. Furthermore, for all t we have: $\mathcal{P}(\mathbf{y}_t) = \{\mathbf{x}_t, z_t\}$, and $\mathcal{P}(\mathbf{x}_t) = \emptyset$. Here $\mathbf{x}_t \in \mathbb{R}^{R_1}$, $z_t \in \{1, \ldots, K\}$, and $\mathbf{y}_t \in \mathbb{R}^{R_2}$; $R_1, R_2 \in \mathbb{N}$. The transition probabilities are given by

$$\varphi_{k'k,t}(\Omega) := \mathbb{P}(z_t = k | z_{t-1} = k^{'}; \mathbf{x}_t, \Omega), \tag{3.1}$$

whereas the probability of being in state k at time t given $\mathbf{x}_{1:t}$ is given by

$$\zeta_{k,t}(\Omega) = \mathbb{P}(z_t = k | \mathbf{x}_{1:t}, \Omega), \tag{3.2}$$

with $\mathbf{x}_{t':t} = (\mathbf{x}_{t'}, \ldots, \mathbf{x}_t)$, $t^{'} < t$, and $\Omega = \{\vartheta_1, \ldots, \vartheta_P\}$, $\vartheta_p \in (0,1)$, being a set of parameters. The emission probability for $\mathbf{y}_t$, given current state z_t and covariate vector $\mathbf{x}_t$, is given by some density function $f_y(\mathbf{y}_t | z_t; \mathbf{x}_t, \Omega)$. A graphic representation of the IO-HMM is given by Figure 3.1.

Now, let us consider IO-HMM under the lens of expectation-maximization (EM). Let $i \in \{1, \ldots, n\}$ be the index of some realization of $\{\mathbf{x}_t, \mathbf{y}_t, z_t\}_{t=1,\ldots,T}$. For brevity, we write $\mathbf{z}^{(i)}_{t':t} = (z^{(i)}_{t'}, \ldots, z^{(i)}_t)$, $t^{'} < t$ (idem for $\mathbf{y}^{(i)}_{t':t}$ and $\mathbf{x}^{(i)}_{t':t}$), and $\mathbf{Z} = (\mathbf{z}^{(1)}_{1:T}, \ldots, \mathbf{z}^{(n)}_{1:T})$ (idem for $\mathbf{Y}$ and $\mathbf{X}$). We are interested in maximizing

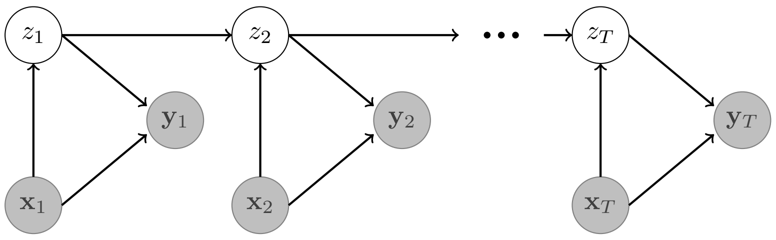


Figure 3.1: IO-HMM

the likelihood function

$$\begin{aligned}\mathcal{L}(\Omega; \mathbf{X}, \mathbf{Y}) &= \mathbb{P}(\mathbf{Y}|\mathbf{X}, \Omega) \\ &= \sum_{\mathbf{Z}} \mathbb{P}(\mathbf{Y}, \mathbf{Z}|\mathbf{X}, \Omega),\end{aligned} \tag{3.3}$$

with respect to Ω. Since it is usually not possible to maximize this function directly, EM rather tries to maximize

$$Q(\Omega; \hat{\Omega}) = \mathbb{E}_{\mathbf{Z}}\left[l_c(\Omega; \mathbf{Z}, \mathbf{Y}, \mathbf{X})|\mathbf{Y}, \mathbf{X}, \hat{\Omega}\right], \tag{3.4}$$

with respect to Ω, keeping some current estimates $\hat{\Omega}$ constant. Here $l_c(\Omega; \mathbf{Z}, \mathbf{Y}, \mathbf{X})$ is the complete data log-likelihood

$$l_c(\Omega; \mathbf{Z}, \mathbf{Y}, \mathbf{X}) = \sum_{i=1}^{n} \log \mathbb{P}(\mathbf{y}_{1:T}^{(i)}, \mathbf{z}_{1:T}^{(i)}|\mathbf{x}_{1:T}^{(i)}; \Omega). \tag{3.5}$$

Once some (possibly local) optimum Ω^* has been found, the current estimates are updated: $\hat{\Omega} \leftarrow \Omega^*$. This process continues until convergence.

In the case of IO-HMM, $Q(\Omega; \hat{\Omega})$ can be written as

$$\begin{aligned}Q(\Omega; \hat{\Omega}) = &\sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{k=1}^{K}\Bigg(\hat{\zeta}_{k,t}^{(i)} \log f_y(\mathbf{y}_t^{(i)}|z_t^{(i)}, \mathbf{x}_t^{(i)}; \Omega) + \\ &\sum_{k'=1}^{K} \hat{h}_{k'k,t}^{(i)} \log \varphi_{k'k,t}^{(i)}(\Omega)\Bigg),\end{aligned} \tag{3.6}$$

with

$$\hat{h}_{k'k,t} = \mathbb{E}\left[a_{k,t}^{(i)} a_{k',t-1}^{(i)} | \mathbf{x}_{1:T}^{(i)}, \mathbf{y}_{1:T}^{(i)}, \hat{\Omega}\right], \tag{3.7}$$

$$a_{k,t}^{(i)} = \begin{cases} 1 & \text{if } z_t^{(i)} = k \\ 0 & \text{otherwise} \end{cases}, \tag{3.8}$$

and $\hat{\zeta}_{k,t}$, being defined as in (3.2), but conditional on current estimates $\hat{\Omega}$.

3.3.2 The generalized cascade model (GCM) and its resemblance to IO-HMM

As mentioned by Chapelle and Zhang [40], many click models show resemblance with hidden Markov models. However, to our knowledge, this resemblance has so far not been made explicit. Neither has previous literature taken advantage of this resemblance. That is unfortunate, as we do believe there is an advantage to such an approach. Writing out the updates for the parameters Ω in a specific click model can be a tedious and error prone job, as is somewhat illustrated in [49, Ch. 4]. Furthermore, this approach only makes partial use of the resemblance between the different click models to quickly find an estimation procedure for Ω. Using the IO-HMM, we are (only) required to define the latent state space, and transition probabilities $\varphi_{k'k,t}^{(i)}(\Omega)$ corresponding to a particular click model, after which we can use the IO-HMM machinery to evaluate $Q(\Omega, \hat{\Omega})$.

To show that we can indeed use IO-HMM to estimate click models, we will first provide a somewhat general definition of a click models having a cascade effect. That is, where the probability of a user clicking an item at position t in a search engine result page (SERP) depends on clicks/skips on items before t, but not after t. First, we shall introduce some extra notation.

We now let i represent a single query session on a search engine. We define a query session as a single realization of the variables $\{\mathbf{x}_t, \boldsymbol{\psi}_t, y_t\}_{t=1,\dots,T}$, with latent state $\boldsymbol{\psi}_t = (\psi_{t,1}, \dots, \psi_{t,P})$, $\psi_{t,p} \in \{0, 1\}$, and $\mathbb{P}(\psi_{t,p} = 1 | \mathcal{P}(\psi_{t,p}); \mathbf{x}_t, \Omega) = \vartheta_{t,p}$. $\mathbf{x}_t$ has the same interpretation as before, while $y_t \in \{0, 1\}$, with $y_t = 1$ if item at position t is clicked, and $y_t = 0$ if the t-th item is not clicked (i.e., skipped). We assume there is exactly one configuration of $\boldsymbol{\psi_t}$ that leads to a positive probability of $y_t = 1$. We call this state the 'click state', and denote it by $\mathcal{C}_t$. I.e., we allow which state is the click state to depend on t.

Each query session results in a SERP consisting of items $\mathcal{S}_i \subset \mathcal{V}$, with $\mathcal{V} = \{1, \dots, V\}$ being the set of possible items the search engine may return. For

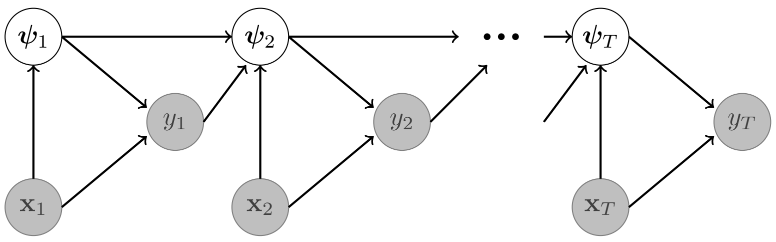


Figure 3.2: GCM

simplicity, we will assume each list $\mathcal{S}_i$ has the same number of items T, and each item in $\mathcal{S}_i$ occupies an unique position $t \in \{1, \ldots, T\}$. The item in position t in list $\mathcal{S}_i$ is given by the bijective function $r_i(t)$. Since, like in IO-HMM, we assume realizations $i = \{1, \ldots, n\}$ to be conditionally independent given $\mathbf{x}_{1:T}^{(i)}$, we will briefly omit index i in Definition 3.3.2.

Definition 3.3.2. A Generalized Cascade Model (GCM) is a Dynamic Bayesian Network consisting of observed states $\{\mathbf{x}_t, y_t\}_{t=1,\ldots,T}$, and latent states $\{\boldsymbol{\psi}_t\}_{t=1,\ldots,T}$ for which, for $t > 1$: $\mathcal{P}(\boldsymbol{\psi}_t) = \{\boldsymbol{\psi}_{t-1}, y_{t-1}, \mathbf{x}_t\}$; $\mathcal{P}(\boldsymbol{\psi}_1) = \{\mathbf{x}_t\}$, and for all t: $\mathcal{P}(y_t) = \{\mathbf{x}_t, \boldsymbol{\psi}_t\}$, and $\mathcal{P}(\mathbf{x}_t) = \emptyset$.

A graphical representation of GCM is given in Figure 3.2. The similarity between GCM and IO-HMM should become quickly apparent. In GCM, transition probabilities also depend on y_{t-1}, which is not the case for IO-HMM. On the other hand, GCM assumes y_t to be binary, instead of in $\mathbb{R}^{R_2}$, and $\boldsymbol{\psi}_t$ is possibly multi-dimensional. However, as Proposition 1 shows, when optimizing using EM, these distinctions do not matter.

Proposition 1. *$Q_{GCM}(\Omega, \hat{\Omega})$ can be written in the form of Eq.* (3.6).

Proof. We will first convert the multi-dimensional state space into a single-dimensional one, which can easily be done as all we assumed all latent state variables are binary. E.g., we may use the transformation

$$z_t^{(i)} = \text{bin}(\boldsymbol{\psi}_t^{(i)}) = \sum_{p=1}^{P} 2^{p-1} \psi_{t,p}^{(i)} + 1, \qquad (3.9)$$

to obtain the single-dimensional discrete state space. Note that the $+1$ is strictly not necessary, but simply added to map to some discrete state space counting from 1.

Using Eq. (3.4), we now have

$$\begin{aligned}
Q_{\text{GCM}}(\Omega, \hat{\Omega}) &= \mathbb{E}_{\mathbf{Z}|\mathbf{Y},\mathbf{X},\hat{\Omega}}\left[l_c^{\text{GCM}}(\Omega; \mathbf{Z}, \mathbf{Y}, \mathbf{X})\right] \\
&= \mathbb{E}_{\mathbf{Z}|\mathbf{Y},\mathbf{X},\hat{\Omega}}\left[\sum_{i=1}^{n} \log \mathbb{P}\left(\mathbf{y}_{1:T}^{(i)}, \mathbf{z}_{1:T}^{(i)} | \mathbf{x}_{1:T}^{(i)}; \Omega\right)\right] \\
&= \mathbb{E}_{\mathbf{Z}|\mathbf{Y},\mathbf{X},\hat{\Omega}}\left[\sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{k=1}^{K} \mathbb{1}_{\{z_t^{(i)}=k\}}\right. \\
&\quad \times \log \mathbb{P}\left(y_t^{(i)} | z_t^{(i)} = k, \mathbf{x}_t^{(i)}, y_{t-1}^{(i)}; \Omega\right) \\
&\quad + \sum_{k'=1}^{K} \mathbb{1}_{\{z_t^{(i)}=k, z_{t-1}^{(i)}=k'\}} \\
&\quad \left. \times \log \mathbb{P}\left(z_t^{(i)} = k | z_{t-1}^{(i)} = k', \mathbf{x}_t^{(i)}, y_{t-1}^{(i)}; \Omega\right)\right] \\
&= \sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{k=1}^{K} \hat{\zeta}_{k,t}^{(i)} \log \mathbb{P}\left(y_t^{(i)} | z_t^{(i)} = k, \mathbf{x}_t^{(i)}, y_{t-1}^{(i)}; \Omega\right) \\
&\quad + \sum_{k'}^{K} \hat{h}_{k'k,t}^{(i)} \log \mathbb{P}\left(z_t^{(i)} = k | z_{t-1}^{(i)} = k', \mathbf{x}_t^{(i)}, y_{t-1}^{(i)}; \Omega\right) \\
&= \sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{k=1}^{K} \hat{\zeta}_{k,t}^{(i)} \log \mathbb{P}\left(y_t^{(i)} | z_t^{(i)} = k, \mathbf{x}_t^{(i)'}; \Omega\right) \\
&\quad + \sum_{k'}^{K} \hat{h}_{k'k,t}^{(i)} \log \mathbb{P}\left(z_t^{(i)} = k | z_{t-1}^{(i)} = k', \mathbf{x}_t^{(i)'}; \Omega\right) \\
&= \sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{k=1}^{K} \left(\hat{\zeta}_{k,t}^{(i)} \log f_y\left(y_t^{(i)} | z_t^{(i)} = k, \mathbf{x}_t^{(i)'}; \Omega\right)\right. \\
&\quad \left. + \sum_{k'=1}^{K} \hat{h}_{k'k,t}^{(i)} \log \tilde{\varphi}_{k'k,t}^{(i)}(\Omega)\right),
\end{aligned} \tag{3.10}$$

with $\mathbf{x}_t^{(i)'} = \left(\mathbf{x}_t^{(i)}, y_{t-1}^{(i)}\right)$. $\square$

Hence, as a result, when our objective is to estimate a GCM using EM, we can simply model it as it were an IO-HMM by adding the previous click/skip to the input vector. For notational simplicity, we will in the remainder of this chapter write $\mathbf{x}_t^{(i)}$ instead of $\mathbf{x}_t^{(i)'}$. We deliberately write $\tilde{\varphi}^{(i)}_{k'k,t}(\Omega)$, and not $\varphi^{(i)}_{k'k,t}(\Omega)$, for reasons that will become apparent in Proposition 2.

3.4 On the estimation of GCMs using EM for the IO-HMM

3.4.1 Notes on the E-step

As briefly discussed, we believe that modeling GCMs as IO-HMMs has as main advantage that, given transition probabilities $\tilde{\varphi}^{(i)}_{k'k,t}(\Omega)$, the IO-HMM framework directly provides an EM procedure. Although this approach will not reduce time complexity, we believe it does reduce the effort required of finding expressions for the EM-updates. In particular, given Proposition 1, Bengio and Frasconi [16] immediately provide the expression required during the E-step. Since GCM is a simplified case of IO-HMM, we can make use of Proposition 2 to somewhat simplify Expression (3.10).

Proposition 2. *There exists transition probabilities* $\{\varphi^{(i)}_{k'k,t}(\Omega)\}^{t=1,\dots,T}_{i=1,\dots,n}$, *with* $k', k \in \{1, \dots, K+1\}$, *such that maximizing* $Q_{GCM}(\Omega, \hat{\Omega})$ *is equivalent to maximizing*

$$Q'_{GCM}(\Omega, \hat{\Omega}) = \sum_{i=1}^{n} \sum_{t=1}^{T} \sum_{k=1}^{K+1} \sum_{k'=1}^{K+1} \hat{h}^{(i)}_{k'k,t} \log \varphi^{(i)}_{k'k,t}(\Omega). \tag{3.11}$$

Proof. As one might expect, our objective here is to absorb the emission probability $f_y(\cdot)$ into the new transition probabilities $\varphi^{(i)}_{k'k,t}(\Omega)$. If we define

$$\vartheta^{(i)}_{t,y} = f_y\left(y_t^{(i)} | z_t^{(i)} = k, \mathbf{x}_t^{(i)'}; \Omega\right), \tag{3.12}$$

then a natural thing to do would be to introduce a new latent binary variable $\psi^{(i)}_{t,y}$, with $\mathbb{P}(\psi^{(i)}_{t,y} = 1 | \boldsymbol{\psi}_t^{(i)}, \mathbf{x}_t^{(i)'}; \Omega) = \vartheta^{(i)}_{t,y}$. However, this would imply $\mathcal{P}(\psi^{(i)}_{t,y}) = \{\boldsymbol{\psi}_t^{(i)}, \mathbf{x}_t^{(i)}\}$. Hence, the resulting model would not be a GCM, as $\psi^{(i)}_{t,y}$ depends on the current latent state, not the previous latent state. To circumvent this problem, we can include the dependency of the emission on the current state into the definition of the transition probability. Here we use that there exists only one click state. I.e., only when $k = \mathcal{C}_t$ does $\vartheta_{t,y}^{(i)}$ affect the transition probability.

At this point, it is useful to consider the expression that we, according to Eq. (3.10), would expect for the transition probability. Let $\boldsymbol{\psi}_k^{'} = (\psi_{k,1}^{'}, \ldots, \psi_{k,P}^{'})$ be the (unique) vector corresponding to $\text{bin}(\boldsymbol{\psi}_k^{'}) = k$. Under the definition of GCM (Def. 3.3.2), $\{\psi_{t,1}^{(i)}, \ldots, \psi_{t,P}^{(i)}\}$ are mutually independent given $\mathbf{x}_t^{(i)}$ and $\boldsymbol{\psi}_{\boldsymbol{t-1}}^{(i)}$, such that we would obtain

$$\tilde{\varphi}_{k'k,t}^{(i)}(\Omega) = \prod_{p=1}^{P} \left(\vartheta_{t,p}^{(i)}\right)^{\psi_{t,p}^{'}} \left(1 - \vartheta_{t,p}^{(i)}\right)^{1-\psi_{t,p}^{'}} . \tag{3.13}$$

Now let $\mathbb{P}(\psi_{t,y}^{(i)} = 1 | \boldsymbol{\psi}_{t-1}^{(i)}, \mathbf{x}_t^{(i)'}; \Omega) = \vartheta_{t,y}^{(i)}$, then after including the additional variable, the transition probability becomes

$$\begin{aligned} \varphi_{k'k,t}^{(i)}(\Omega) = &\prod_{p=1}^{P} \left(\vartheta_{t,p}^{(i)}\right)^{\psi_{t,p}^{'}} \left(1 - \vartheta_{t,p}^{(i)}\right)^{1-\psi_{t,p}^{'}} \\ &\times \left[\left(\vartheta_{t,y}^{(i)}\right)^{\psi_{t,y}^{'}} \left(1 - \vartheta_{t,y}^{(i)}\right)^{1-\psi_{t,y}^{'}}\right]^{\mathbb{1}_{\{k=\mathcal{C}_t\}}} . \end{aligned} \tag{3.14}$$

Now let $z_t^{(i)'} = \text{bin}\left(\boldsymbol{\psi}_t^{(i)'}\right)$, with $\boldsymbol{\psi}_t^{(i)'} = \left(\psi_{t,1}^{(i)}, \ldots, \psi_{t,P}^{(i)}, \psi_{t,y}^{(i)}\right)$ being the state in the new augmented state space, we obtain $f_y(y_t^{(i)} | z_t^{(i)'} = k, \mathbf{x}_t^{(i)}; \Omega) = 1$. Hence, indeed this new emission probability does not depend on Ω, and the emission can be ignored in the maximization.

□

It should also be noted that in many click models, including CZM and UBM, the emission probability does not depend on Ω by definition. Hence, many models do not require an additional latent state variable, and for those models the cardinality of the state space remains the same. Also, the analysis of Proposition 2 still holds if we would have $\vartheta_{t,y}^{(i)} = 1$, hence we can always augment the state space. For notational simplicity, we will redefine K and P to represent the size of the augmented state space and augmented number of latent variables at (i, t) respectively.

Dropping superscript (i) for a moment, the E-step is given by Proposition 3, which are also known as the forward-backward equations.

Proposition 3. *Let* $\alpha_{k,t} = \mathbb{P}(\mathbf{y}_{1:t}, z_t = k|\mathbf{x}_{1:t}; \Omega)$ *and* $\beta_{k,t} = \mathbb{P}(\mathbf{y}_{t+1:T}|z_t = k, \mathbf{x}_{t+1:T}; \Omega)$*, then we obtain:*

$$\hat{\alpha}_{k,t} = \mathbb{1}_{\{y_t=1|z_t=k\}} \sum_{k'=1}^{K} \hat{\varphi}_{k'k,t}(\mathbf{x}_t)\hat{\alpha}_{k',t-1}, \tag{3.15}$$

$$\hat{\beta}_{k,t} = \sum_{k'=1}^{K} \hat{\varphi}_{kk',t}(\mathbf{x}_{t+1})\hat{\beta}_{k',t+1}\mathbb{1}_{\{y_{t+1}=1|z_{t+1}=k'\}}, \tag{3.16}$$

and

$$\hat{h}_{k'k,t} = \frac{\hat{\beta}_{k,t}\hat{\alpha}_{k',t-1}\hat{\varphi}_{k'k,t}(\mathbf{x}_t)}{\sum_{\ell=1}^{K} \hat{\alpha}_{\ell,T}}\mathbb{1}_{\{y_t=1|z_t=k\}}, \tag{3.17}$$

with $\hat{\varphi}_{kk',t}(\mathbf{x}_t)$ *the estimated transition probability under* $\hat{\Omega}$*.*

Proof. Follows directly from [16]. □

Hence, Proposition 3 provides the desired explicit expressions of the E-step for the GCM.

3.4.2 Notes on the M-step

One useful property of GCM is that Eq. (3.11) can be split into multiple optimization problems.

Proposition 4. *Let* $\boldsymbol{\theta}_p$ *be the weight vector of dimension* m_p *for parameter* p*, such that* $\vartheta_{t,p}^{(i)} = f_{act}^{p}(\mathbf{x}_{t,p}^{(i)}; \boldsymbol{\theta}_p) = \mathbb{P}(\psi_{t,p}^{(i)} = 1|\mathcal{P}(\psi_{t,p}^{(i)}), \mathbf{x}_{t,p}^{(i)}; \Omega)$*, with* $\mathbf{x}_{t,p}^{(i)}$ *the covariate vector for parameter* p*, then*

$$\arg\max_{\Omega} Q_{GCM}(\Omega, \hat{\Omega}) = \left\{ \arg\max_{\boldsymbol{\theta}_p \in \mathbb{R}^{m_p}} Q_{GCM}^{'}(\boldsymbol{\theta}_p, \hat{\Omega}) \right\}_{p=1,\ldots,P}. \tag{3.18}$$

Proof. Let $\boldsymbol{\psi}_k^{'} = (\psi_{k,1}^{'}, \ldots, \psi_{k,P}^{'})$ be the (unique) vector corresponding to $\text{bin}(\boldsymbol{\psi}_k^{'}) = k$. Note that we now assume this represents the state in the augmented state space, i.e., the vector includes $\psi_{k,y}^{'}$, which was defined in Proposition 2. Furthermore, let $\mathcal{K}(k)$ be original state that we would obtain by omitting $\psi_{k,y}^{'}$ in $\boldsymbol{\psi}_k^{'}$.

We define $\mathcal{A}_{t,p,i}^{(+)}$, $\mathcal{A}_{t,p,i}^{(-)}$ as be the set of all transition $(k^{'},k) \in \{1,\dots,K+1\} \times \{1,\dots,K+1\}$ having $\vartheta_{t,p}^{(i)} \in (0,1)$, and for which $\psi_{t,p}^{(i)} = 1$ and $\psi_{t,p}^{(i)} = 0$ respectively. Then

$$\begin{aligned}
Q_{\mathrm{GCM}}^{'}(\Omega,\hat{\Omega}) &= \sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{k=1}^{K+1}\sum_{k'=1}^{K+1} \hat{h}_{k'k,t}^{(i)} \log \varphi_{k'k,t}^{(i)}(\Omega) \\
&= \sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{k=1}^{K}\sum_{k'=1}^{K} \hat{h}_{k'k,t}^{(i)} \sum_{\substack{\psi \in \{\psi_{t,1}^{(i)},\dots,\psi_{t,P}^{(i)}\} \\ \setminus \{\psi_{t,y}^{(i)}\}}} \log \mathbb{P}(\psi | \mathcal{P}(\psi_{t,p}^{(i)}), \mathbf{x}_{t,p}^{(i)}; \Omega) \\
&\quad + \mathbb{1}_{\{\mathcal{K}(k) = \mathcal{C}_{\mathcal{K}(k)}\}} \log \mathbb{P}(\psi_{t,y}^{(i)} = \psi_{k,y}^{'} | \mathcal{P}(\psi_{t,y}^{(i)}), \mathbf{x}_{t,y}^{(i)}; \Omega) \\
&= \sum_{p=1}^{P}\sum_{i=1}^{n}\sum_{t=1}^{T} \sum_{(k',k) \in \mathcal{A}_{t,p,i}^{(+)}} \hat{h}_{k'k,t}^{(i)} \log \mathbb{P}(\psi_{t,p}^{(i)} = 1 | \mathcal{P}(\psi_{t,p}^{(i)}), \mathbf{x}_{t,p}^{(i)}; \Omega) \\
&\quad + \sum_{(k',k) \in \mathcal{A}_{t,p,i}^{(-)}} \hat{h}_{k'k,t}^{(i)} \log \mathbb{P}(\psi_{t,p}^{(i)} = 0 | \mathcal{P}(\psi_{t,p}^{(i)}), \mathbf{x}_{t,p}^{(i)}; \Omega)
\end{aligned} \tag{3.19}$$

Hence, optimizing $Q_{\mathrm{GCM}}(\Omega,\hat{\Omega})$ is therefore equivalent to optimizing

$$\begin{aligned}
Q_{\mathrm{GCM}}^{p}(\boldsymbol{\theta}_p,\hat{\Omega}) = \sum_{i=1}^{n}\sum_{t=1}^{T} & \sum_{(k',k) \in \mathcal{A}_{t,p,i}^{(+)}} \hat{h}_{k'k,t}^{(i)} \log f_{\mathrm{act}}^{p}(\mathbf{x}_{t,p}^{(i)}; \boldsymbol{\theta}_p) \\
& + \sum_{(k',k) \in \mathcal{A}_{t,p,i}^{(-)}} \hat{h}_{k'k,t}^{(i)} \log(1 - f_{\mathrm{act}}^{p}(\mathbf{x}_{t,p}^{(i)}; \boldsymbol{\theta}_p))
\end{aligned} \tag{3.20}$$

for each $p \in \{1,\dots,P\}$ separately. $\square$

Proposition 5. *Let*

$$\hat{h}_{p}^{(+)} = \sum_{i=1}^{n}\sum_{t=1}^{T} \sum_{(k',k) \in \mathcal{A}_{t,p,i}^{(+)}} \hat{h}_{k'k,t}^{(i)}, \tag{3.21}$$

and

$$\hat{h}_p^{(-)} = \sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{(k',k)\in\mathcal{A}_{t,p,i}^{(-)}} \hat{h}_{k'k,t}^{(i)}. \tag{3.22}$$

In case $f_{act}^{p}(\boldsymbol{\theta}_p; \mathbf{x}_{t,p}^{(i)}) = \theta_p = \vartheta_p$*,* $\theta_p \in (0,1)$*, the M-step becomes:*

$$\hat{\vartheta}_p = \frac{\hat{h}_p^{(+)}}{\hat{h}_p^{(+)} + \hat{h}_p^{(-)}} \tag{3.23}$$

Proof. Follows directly from [49, pp. 27-30]. □

Hence, by modeling GCM as an IO-HMM, we find exactly the same EM updates as in [49]. However, IO-HMM also directly gives the E-step (Proposition 3), which in [49] still had to be derived for each variable separately. Furthermore, Bengio and Frasconi [16] also show that if $\mathbf{x}_t = x_t \in \{1, \dots, L\}$, for some $L \in \mathbb{N}$, then the M-step also has an analytical solution. Apart from introducing IO-HMM and its estimation using EM, Bengio and Frasconi [16] also model $\mathbb{P}(\psi_{t,p}^{(i)} | \mathcal{P}(\psi_{t,p}^{(i)}), \mathbf{x}_t; \Omega)$ as the output of a neural network.

3.4.3 Vectorized EM

Vectorized E-step To conclude this section, we will elaborate on how Q_{GCM} is optimized in the `gecasmo` package. This requires rewriting the E-step and M-step to vector notation. Since in the following we consider a single query session, we will again drop superscript i.

We define

$$A = \begin{pmatrix} \alpha_{10} & \dots & \alpha_{1T} \\ \vdots & \ddots & \vdots \\ \alpha_{K0} & \dots & \alpha_{KT} \end{pmatrix}, \quad B = \begin{pmatrix} \beta_{10} & \dots & \beta_{1T} \\ \vdots & \ddots & \vdots \\ \beta_{K0} & \dots & \beta_{KT} \end{pmatrix}, \tag{3.24}$$

and let

$$H_t = \begin{pmatrix} h_{11,t} & \dots & h_{1K,t} \\ \vdots & \ddots & \vdots \\ h_{K1,t} & \dots & h_{KK,t} \end{pmatrix}, \quad M_t = \begin{pmatrix} \varphi_{11,t} & \dots & \varphi_{1K,t} \\ \vdots & \ddots & \vdots \\ \varphi_{K1,t} & \dots & \varphi_{KK,t} \end{pmatrix},$$
$$D = \begin{pmatrix} \delta_{10} & \dots & \delta_{1T} \\ \vdots & \ddots & \vdots \\ \delta_{K0} & \dots & \delta_{KT} \end{pmatrix}, \tag{3.25}$$

with $\delta_{kt} = 1$ if state k is the click state at time t (0 otherwise). Note that we only have one click state, i.e., $\sum_{k=1}^{K} \delta_{kt} = 1$ for all $t \in \{0, \dots, T\}$. Let $\odot$ be the element-wise product. Using Expressions (3.15), (3.16), and (3.17), we obtain the following vectorized E-step (Alg. 1).

Algorithm 1: Vectorized E-step

```
1 $B_{1:K,T} \leftarrow 1$
2 $A_{1:K,0} \leftarrow D_{1:K,0}$
3 for $t \leftarrow 1$ to $T$ do
4     $A_{1:K,t} \leftarrow y_t \left[D_{1:K,t} \odot \left(M_t^\top A_{1:K,t-1}\right)\right] +$
        $(1 - y_t) \left[(1 - D_{1:K,t}) \odot \left(M_t^\top A_{1:K,t-1}\right)\right]$
5 for $t \leftarrow T$ to $1$ do
6     $\Lambda_t \leftarrow y_t \left(D_{1:K,t} \odot B_{1:K,t}\right) + (1 - y_t) \left[(1 - D_{1:K,t}) \odot B_{1:K,t}\right]$
7     $B_{1:K,t-1} \leftarrow M_t \Lambda_t^\top$
8     $H_t \leftarrow \left[\frac{\Lambda_t}{\sum_{k=1}^{K} \alpha_{k,T}} A_{1:K,t-1}^\top\right] \odot M_t^\top$
```

Vectorized M-step For convenience, we will slightly rewrite (3.20). Let

$$c^{(o)}_{k^{'}k,t,p,i} = \begin{cases} 1 & \text{if } (k^{'},k) \in \mathcal{A}^{(o)}_{t,p,i}, \quad o = + \\ -1 & \text{if } (k^{'},k) \in \mathcal{A}^{(o)}_{t,p,i}, \quad o = - \\ 0 & \text{otherwise} \end{cases}, \tag{3.26}$$

$$I^{(o)}_{t,p,i} = \begin{pmatrix} c^{(o)}_{11,t,p,i} & c^{(o)}_{12,t,p,i} & \cdots & c^{(o)}_{1K,t,p,i} \\ \vdots & \vdots & \ddots & \vdots \\ c^{(o)}_{K1,t,p,i} & c^{(o)}_{K2,t,p,i} & \cdots & c^{(o)}_{KK,t,p,i} \end{pmatrix}, \tag{3.27}$$

and

$$w^{(o)}_{t,p,i} = \left(\left(\hat{H}^{(i)}_t \odot I^{(o)}_{t,p,i}\right)\mathbf{1}\right)^{\top}\mathbf{1}, \tag{3.28}$$

with $\mathbf{1}$ a vector of ones of size K, $o \in \{+,-\}$. I.e., $w^{(o)}_{t,p,i}$ is the grand sum of the matrix obtain from $\hat{H}^{(i)}_t \odot I^{(o)}_{t,p,i}$.

Now let $w_{t,p,i} = w^{(+)}_{t,p,i} + w^{(-)}_{t,p,i}$. Expression (3.20) can then be rewritten as

$$\begin{aligned} Q^{p}_{\text{GCM}}(\boldsymbol{\theta}_p, \hat{\Omega}) &= \sum_{i=1}^{n}\sum_{t=1}^{T} w^{(+)}_{t,p,i} \log f^{p}_{\text{act}}(\mathbf{x}_t;\boldsymbol{\theta}_p) + (-w^{(-)}_{t,p,i}) \log\left(1 - f^{p}_{\text{act}}(\mathbf{x}_t;\boldsymbol{\theta}_p)\right) \\ &= \sum_{i=1}^{n}\sum_{t=1}^{T} |w_{t,p,i}| \left[\frac{\operatorname{sgn}(w_{t,p,i})+1}{2} \log f^{p}_{\text{act}}(\mathbf{x}_t;\boldsymbol{\theta}_p) \right. \\ &\quad \left. + \left(1 - \frac{\operatorname{sgn}(w_{t,p,i})+1}{2}\right) \log(1 - f^{p}_{\text{act}}(\mathbf{x}_t;\boldsymbol{\theta}_p))\right]. \end{aligned} \tag{3.29}$$

In the latter expression, we use that for some triple (t,p,i), either $w^{(+)}_{t,p,i}$ or $w^{(-)}_{t,p,i}$ is non-zero. Hence, we only have to keep one $T \times P \times n$ tensor in memory, and use the sign of the weight to determine whether the weight is with respect to $\psi^{(i)}_{t,p} = 1$ or $\psi^{(i)}_{t,p} = 0$. Although the former expression in (3.29) has the same form as binary cross-entropy, we have $w^{(+)}_{t,p,i} + (-w^{(-)}_{t,p,i}) \leq 1$, hence it should be interpreted as a weighted log loss function.

Finally, the GCM's EM algorithm is summarized in Algorithm 2.

Algorithm 2: GCM's EM procedure

```
  inputs : Initial parameter estimates $\hat{\Omega}$;
           Covariates matrices $\{X_p\}_{p=1,\ldots,P}$;
           Parameter activation functions $\{f^p_{\text{act}}(\cdot)\}_{p=1,\ldots,P}$;
           Parameter activation matrices $\{I^{(+)}_{p,t}, I^{(-)}_{p,t}\}^{t=t\ldots,T}_{p=1,\ldots,P}$;
           Transition matrices $\{M^{(i)}_t\}^{t=1,\ldots,T}_{i=1,\ldots,n}$;
           Tolerance parameter $\epsilon$;
  output: Fitted parameters $\{\hat{\boldsymbol{\theta}}_p\}_{p=1,\ldots,P}$
1 Choose some initial parameters $\{\hat{\boldsymbol{\theta}}_p\}_{p=1,\ldots,P}$, and $\Delta > \epsilon$;
2 while $\Delta > \epsilon$ do
3     Run the E-step (Alg. 1), given current estimates $\hat{\Omega}$;
4     Compute the weights for each variable $p$ using Expression (3.28);
5     Find for each variable $p$ new weights $\hat{\boldsymbol{\theta}}^{'}_p$ by optimizing (3.29);
6     $\Delta = \sum_{p=1}^{p} |\hat{\boldsymbol{\theta}}_p - \hat{\boldsymbol{\theta}}^{'}_p|$;
7     Set $\hat{\boldsymbol{\theta}}_p \leftarrow \hat{\boldsymbol{\theta}}^{'}_p$;
8 return $\{\hat{\boldsymbol{\theta}}_p\}_{p=1,\ldots,P}$;
```

3.5 Modeling click models as GCMs

3.5.1 Mapping click models to GCM

In Section 3.3.2, we introduced the generalized cascade model, and showed that optimization with EM is equivalent to that of the IO-HMM model, given that we add the click/skip at position $t-1$ to the input vector $\mathbf{x}^{(i)}_t$. In this section, we will show that common click models such as the Chapelle-Zhang model (CZM) [40], and the User Browser Model (UBM) [76], can be rewritten as GCMs. As a direct consequence, any simplification of CZM or UBM (such as the cascade model [53] or the dependent click model [95]) are also GCMs. In fact, any click model where the latent variables can be combined into one discrete latent state variable z_t, such that the remaining model is still Markovian w.r.t. its parent as defined in GCM's definition (Def. 3.3.2), can be modeled as GCM. Hence, also more general models than CZM and UBM can be modeled as GCM.

Definition 3.5.1. The Chapelle-Zhang model (CZM) [40] is a Dynamic Bayesian Network consisting of latent binary variables $\{R^{(i)}_{r_i(t)}, S^{(i)}_{r_i(t)}, E^{(i)}_t\}^{i=1,\ldots,n}_{t=1,\ldots,T}$, and observed binary variables $\{y_t^{(i)}\}_{t=1,\ldots,T}^{i=1,\ldots,n}$, where $R_{r_i(t)}^{(i)} = 1$ if the user finds the item at position $r_i(t)$ attractive (0 otherwise), $S_{r_i(t)}^{(i)} = 1$ if the user is satisfied with the item at position $r_i(t)$ after having clicked the item (0 otherwise), $E_t^{(i)} = 1$ if the item at position t is evaluated by the user (0 otherwise), and $y_t^{(i)} = 1$ if the item at position t is clicked (0 otherwise). Given SERP $\mathcal{S}_i$, users navigate through the SERP according to

$$\mathbb{P}(R_{r_i(t)}^{(i)} = 1) = \phi_{r_i(t)}^{(R)}; \tag{3.30}$$

$$\mathbb{P}(S_{r_i(t)}^{(i)} = 1 | y_t^{(i)} = c) = \begin{cases} \phi_{r_i(t)}^{(S)} & \text{if } c = 1 \\ 0 & \text{if } c = 0 \end{cases}; \tag{3.31}$$

$$\mathbb{P}(E_t^{(i)} = 1 | E_{t-1}^{(i)} = a, S_{r_i(t-1)}^{(i)} = b) = \begin{cases} \gamma & \text{if } a = 1, b = 0 \\ 0 & \text{otherwise} \end{cases}, \quad t > 1; \tag{3.32}$$

$$E_1^{(i)} = 1; \tag{3.33}$$

$$y_t^{(i)} = 1 \iff E_t^{(i)} = 1, R_{r_i(t)}^{(i)} = 1. \tag{3.34}$$

Definition 3.5.2. The User Browser Model (UBM) [76] is a Dynamic Bayesian Network consisting of latent binary variables $\{R_{r_i(t)}^{(i)}, E_t^{(i)}\}_{t=1,\ldots,T}^{i=1,\ldots,n}$, and observed variables $\{y_t^{(i)}\}_{t=1,\ldots,T}^{i=1,\ldots,n}$, all having the same interpretation as in CZM, for which we assume users navigate through a SERP according to

$$\mathbb{P}(R_{r_i(t)}^{(i)} = 1) = \phi_{r_i(t)}^{(R)}; \tag{3.35}$$

$$\mathbb{P}(E_t^{(i)} = 1 | y_{t'}^{(i)} = 1) = \gamma_{t't}, \quad t' < t; \tag{3.36}$$

$$y_0^{(i)} = 1; \tag{3.37}$$

$$y_t^{(i)} = 1 \iff E_t^{(i)} = 1, R_{r_i(t)}^{(i)} = 1. \tag{3.38}$$

In Propositions 6 and 7 we again drop super-/subscript i.

Proposition 6. *CZM is a GCM*

Proof. Let $\boldsymbol{\psi}_t = (S_{r(t-1)}, R_{r(t)}, E_t)$ and $z_t = \text{bin}(\boldsymbol{\psi}_t)$. For completeness, we introduce auxiliary variables $S_{r(0)} = 0$ and $x_t = 1$ for all t. Then indeed we find $\mathcal{P}(z_t) = \{z_{t-1}, y_{t-1}, x_t\}$ for $t > 1$, $\mathcal{P}(z_1) = \{x_1\}$, $\mathcal{P}(y_t) = \{x_t, z_t\}$, and $\mathcal{P}(x_t) = \emptyset$ for all t. $\square$

Hence, the idea of modeling click models as GCM is to combine the binary latent variables $\psi_{t,1}^{(i)}, \ldots, \psi_{t,P}^{(i)}$ between two subsequent clicks into one latent variable by using Equation (3.9). The size of the state space of z_t therefore becomes $\mathcal{O}(2^P)$. If after this transformation the click model is Markovian with respect to a subset of the parents as defined in GCM, then we can conclude that the model is a GCM, and we are allowed to use the IO-HMM machinery to estimate the model's parameters using EM.

Perhaps less obvious is that UBM can also be modeled as GCM. Looking at the definition of UBM (Definition 3.5.2), we find that UBM is Markovian in the last clicked item. Hence, it is not Markovian in the previous position, as is required by GCM. However, we can pass information about the last clicked item from position t to position $t+1$, by expanding the state space. This way, we can make UBM Markovian in the previous position.

Proposition 7. *UBM is a GCM*

Proof. From the definition of UBM, we find that it only has, at some time t, two latent variables: E_t and $R_{r(t)}$. The parents are given by $\mathcal{P}(R_{r(t)}) = \emptyset$, and $\mathcal{P}(E_t) = \{y_{t'}\}$, with $t^{'} = \max\{\tilde{t} \in \{0, \ldots, t-1\} | y_{\tilde{t}} = 1\}$. We now define $\mathbf{e}_t = (e_{1,t}, \ldots, e_{T,t})$, where

$$e_{t',t} = \begin{cases} \mathbb{1}_{\{t'<t\}} e_{t',t-1}(1-y_t) + \mathbb{1}_{\{t'=t\}} y_t & \text{if } t > 0 \\ 1 & \text{if } t = 0 \end{cases} . \tag{3.39}$$

I.e., $\mathbf{e}_t$ stores the last clicked item before position t. Since $e_{t',t} \in \{0,1\}$, we can define $\boldsymbol{\psi}_t = (\mathbf{e}_t, R_{r(t)}, E_t)$, and $z_t = \text{bin}(\boldsymbol{\psi}_t)$. As a result, we again obtain a one-dimensional discrete latent state space. Let $x_t = 1$ for all t, then indeed we find $\mathcal{P}(z_t) = \{z_{t-1}, y_{t-1}, x_t\}$ for $t > 1$; $\mathcal{P}(z_1) = \{x_1\}$; $\mathcal{P}(y_t) = \{x_t, z_t\}$, and $\mathcal{P}(x_t) = \emptyset$ for all t. $\square$

Following the derivations of Propositions 6 and 7, we could replace $\phi_{r_i(t)}^{(A)}$, $\phi_{r_i(t)}^{(S)}$, and γ with activation functions $f_{\text{act}}^{(A)}(\mathbf{x}_t^{(A)}; \boldsymbol{\theta}^{(A)})$, $f_{\text{act}}^{(S)}(\mathbf{x}_t^{(S)}; \boldsymbol{\theta}^{(S)})$, and

$f_{\text{act}}^{(E)}(\mathbf{x}_t^{(E)};\boldsymbol{\theta}^{(E)})$, in case of CZM. To ensure the output of these activation functions are probabilities, we ensure the activation functions all map to the interval $(0,1)$. Here $\boldsymbol{\theta} = (\boldsymbol{\theta}^{(A)}, \boldsymbol{\theta}^{(S)}, \boldsymbol{\theta}^{(E)})$ is a vector of unknown parameters, and $\mathbf{x}_t^{(A)}, \mathbf{x}_t^{(S)}, \mathbf{x}_t^{(E)}$ is observed data, stored in $\mathbf{x}_t$, i.e., $\mathbf{x}_t = (\mathbf{x}_t^{(A)}, \mathbf{x}_t^{(S)}, \mathbf{x}_t^{(E)})$. The same argument holds for UBM. As a consequence, also generalizations of CZM and UBM can be modeled as GCM (e.g., the Bayesian Sequential State Model [229], or many of the 'advanced click models' described in Chuklin et al. [49, Ch. 8]), as long as the Markovian properties of GCM are met.

In terms of the cardinality of the state space, we do notice that adding the vector of the last clicked item to the state space, as we have done with UBM, at first seems to lead to a considerable memory burden. Where CZM has a cardinality smaller than $2^3 + 1$ (as some combinations of $\{R_t, E_t, S_{t-1}\}$ are infeasible), for UBM this becomes $\mathcal{O}(2^T)$. However, in most cases the number of positions T can be taken small. Click probabilities have a geometric decay in T, and many empirical studies show that these probabilities are often negligible for $T > 10$ (e.g., [53, 22]). Hence, we do not expect such state spaces to become a burden.

3.5.2 Explicit mappings

In the following examples, we will provide an explicit mapping from CZM and UBM to GCM. That is, how the transition matrix and activation matrices of these two models could be defined to be able to run the GCM's EM procedure (Alg. 2). These example mappings have also been implemented and can by found on the `gecasmo`'s Github page [60].

CZM to GCM We will first consider how CZM can be written as GCM. As Proposition 6 already showed, CZM can be modeled as a GCM by defining the latent state as $\boldsymbol{\psi}_t^{(i)} = (S_{r_i(t-1)}^{(i)}, R_{r_i(t)}^{(i)}, E_t^{(i)})$, where $S_{r_i(t-1)}^{(i)}$, $R_{r_i(t)}^{(i)}$, and $E_t^{(i)}$ are interpreted as in Definition 3.5.1. One of the possible discretizations of the state space is given by Table 3.1. We have 3 binary variables, which normally would lead to a state space of size 8. However, we add an additional absorbing state O, and the two states having $(S_{r_i(t-1)}^{(i)} = 1, E_t^{(i)} = 1)$ are infeasible, therefore we end up with a total of 7 states. The click state represents the state in which an item is clicked, and as discussed before, we assume there is only one such state for each t. For convenience, we also added the corresponding click value $y_t^{(i)}$, though it is not part of the state space.

Using this state mapping and the probabilities introduced in Definition 3.5.1,

State	$S^{(i)}_{r_i(t-1)}$	$E^{(i)}_t$	$R^{(i)}_{r_i(t)}$	$y^{(i)}_t$
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3 ($\mathcal{C}_t$)	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6 (O)	-	-	-	-
-	*1*	*1*	*0*	0
-	*1*	*1*	*1*	1

Table 3.1: CZM state space mapping

and by writing $\bar{x} = 1 - x$, we find the transition matrix.

$$M_{i,t} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \bar{\gamma}_i \bar{\phi}^{(A)}_{r_i(t)} & \bar{\gamma}_i \phi^{(A)}_{r_i(t)} & \gamma_i \bar{\phi}^{(A)}_{r_i(t)} & \gamma_i \phi^{(A)}_{r_i(t)} & 0 & 0 & 0 \\ \bar{\gamma}_i \bar{\phi}^{(A)}_{r_i(t)} \bar{\phi}^{(S)}_{r_i(t)} & \bar{\gamma}_i \phi^{(A)}_{r_i(t)} \bar{\phi}^{(S)}_{r_i(t)} & \gamma_i \bar{\phi}^{(A)}_{r_i(t)} \bar{\phi}^{(S)}_{r_i(t)} & \gamma_i \phi^{(A)}_{r_i(t)} \bar{\phi}^{(S)}_{r_i(t)} & \bar{\phi}^{(A)}_{r_i(t)} \phi^{(S)}_{r_i(t)} & \phi^{(A)}_{r_i(t)} \phi^{(S)}_{r_i(t)} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.40}$$

To further define the model in GCM, we rely on the activation matrices $\{I^{(+)}_{\tilde{p},t}, I^{(-)}_{\tilde{p},t}\}^{t=t...,T}_{\tilde{p}=1,...,\tilde{P}}$, and the activation functions $\{f^p_{\text{act}}\}_{p=1,...,P}$. The activation matrices can directly be obtained from the transition matrix, using Equations (3.26), (3.27), and (3.40).

UBM to GCM As discussed in Proposition 7, we can model UBM as a GCM by passing information about the last clicked item through positions $1, \dots, T$. The state of the system at some (i, t) is defined by $\boldsymbol{\psi}_t^{(i)} = (\mathbf{e}_t^{(i)}, R_{r_i(t)}^{(i)}, E_t^{(i)})$. For simplicity, we will replace $\mathbf{e}_t$ by $t^{'}$, i.e., the last clicked item before t, such that we can use the following state space mapping (Table 3.2). Note that, although we have multiple click states, given $t^{'}$, there is still only one click state.

State	$t^{'}$	$E_t^{(i)}$	$R_{r_i(t)}^{(i)}$	$y_t^{(i)}$
0	0	0	0	0
1	0	1	0	0
2	0	0	1	0
3 ($\mathcal{C}_{t=0}$)	1	1	1	1
4	1	0	0	0
...	...	...	...	...
$4T$	T	0	0	0
$4T + 1$	T	1	0	0
$4T + 2$	T	0	1	0
$4T + 3$ ($\mathcal{C}_{t=T}$)	T	1	1	1
O	-	-	-	-

Table 3.2: UBM state space mapping

The UBM transition matrix is somewhat more involved than that of CZM. First, we consider two 4×4 matrices containing the probabilities of clicking/skipping some item at position t, given the last click was at position $t^{'}$. The click probability matrix is given by

$$M_{i,t^{'}t}^{(+)} = \mathbf{1}\begin{pmatrix}0, & 0, & 0, & \phi^{(R)}\gamma_{t^{'}t}\end{pmatrix}. \tag{3.41}$$

Here, the elements in the vector $(0, 0, 0, \phi^{(R)}\gamma_{t^{'}t})$ correspond with the states $(E_t^{(i)} = 0, R_{r_i(t)}^{(i)} = 0)$; $(E_t^{(i)} = 1, R_{r_i(t)}^{(i)} = 0)$; $(E_t^{(i)} = 0, R_{r_i(t)}^{(i)} = 1)$; and $(E_t^{(i)} = 1, R_{r_i(t)}^{(i)} = 1)$. The skip probability matrix is given by

$$M_{i,t^{'}t}^{(-)} = \mathbf{1}\begin{pmatrix}\bar{\phi}^{(R)}\bar{\gamma}_{t^{'}t}, & \bar{\phi}^{(R)}\gamma_{t^{'}t}, & \phi^{(R)}\bar{\gamma}_{t^{'}t}, & 0\end{pmatrix}. \tag{3.42}$$

Here $\mathbf{1}$ is a vector of ones with 4 elements. By combining these matrices into one transition matrix, we obtain

$$M_{i,t'\,t} = \begin{pmatrix} M^{(-)}_{i,0t} & M^{+}_{i,t'\,1} & \dots & M^{+}_{i,t'\,T} & \mathbf{0} \\ [0] & M^{(-)}_{i,1t} & \dots & M^{+}_{i,t'\,T} & \mathbf{0} \\ \vdots & \ddots & \ddots & \vdots & \mathbf{0} \\ [0] & \dots & M^{-}_{i,(T'-1)T} & M^{+}_{i,(T'-1)T} & \mathbf{0} \\ [0] & \dots & \dots & \dots & \mathbf{1} \\ 0 & \dots & \dots & \dots. & 1 \end{pmatrix}, \qquad (3.43)$$

with [0] a 4×4 matrix consisting of only zeros. The vectors $\mathbf{0}$ and $\mathbf{1}$ are both vectors of 4 elements with only zeros and ones respectively. The last row contains only scalars.

Again, the activation matrices can be found using Expressions (3.26), (3.27), and (3.40).

3.6 Conclusion

In this chapter, we presented two contributions. First, we proposed an alternative view on estimating click models. We defined the generalized cascade model (GCM), and showed that the expectation-maximization (EM) algorithm for GCMs is equivalent to the EM algorithm of an Input-Output Hidden Markov Model (IO-HMM), given that we add the observed click at position $t-1$ to the covariate vector at time t. As a consequence, we can directly use the EM algorithm for IO-HMMs, which provides us with an estimation procedure for GCMs without having to derive further expressions.

Since many click models, including the User Browser Model (UBM) and Chapelle-Zang Model (CZM), can be written as GCM, it can be applied to a large class of click models. Modeling click models as GCM also has the benefit that, if a click model can be modeled as a GCM and the covariate vector $\mathbf{x}_t$ has a finite countable state space, then the M-step has an analytical solution. This solution is equivalent to the result one would obtain by explicitly writing out the expressions of the E-step and M-step of the algorithm.

Second, we introduced the `gecasmo` package, which not only includes an EM implementation for IO-HMMs, but also takes care of the mapping from GCM to IO-HMM. As a result, one only needs to define, for some specific click model, the activation matrices and activation functions, from which the IO-

HMM's transition matrix can be determined. To provide some more clarity on how to define such activation matrices and functions, we included an example for UBM and CZM. Also implementation-wise, the usage of IO-HMM's EM algorithm is convenient. Where previous packages for estimating click models relied on the user having to implement the EM updates manually, `gecasmo` only requires the activation matrices and activation functions. Furthermore, as users can define their activation functions as neural networks, the model contains flexibility in writing click models as GCMs.

Further research may move into several directions. First, in this chapter we mainly focused on UBM and CZM as examples, as these two click models are both generalization of many other click models, but also because they are often used as a starting point for more complex models. As we have seen in the UBM case, even models with a non-first-order examination assumption can be modeled as GCM by expanding the state space, and under small list sizes, do not suffer too much from the curse of dimensionality. We expect that even further generalizations to UBM or CZM can be mapped to a GCM. It would be interesting to consider to what extent these generalizations can be modeled as GCM, and if such mapping remains practical.

Second, the method could benefit from further empirical validation. In particular, for implementational convenience, Expression (3.29) is numerically optimized in `gecasmo`. However, many click models have analytical solutions to the M-step (see Section 3.4.2). I.e., it would be beneficial to further examine the stability of the numerical method, and compare the results with the analytical solution.

Third, we limited our scope to EM procedures, which does not imply that alternative estimation procedures, such as variational inference or Markov Chain Monte Carlo, could not benefit from viewing click models as GCM. On the contrary, it would be interesting to consider whether modeling click models as GCM, or more general as IO-HMM, is also convenient under these estimation techniques.

3.7 Overview of notation

Table 3.3: Notation overview

Variable	**Interpretation**
$\{1,\dots,n\}$	The set of query sessions, indexed by i.
$\{1,\dots,V\}$	Set of all items, indexed by v.
$\mathcal{S}_i$	Set of items in SERP corresponding to query session i.
$\{1,\dots,T\}$	Set of all list positions, indexed by t.
Ω	Set of all parameters in the GCM.
K	Size of the (possibly augmented) state space.
$r_i(t)$	The item in list position v of query session i.
$\mathcal{C}_t$	The click state at time t.
$\boldsymbol{\psi}_t^{(i)} = (\psi_{t,1}^{(i)},\dots,\psi_{t,P}^{(i)})$	Vector of all latent variables at position t of query session i, which determines the latent state of GCM, and can uniquely be mapped to $z_t^{(i)}$.
$\mathbf{x}_{t,p}^{(i)} = (x_{t,p,1}^{(i)},\dots,x_{t,p,m_p}^{(i)})$	Vector of covariate values for parameter p.
$\vartheta_{t,p}^{(i)}$	Probability of a latent variable being one, given the previous state.
$f_{\text{act}}^{p}(\cdot)$	Activation function for paramter p.
$\boldsymbol{\theta}_p$	Weight vector for parameter p.
$y_t^{(i)}$	1 if the item in position t of SERP i is clicked, 0 otherwise.
$\mathcal{P}(X)$	The set of all parents of node X in a dynamic Bayesian network.
$z_t^{(i)}$	The single-dimensional discrete state that uniquely corresponds to some $\boldsymbol{\psi}_t^{(i)}$.
$\varphi_{k'k,t}^{(i)}$, $M_{i,t}$	Transition probability and transition matrix.
$\zeta_{k,t}^{(i)}$	Probability of being in state k at (i,t).
$f_y(\cdot)$	Emission probability of the IO-HMM.

Table 3.3 – *Continued*

Variable	**Interpretation**
$h_{k'k,t}$, $H_{i,t}$	Expectation of the joint distribution of being in state k at time t, and state $k^{'}$ at time $t-1$, given all available data and current estimate $\hat{\Omega}$.
$c^{+}_{k'k,t,p}$, $c^{-}_{k'k,t,p}$, $I^{+}_{p,t}$, $I^{-}_{p,t}$	Indicators of whether parameter p positively ($\psi_p = 1$) or negatively ($\psi_p = 0$) influences the transition probability between state $k^{'}$ and k.
$\alpha^{(i)}_{k,t}$, A_i	Forward probabilities in the forward-backward equations.
$\beta^{(i)}_{k,t}$, B_i	Backward probabilities in the forward-backward equations.
$\delta^{(i)}_{kt}$, D_i	Indicator of whether state k is a click state.
$R^{(i)}_{v}$, $\phi^{(R)}_{v,i}$	Indicator (probability) whether the user (of query session i) finds item v attractive (also referred to as the *relevance* in the click literature).
$S^{(i)}_{v}$, $\phi^{(S)}_{v,i}$	Indicator whether the user (of query session i) is satisfied with item v after a click.
$E_{i,t}$	Indicator whether the item at position t is evaluated.
γ_i	Search continuation probability, has different sub/superscripts based on the underlying click model.

Chapter 4

Click Model Simulation: A Case Study

In this chapter, we propose a click simulation model capable of simulating users' interactions with a search engine. We illustrate the simulation model by applying it to the problem of detecting unique users from the session data of a search engine. In real click datasets, the user initiating the session may be censored, as unique users are often determined by their cookies. Therefore, analyzing this problem using a click simulation model, for which we have an uncensored ground truth, allows for studying the effect of cookie churn itself. Furthermore, it allows for studying how well clustering algorithms perform in detecting clusters of sessions that originated from a single user. To cluster sessions, we compare various constrained DBSCAN*-type clustering algorithms. From this comparison, we find that even though the clusters found by the best DBSCAN*-type algorithm did significantly outperform other benchmark clustering methods, it performed considerably worse when using the observed cookie clusters. I.e., the results suggest that while clustering algorithms may be useful to detect similar users, cookie tracking remains the preferred method for tracking individual users.

4.1 Introduction

The current internet environment heavily relies on cookies for the enhancement of our internet browsing experience. These cookies are small pieces of data stored in the browser after being received from a server, along with a requested web page from that server. If the internet user pushes subsequent requests to the server, the cookie is send along, allowing the server to recognize the user and adjust its response accordingly. Hence, as cookies allow identifying users over multiple requests, they play a crucial role in session management, the personalization of websites and ads, and user tracking.

However, the usage of multiple devices, multiple browsers, and the focus on cookie management has made the problem of identifying single users over multiple sessions more complex. One study reports that as much as 20% of all internet users delete their cookies at least once a week, whereas this percentage increases to 30% when considering cookie churn on a monthly basis [55]. Not being able to track internet users may lead to sub-optimal behavior of search engines and online ads, as these have less information about previous search and click behavior to infer the user's preference for certain items from. As cookie churn and the usage of multiple devices censors the underlying user who is generating web traffic, we call this user censoring.

Following the 2015 ICDM and 2016 CIKM machine learning challenges [109, 50], cross-device matching has in recent years received considerable scientific attention. Cross-device matching refers to the problem of identifying individual internet users from a set of internet logs, where internet users may have been using multiple devices, and are therefore tracked as separate users. These studies, however, do have some limitations. Most approaches mentioned in the literature are limited to finding pairwise matches, i.e., pairs of sessions that are likely to originate from the same user. Such inference is however insufficient if one is interested in identifying exclusive session clusters consisting of more than two sessions.

Furthermore, there seems to be ambiguity in what exactly is meant by cross-device matching, or by session clustering, and to what extent successful methods applied to one problem will also work well on other problems. The ICDM and CIKM competitions consider the problem from the perspective of an online advertiser, advertising on multiple websites. Other approaches (e.g., [55, 129, 119]) consider the problem from the perspective of a single website. At this point, it is unclear whether approaches that work well on a single website are likely to be successful in the online advertisement case, and vice versa. Apart from this multi vs single website perspective, most datasets studied seem to originate from large advertisers or search engines, which raises the question of how generalizable these approaches are for websites or advertisers with less traffic or less heterogeneous searches.

To allow for sensitivity analysis in session clustering, we consider the single website perspective, and propose a single query click simulation model that allows for cookie censoring. Simulation has two main advantages: 1) by adjusting the simulation parameters, we may study how session clustering algorithms perform on websites with different user browsing characteristics. 2) It provides a ground truth which, due to user censoring, is only partially observed in real world datasets. Apart from the ground truth being useful in evaluating clustering algorithms, it also allows for studying the effects of user censoring on typical website statistics, such as the number of unique visitors on a website.

Besides introducing the simulation model, we compare several clustering approaches on multiple simulated datasets, where all clustering methods are based on the DBSCAN* and HDBSCAN* algorithms. To measure their effectiveness, we not only consider the error in terms of typical supervised clustering error measures such the Adjusted Rand Index, but also in terms of the error in estimating overall web statistics, such as the number of unique users, distribution of the number of sessions per user, and the user conversion distribution.

This chapter has the following structure. Section 4.2 discussed relevant literature related to session clustering. Section 4.3 discusses the simulation model, adaptions of (H)DBSCAN*, and experimental set-up. Section 4.4 discusses the results, whereas Section 4.5 discusses the implications and ideas for further research.

4.2 Related work

Simulating click behavior is definitely not a new concept. Chuklin et al. [49, pp. 75-77] suggests using pre-fitted click models for this purpose, where the model is pre-fitted to public click data sets. One risk of using pre-fitted models is an availability bias: can the characteristics of public click data sets, commonly provided by large search engines, easily be generalized over all search engines? Also, these data sets do not always provide the type of information one is interested in, such as the device used to initiate a session.

Fleder and Hosanagar [83] provide a generative approach for modeling user preferences, which we will discuss in more depth in Section 4.3.1. This model can be used as an alternative to model users' preferences for clicking items. Using pre-fitted or generative models do have a trade-off in terms of accuracy vs interpretability. E.g., the former may have an accurate estimate of users' item preference, but provides little understanding in why this preference over different products has a certain shape, whereas for the latter we expect this to be vice versa.

Several authors have studied how cookie censoring occurs. E.g., [52, 75, 55] consider cookie churn, whereas [176] considers specifically cross-device behavior. Results from these studies can be used to model cookie churn dynamics in a simulation model.

Identifying unique users from sessions can be seen as a specific case of the entity/identity resolution problem [119]. Though, what makes this problem special is the nature of the dataset, which typically consists of a large number of sessions, and of which clicks and web page meta-data (such as the URL) are the main sources of information. Because of these characteristics, entity resolution algorithms that do not account for these characteristics are likely to fail in their objective.

Session matching can be applied from an online advertiser's perspective, as was the case during the 2015 ICDM and 2016 CIKM machine learning challenges [179, 203, 233, 148, 223, 72, 191, 190, 213, 217], or from the perspective of a single website [55, 129, 119]. What remains unclear is whether these two problems can be considered the same. Although in both cases the main motivation for cookie-matching may be the same, e.g., increasing the click-through rate by means of personalization, the type of data is bound to be different. When advertising on multiple websites, the data seems to consist for a substantial part out of a large variety of visited URLs. Hence, proposed approaches from the advertisement perspective tend to rely heavily on natural language processing techniques [124, 217, 190, 191, 223, 148]. In case of a single website, the URLs or web pages' meta-data may be less diverse, and the "unique fingerprints" [181] users create while browsing a single website may therefore be less distinctive than on multiple websites.

Most often, both the single and multiple website perspectives are modeled as a binary classification problem. Here, a model is trained to identify whether two feature vectors describing sessions a and b originate from the same user. Striking is the success of tree-boosting methods for this task, which also in both the 2015 ICDM and 2016 CIKM machine learning competitions showed promising results. For a more in-depth discussion of the different methods applied in cross-device matching, modeled as a binary classification problem, we refer to Karakaya et al. [124]. Also worth mentioning is that many methods proposed to both the 2015 ICDM and 2016 CIKM competitions allow for overlapping user clusters. As the objective is to find pairs of sessions likely to originate from the same user, this may result in sessions a, b, c to be classified as $f(a, b) = 1$ and $f(a, c) = 1$, but $f(b, c) = 0$, f being the same user classifier. Such result may be undesirable in some practical applications.

A slight generalization of the cross-device matching problem is the cookie matching problem, in which we are given a set of sessions that are already partially labeled into users using cookies, but only partially due to some form of user censoring. I.e., cross-device matching and cookie matching only seem to differ in whether one assumes that user censoring only occurs because of cross-device usage, or also because of cookie churn. However, many approaches proposed in the literature can be applied to both problems. Hence, in these formulations, this distinction seems irrelevant. Various authors have considered the cookie matching problem, though under different names such as: 'user stitching' [119], 'visitor stitching' [129], or 'automatic identity linkage' [112]. Like in cross-device matching, these studies tend to allow for overlapping clusters.

One approach that does not allow for overlapping clusters is considered by [179], using classical bipartite matching algorithms such as the Hungarian algorithm. However, it is questionable to what extent these approaches are scalable, as the paper works with relatively small data sets. Furthermore, as users might have more than two cookies, bipartite matching will only solve part of the problem.

Dasgupta et al. [55] also move beyond pairwise clustering. The authors consider a combination of several similarity measures to determine whether two cookies originate from the same user, and apply a greedy graph coloring algorithm to cluster a session graph into user clusters. However, since multi-device usage as we observe on websites now was not that much the case when the paper was published in 2012, the algorithm strongly relies on the assumption that only one device is used at a time. This allowed the authors to only consider non-overlapping cookies in terms of time as candidates for cookie-matching, whereas in the multi-device case, such a constraint would not be able to identify unique users simultaneously using multiple devices.

In this chapter, we will use the term session clustering to relate to the problem of identifying unique users from session data. We prefer this term, as our methods do not per se require having partial session clusters from cookies, something that would be the case in cookie matching. Furthermore, we seek non-overlapping clusters, whereas ‘matching’ relates to training a classifier to predict whether two sessions originate from the same user. However, still many of the methods discussed so far are applicable to this formulation of the problem.

We take a similar approach as [190] towards session clustering. This approach first trains a classifier that predicts whether sessions a and b originate from the same user (that is, share the same cookie in the data). Next, each session forms pairs with its K nearest neighbor (K-NN) sessions, after which each nearest neighbor is re-evaluated using the classifier on whether the session and neighbor indeed originate from the same user. All sessions included in the remaining pairs are subsequently clustered using a greedy clustering algorithm, from which all sessions in a cluster are also added to the set of session pairs. This method shows some similarity with DBSCAN [78], where also K-NN is used to quickly identify similar data points. However, DBSCAN computes a (possibly approximate) minimum spanning tree (MST), from which a quick approximation can be made of the distances between points. Compared to the greedy clustering approach by [190], this leads to a considerable speed up. On the other hand, as DBSCAN misses a constraint on the maximum cluster size, we will turn to two of DBSCAN’s descendants: DBSCAN* and HDBSCAN* [31, 32], which can quite easily be adjusted to incorporate a maximum cluster constraint.

4.3 Methods

4.3.1 Simulating click data with cookie-churn

We consider a simulation model that models how users behave when interacting with a search engine. We choose to simulate behavior on a search engine, and not behavior on other types of websites, as there is extensive literature on what type of parametric models are accurate for modeling user behavior on search engines [49]. Furthermore, apart from dedicated search engines, a search tool is also a common feature on websites having other purposes [162]. Hence, we believe it is likely that this behavior is also found elsewhere.

To avoid overcomplexifying the simulation model, we only consider the case in which users push one or multiple homogeneous queries to the search engine. I.e., the query itself is the same over all users, and one user may repeat this query a number of times. Users do have different item preferences for the items the search engine may return. Furthermore, the item order may be different in each Search Engine Result Page (SERP). The simulation model consists of three parts. The first part models how users navigate through the SERP, the second part models how users' utility function is determined, while the third part models how the session generating user is censored due to cookie churn or the usage of multiple devices. For reference, Table 4.6 (Section 4.6) provides an overview of the most important variables in the simulation model.

Simulating SERP interactions

Two types of interactions between a user and the search engine are considered. First, users may push the (homogeneous) query to the server, and receive the SERP in response. Second, users may click on results in the SERP. At each interaction, the server checks whether the user has an active cookie. If not, a new cookie is send along with the server's response (which is either the SERP, or the content page of a particular item in the SERP), and stored in the user's browser. We will discuss how cookie churn is modeled in Section 4.3.1.

All interactions are stored by the server, which provides a label for the cookie, device and query-session. This query-session is defined in terms of a set of interactions with one SERP. Hence, where in practice a browser session is typically defined by some period of interaction, we deliberately choose to model a session as a set of interactions with one SERP, irrespective of the time between two interactions with this SERP.

To simulate clicks on a search engine, we employ the Simplified Chapelle-Zhang Model (SCZM) [40]. Although this model is known in the literature as the Simplified Dynamic Bayesian Network model (SDBN), we renamed the model for the reason discussed in Section 3.2. We choose to use SCZM for two reasons: 1) the model, though simple, seems to perform reasonably well in comparison with other parametric click models when predicting clicks [49]. 2) SCZM captures the ordering effect of items in the SERP. I.e., users may not always reflect their preferences correctly in their clicks, as their behavior is also determined by how items are ordered. Including this 'cascade effect' provides more realistic results.

To describe the simulation model, the following notation will be used. Let $i \in \{1, \ldots, n\}$ be a query-session, which produces a SERP of unique items $\mathcal{S}_i \subseteq \mathcal{V}$, with $\mathcal{V} = \{1, \ldots, V\}$ the set of all items, indexed by v. We assume all SERPs $1, \ldots, n$ to have the same number of items T. Let $u_i \in \mathcal{U}$ denote the user initiating query-session i, with $\mathcal{U} = \{1, \ldots, U\}$ the set of all users. The user index u is used instead of u_i in case the precise query-session i is irrelevant. $r_i(t)$ denotes the item at position t in query-session i. Likewise, $r_i^{-1}(v)$ gives the position of user u in query-session i, and $r_i^{\max}$ denotes the largest position of a clicked item in $\mathcal{S}_i$, where $r_i^{\max} = 0$ if no items were clicked during query-session i.

SCZM considers three latent variables: $R_v^{(i)}$ denotes whether user u_i is attracted to item v during query-session i. This variable is also known as the relevance of item v for the user initiating session i. The probability of item v being relevant to user u in session i is given by $\phi_{u,v}^{(R)}$. $S_v^{(i)}$ denotes whether user u_i is satisfied with item v after having clicked the item, which happens with probability $\phi_{u,v}^{(S)}$, and $E_t^{(i)}$ denotes whether user u_i will evaluate the item in position t during query-session i. Whether the item at position t in SERP i is clicked is denoted by the binary variable $y_t^{(i)}$.

The model follows the cascade hypothesis, that is, it assumes a user always evaluates the first item ($E_1^{(i)} = 1$ for all $i = 1, \ldots, n$), after which the user decides to evaluate subsequent items in the list according to the perceived attraction and satisfaction of the previous evaluated items in the list according to

$$E_1^{(i)} = 1; \tag{4.1}$$

$$\mathbb{P}(R_v^{(i)} = 1) = \begin{cases} \phi_{u_i,v}^{(R)} & \textit{if } v \in \mathcal{S}_i \\ 0 & \textit{otherwise} \end{cases}; \tag{4.2}$$

$$\mathbb{P}\left(S_v^{(i)} = 1 \mid y_{r_i^{-1}(v)}^{(i)} = 1\right) = \begin{cases} \phi_{u_i,v}^{(S)} & \textit{if } v \in \mathcal{S}_i \\ 0 & \textit{otherwise} \end{cases}; \tag{4.3}$$

$$y_t^{(i)} = 0 \Rightarrow S_{r_i(t)}^{(i)} = 0; \tag{4.4}$$

$$E_{t-1}^{(i)} = 1, S_{r_i(t-1)}^{(i)} = 0 \iff E_t^{(i)} = 1, \quad t > 1; \tag{4.5}$$

$$y_t^{(i)} = 1 \iff E_t^{(i)} = 1, R_{r_i(t)}^{(i)} = 1. \tag{4.6}$$

To come up with reasonable values for $\phi_{u,v}^{(R)}$ and $\phi_{u,v}^{(S)}$, we used the same approach as in [83]. That is, users are represented by the vectors $\boldsymbol{\eta}_u = \left(\eta_1^{(u)}, \eta_2^{(u)}\right)$, $u \in \mathcal{U}$, where $\eta_1^{(u)}$ and $\eta_2^{(u)}$ are drawn from two independent standard normal distributions. Likewise, all items can be represented by the vectors $\boldsymbol{\psi}_v = \left(\psi_1^{(v)}, \psi_2^{(v)}\right)$, where again $\psi_1^{(v)}$ and $\psi_2^{(v)}$ are drawn from independent standard normal distributions. The probabilities $\phi_{u,v}^{(R)}$, and $\phi_{u,v}^{(S)}$ are then determined by the multinomial logits

$$\phi_{u,v}^{(R)} = \frac{e^{\omega_{u,v}+\nu^{(R)}}}{\sum_{v' \in \mathcal{V}\setminus\{v\}} e^{\omega_{u,v'}} + e^{\omega_{u,v}+\nu^{(R)}}}, \tag{4.7}$$

$$\phi_{u,v}^{(S)} = \frac{e^{\omega_{u,v}+\nu^{(S)}}}{\sum_{v' \in \mathcal{V}\setminus\{v\}} e^{\omega_{u,v'}} + e^{\omega_{u,v}+\nu^{(S)}}}, \tag{4.8}$$

with

$$\omega_{u,v} = -q \log \delta(\boldsymbol{\eta}_u, \boldsymbol{\psi}_v). \tag{4.9}$$

Here δ is some distance function, in our case Euclidean distance. $q \in \mathbb{R}^+$ is some constant value that models the users' preferences towards nearby items, and $\nu^{(R)}$, $\nu^{(S)}$ are salience parameters for attraction and satisfaction respectively.

The order in which items are presented is determined as follows. First, during a warm-up phase, we simulate clicks for $U_{\text{warm-up}}$ users, while randomly ordering the items such that all have equal probability of being positioned at positions $t = 1, \ldots, T$. Next, we estimate the overall probability of each item being found attractive, and we use these probabilities as weights to determine the item order for subsequent query-sessions. More specifically, for each query-session i, we draw items $\mathcal{S}_i$ from a multinomial distribution with parameters $\hat{\phi}_v / \sum_{v\in\mathcal{V}} \hat{\phi}_v$, $v = 1, \ldots, V$; without replacement. The estimate of overall attraction is given by [49, p. 26],

$$\hat{\phi}_v = \frac{1}{|\mathcal{I}_v|} \sum_{i \in \mathcal{I}_v} y^{(i)}_{r_i^{-1}(v)}, \tag{4.10}$$

with

$$\mathcal{I}_u = \left\{ \mathcal{S}_i : v \in \mathcal{S}_i, r_i^{-1}(v) \leq r_i^{\max} \right\}. \tag{4.11}$$

To avoid $\hat{\phi}_v$ to be (close to) zero, we impose a minimum probability of 10^{-5} for all $v \in \mathcal{V}$.

Cookie censoring

Cookie censoring is incorporated in the simulation model in two ways: by incorporating time and letting cookies churn after some random time $\mathcal{T}$, and by switching from device d to some other device $d^{'}$. First, we consider the cookie lifetime $\mathcal{T}^{\text{cookie}}_{u,o,d}$ for the o-th cookie of user u on device d, and the user lifetime $\mathcal{T}^{\text{user}}_{u}$. Whenever the cookie lifetime of cookie o ends, but the current user lifetime is strictly smaller than $\mathcal{T}^{\text{user}}_{u}$, a new cookie $o^{'}$ is created, which lifetime is drawn from the cookie lifetime distribution F^{cookie}. For a period of $\mathcal{T}^{\text{cookie}}_{u,o',d}$, all click behavior of user u on device d will now be registered under cookie $o^{'}$.

Second, after each query-session a user may switch from device d to $d^{'}$, which happens according to transition matrix P. Whenever a user switches devices, we consider whether the user has used this device before. If not, a new cookie $o^{'}$ is created, and we draw a new cookie lifetime from F^{cookie}. However, the cookie lifetime $\mathcal{T}^{\text{cookie}}_{u,o,d}$ does not end prematurely when the user switches from device d to $d^{'}$. If later on the user switches back to device d while the cookie lifetime $\mathcal{T}^{\text{cookie}}_{u,o,d}$ has not ended, the behavior of user u is again tracked via cookie o until another device switch occurs or cookie o churns.

Putting this censoring into practise requires us to provide five distributions: 1) a distribution F^{abs} for the time between query-sessions, which following [75] we will refer to as the *absence time*, 2) a distribution for the cookie lifetime (F^{cookie}), 3) a distribution for the user lifetime (F^{user}), 4) the device transition matrix P, and 5) the initial device probability F^{device}.

For the absence time distribution, we use some results from [75]. Although Dupret and Lalmas [75] fitted a Cox survival model to user absence data in order to estimate user lifetimes, we refitted the data mentioned in the paper with a different model for two reasons. First, there is ambiguity in the method used to model absence time. The authors fit a Cox survival model with one covariate. As the (log-)likelihood of a Cox survival model omits the estimation of the base hazard, the method for estimating this base hazard should be provided (e.g., the Breslow estimator). However, the paper does not report which method was used to fit the baseline hazard. Second, results from Dasgupta et al. [55] on cookie churn suggests that, when taking into account longer periods than 7 days, absence time has a fat-tailed distribution. We found that a Pareto-I with scale parameter $m = 1$ and shape $\alpha = 0.11$ seems to fit the data from [75] approximately well. This distribution was therefore used to model F^{abs}. To allow for absence times smaller than 1 (but still positive), we subtracted one from all drawn lifetimes.

To model the cookie lifetime, we used the results from [55], who find that a hyper-exponential distribution with one over the rate being equal to 50 seconds (with probability .06), 25 minutes (with probability .07), 14 hours (with probability .07), 15 days (with probability .18), and 337 days (with probability .62), fits reasonably well. Here, cookie lifetime is defined as the time difference between the first and last observed action from a single cookie. We consider time at a minute scale, and therefore rounded up the first phase (50 seconds) to one minute.

The user lifetime is obtained by sampling from N cookie lifetime distributions, where N itself is drawn from a geometric distribution with parameter ρ. As the cookie lifetime distribution is modelled as a hyper-exponential, we will refer to this distribution as a repeated hyper-exponential distribution. Although we sample from the cookie life time distribution, the user lifetime is independent from the cookie lifetimes: they only share the underlying hyper-exponential distribution, not the realizations of that distribution.

To model device transition matrix P, we use the results from [176], who study device transitions between four devices: a PC, tablet, smartphone and game console. We adopted the transition probabilities found in this paper, where we dropped the game console as the found transition probabilities from and to this device were marginal. After dropping the game console, the probabilities were normalized to obtain transition matrix P. The initial device probability distribution F^{device} is also obtained using the results from [176], and is modeled as a multinomial distribution with parameter $\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3)$; π_1, π_2, π_3 being the probability of the PC (Dev. 1), tablet (Dev. 2), and smartphone (Dev. 3) being the first device respectively. The normalized initial and transition probabilities from [176] are given in Table 4.1

Table 4.1: Initial device and device transition probabilities adopted from [176]

	$\boldsymbol{\pi}$	Dev. 1	Dev. 2	Dev 3
Dev. 1	.64	.9874	.0042	.0084
Dev. 2	.11	.00256	.9697	.0046
Dev. 3	.25	.029	.0018	.9773

Summary of the simulation procedure

The entire simulation procedure is given in Algorithms 3, 4, and 5. Algorithm 3 describes how user preferences are obtained and how the overall popularity is determined, whereas Algorithms 4 and 5 (see Section 4.6) describe how clicks and cookie churn are simulated over a set of users.

For convenience, we have written the set of warm-up users as $\mathcal{U}_{\text{warm-up}}$, $\hat{\boldsymbol{\phi}} = (\hat{\phi}_1, \ldots, \hat{\phi}_V)$, and $\mathbf{y}_i = (y_1^{(i)}, \ldots, y_T^{(i)})$. The location and scale parameter of the Pareto-I distribution are written as m and α, whereas the rate and rate probability of the hyper-exponential distribution are given by the vectors $\boldsymbol{\lambda}$ and $\mathbf{p}$. Last, let I_d be a 3×3 matrix where the d-th column contains all ones, whereas the rest of the matrix contains all zeros.

The simulation iterates over all users, where for each user new query-sessions are simulated until the user lifetime has elapsed. For each user, first the initial device is drawn, along with a cookie lifetime for that user on that device, and the total user lifetime. Next, query-sessions are simulated for each user in four steps. First, $\mathcal{S}_i$ is (iteratively) drawn using the overall item popularity $\hat{\boldsymbol{\phi}}$, and we simulate clicks using the SCZM model described in Section 4.3.1, which are stored in dataset $\mathcal{D}$. Second, we simulate the time until the next session. Third, the device of the next session is determined. Fourth, we check whether the last cookie on the new device has churned. If so, a new cookie is created with a corresponding new cookie lifetime.

Although Algorithm 4 assumes all users arrive at $t = 0$, we shift all times after the simulation to obtain click behavior spread out over time. Here we assume a Poisson arrival process with rate γ. I.e., the first query-session of user u starts some exponentially distributed time after the initial query-session of user $u - 1$. Note that these inter-first session times only depend on the time of the first session of the previous user, not on any other subsequent behavior of that user.

4.3.2 Session clustering

(H)DBSCAN*

Hierarchical clustering using Minimum Spanning Trees (MST) Before we discuss the adjustment made to the HDBSCAN* and DBSCAN* algorithms, we will first briefly describe the two algorithms. We first discuss the overlapping part in both algorithms, after which we discuss their differences. Following the terminology by [31, 32] and [171], let $X = \{X_1, \ldots, X_n\}$ be a

Algorithm 3: User simulation procedure

```
inputs : Users $\mathcal{U}$, items $\mathcal{V}$, warm-up users $\mathcal{U}_{\text{warm-up}}$, device
         probabilities $\boldsymbol{\pi}$ and $P$, $F^{\text{cookie}}$, parameters $\mathbf{p}$ and $\boldsymbol{\lambda}$, user
         lifetime geometric phases parameter $\rho$, and attraction and
         satisfaction parameters $\boldsymbol{\phi}^{(R)}$ and $\boldsymbol{\phi}^{(S)}$
output: Simulation realization $\mathcal{D}$
1 Draw $\eta_1^{(u)}, \eta_2^{(u)}, \psi_1^{(v)}, \psi_2^{(v)}$ i.i.d. from a standard normal distribution
    for all $v \in \mathcal{V}$ and $u \in \mathcal{U}$;
2 Compute similarities $\omega_{u,v}$ according to (4.9);
3 Compute the probability of attraction and satisfaction, using (4.7);
4 Set $\hat{\phi}_v \leftarrow 1$ for all $v \in \mathcal{V}$;
5 $\mathcal{D}_{\text{warm-up}} \leftarrow$
    SIMULATE_CLICKS($\mathcal{U}_{\text{warm-up}}, \mathcal{V}, \hat{\boldsymbol{\phi}}, \boldsymbol{\pi}, P, \mathbf{p}, \boldsymbol{\lambda}, \rho, \boldsymbol{\phi}^{(R)}, \boldsymbol{\phi}^{(S)}$);
6 Recompute $\hat{\boldsymbol{\phi}}$ according to (4.10);
7 $\mathcal{D} \leftarrow$ SIMULATE_CLICKS($\mathcal{U} \setminus \mathcal{U}_{\text{warm-up}}, \mathcal{V}, \hat{\boldsymbol{\phi}}, \boldsymbol{\pi}, P, \mathbf{p}, \boldsymbol{\lambda}, \rho, \boldsymbol{\phi}^{(R)}, \boldsymbol{\phi}^{(S)}$);
8 return $\mathcal{D}$;
```

set of data points, let $\kappa_k(X_i)$ be the distance from point X_i to its k-th nearest neighbor (for some given value of $k \in \mathbb{N}$), and let $\delta(X_i, X_{i'})$ be some distance measure between points X_i and $X_{i'}$. Based on this original distance measure, DBSCAN* considers an alternative distance measure, which is named the *mutual reachability distance*, and is defined as follows:

$$\delta_k^{\text{mreach}}(X_i, X_{i'}) = \begin{cases} \max\{\kappa_k(X_i), \kappa_k(X_{i'}), \delta(X_i, X_{i'})\} & \textit{if } X_i \neq X_{i'} \\ 0 & \textit{if } X_i = X_{i'} \end{cases} . \tag{4.12}$$

Although DBSCAN* does not specify the exact distance measure δ, we will (like in Section 4.3.1) assume this is Euclidean distance. The main motivation for introducing this mutual reachability distance is to better identify different clusters with high density of arbitrary shape, as the measure tends to push different high density clusters further apart.

Given the mutual reachability distance, (H)DBSCAN* represents each data point as a node in a complete weighted graph G, where the weights are simply the mutual reachability distances between data pairs. Using G, the algorithm first computes a minimum spanning tree (MST), which allows for fast identification of clusters. The MST is also used to approximate distances: the distance between two non-adjacent points X_i and $X_{i'}$ in the MST can be approximated by the path length $X_i \to X_{i'}$ in the MST. At the same time, this distance is a lower bound on the actual distance: otherwise, $X_i \to X_{i'}$ would be adjacent in the MST.

From this MST, one can build a dendogram of the data points in an agglomerative manner. First, (H)DBSCAN* assigns each data point $X_1, \ldots, X_n$ to separate clusters $\mathcal{B}_1^0, \ldots, \mathcal{B}_n^0$. Here the superscript is used to indicate the hierarchy level of the cluster, which at this stage is zero. Second, it iterates through the edges in G, increasing in terms of their weights. For some edge $(i, i^{'})$ having the smallest edge weight, it finds the two clusters with the highest hierarchy levels $h_i^{\max}$ and $h_{i'}^{\max}$, to which i and $i^{'}$ are assigned to respectively. Next, it and creates a new cluster $\mathcal{B}_j^{\max\{h_i^{\max}, h_{i'}^{\max}\}+1}$, which includes all data points included in the highest hierarchy clusters to which X_i and $X_i^{'}$ were previously assigned to. If this process is repeated for all edges in G, the last edge will create a cluster containing all data, which occurs at level H.

DBSCAN* The construction of the dendogram occurs both in DBSCAN* and HDBSCAN* in the same manner. However, as both methods wish two find non-overlapping clusters, the two methods split ways from there. In DBSCAN*, one would take some value $\epsilon \in \mathbb{R}^+$, and remove all cluster merges in the dendogram that were merged with a weight strictly greater than the chosen maximum distance ϵ. This would lead to a set of disconnected binary trees $\mathcal{T}$, and a set of singleton points $\mathcal{N}$. The singleton points are points for which their k-th nearest neighbor is already at a further distance than ϵ, and these points are consequently labeled as noise. All data points in one tree $\tau \in \mathcal{T}$ are labeled as one cluster.

HDBSCAN* The underlying assumption of cutting the dendogram at level ϵ, is that all clusters have (approximately) the same density. This density is in HDBSCAN* approximated by $\theta = 1/\epsilon$, i.e., close points imply high density. HDBSCAN* allows for different cut-off levels of ϵ, or similarly of θ, where the optimal cut-off level for some cluster is determined via the notion of *relative excess of mass*, which we will introduce in a moment.

More precisely, let M be some given minimum cluster size. To somewhat simplify notation, we let index j refer to any cluster, irrespectively of hierarchy h, such that h can be dropped. HDBSCAN* first creates a *condensed tree* from the dendogram in the following way. It starts at the root of the dendogram, having label j_0, and considers its children. These were merged at some density $\theta_{j,j'}$, merging two clusters with labels j and $j^{'}$. It then considers three options:

1) if both children have less than M points, all points in $\mathcal{B}_j$ and $\mathcal{B}_{j'}$ "fall-out" of the cluster at density $\theta_{j,j'}$, implying that for densities greater than $\theta_{j,j'}$ all points in $\mathcal{B}_j$ and $\mathcal{B}_{j'}$ are labeled as noise. 2) If only one cluster $\mathcal{B}_j$ has less than M points, all points in $\mathcal{B}_j$ fall-out at density $\theta_{j,j'}$, while the parent cluster label (j_0) is now continued for all observations in $\mathcal{B}_{j'}$. I.e., we replace label j' by j_0, and as a result the exact cluster j_0 now refers to depends on whether we pick a density larger or smaller than $\theta_{j,j'}$. 3) If both children have more than M observations, clusters $\mathcal{B}_j$ and $\mathcal{B}_{j'}$ keep their labels j and j'. I.e., label j_0 is not continued, and clusters $\mathcal{B}_j$ and $\mathcal{B}_{j'}$ are considered separate clusters for densities larger than $\theta_{j,j'}$. After both children have been relabeled, this process is repeated using these new labels until all nodes have been relabeled.

The resulting condensed tree is essentially still the same as the original dendogram, but with different labels. I.e., by continuing the parent (option 2), some labels now may refer to different clusters, dependent on density θ. Let $\{1, \ldots, m\}$ be the resulting set of labels from relabeling. For each label $j \in \{1, \ldots, m\}$, let $\mathcal{B}_j$ be the set of observations labeled j at the minimum density for which j exists. Furthermore, let $\theta_j^{\max}(X_i)$ and $\theta_j^{\min}(X_i)$ be the densities at which observation X_i falls off cluster j and the density at which X_i first occurs in cluster j respectively. Note that $\theta_j^{\min}(X_i)$ is either zero (when j is the label continued from the root node), or the density at which cluster j splits off from its parent, hence it has the same value for all $X_i \in \mathcal{B}_j$.

Clusters $\{\mathcal{B}_1, \ldots, \mathcal{B}_m\}$ may still be overlapping. To find non-overlapping clusters, HDBSCAN* introduces the *relative excess of mass* of cluster j as $\sigma(j)$, which is defined by:

$$\sigma(j) = \sum_{X_i \in \mathcal{B}_j} \left[\theta_j^{\max}(X_i) - \theta_j^{\min}(X_i)\right]. \tag{4.13}$$

The relative excess of mass has an intuitive argument for clustering. Large values for $\sigma(j)$ imply that when increasing the density, the cluster remains more or less intact (apart from some noise points splitting off at higher densities). As a result $\theta_j^{\max}(X_i) - \theta_j^{\min}(X_i)$ becomes large. I.e., the relative excess of mass can be used as a measure of cluster quality. Hence, HDBSCAN* optimizes the sum of relative excess of mass over a subset of clusters $\{\mathcal{B}_1, \ldots, \mathcal{B}_m\}$ such that this subset is non-overlapping.

Introducing maximum cluster sizes to HDBSCAN* and DBSCAN*

To return to the problem at hand: identifying small session clusters from the set of all sessions that may be originating from the same user, HDBSCAN* and DBSCAN* can obviously be used for this purpose. Apart from the earlier discussed benefit of speed by clustering via MST, incorporating noise points would also intuitively make sense in identifying potential users from sessions: we would expect that quite a large (though unknown) percentage of all sessions might still be from users only initiating a single session.

By tweaking parameters k, (the k-th nearest neighbor in nearest neighbor distance κ_k), ϵ (dendogram cut-off point in case of DBSCAN*), and M (minimum number of points before a cluster is considered noise in HDBSCAN*) one can obtain session clusters that obey a maximum cluster size $\beta \in \mathbb{N}$. However, some early experiments with DBSCAN* and HDBSCAN* showed that the resulting clusters tended to either very large clusters, or labeled (almost) every point as noise. For that reason, we chose to adjust both algorithms, in order to obtain more small clusters having a size smaller than β.

To impose the clusters to be more fine grained, we impose a restriction on the maximum cluster size of the clusters found by (H)DBSCAN*. We do so in three different ways: max-size DBSCAN* (MS-DBSCAN*) imposes this restriction on DBSCAN*, whereas MS-HDBSCAN*$^-$ and MS-HDBSCAN*$^+$ are two ways to impose the restriction on HDBSCAN*.

First we consider MS-DBSCAN*. This algorithm is only a slight adaptation to the DBSCAN* algorithm described in Section 4.3.2. Given the binary trees $\mathcal{T}$, obtained by removing all nodes and edges in the dendogram above distance ϵ, we further remove all cluster nodes j for which $|\mathcal{B}_j| > \beta$. Doing so results in two new sets: $\widetilde{\mathcal{N}}$ and $\widetilde{\mathcal{T}}$, again representing singleton points that we assume to be noise, and all points in a tree $\tau \in \widetilde{\mathcal{T}}$ receive the same cluster.

Second are the adaptations of HDBSCAN*. The first steps of these two adaptations are the same. First, all clusters $\mathcal{B}_j \in \{\mathcal{B}_1, \dots, \mathcal{B}_m\}$ having $|\mathcal{B}_j| > \beta$ are removed from the dendogram. This, like in DBSCAN*, gives two sets: noise points $\mathcal{N}$ and trees $\mathcal{T}$. Second, for each sub-tree $\tau \in \mathcal{T}$, we again optimize the the total relative excess of mass subject to non-overlapping clusters. The difference between MS-HDBSCAN*$^-$ and MS-HDBSCAN*$^+$ arises when a leaf node of the condensed tree (that is, a label that does not split at some larger density into two new labels, though noise points may split off) of some condensed sub-tree τ is in the set of optimal non-overlapping clusters. In case of MS-HDBSCAN*$^-$, all observations in $\mathcal{B}_j$ are given the same label, whereas in case of MS-HDBSCAN*$^+$, these are considered noise.

Session cluster re-evaluation

As one might have noticed, so far we have not used any information from the cookies. I.e., knowing which sessions have the same cookie could provide valuable information about the underlying user. In particular, we wish to train a model that can function as an alternative to standard distance measures δ, such as Euclidean or Manhattan distance, which we then again can plug into the adapted (H)DBSCAN* algorithms described in Section 4.3.2.

Obtaining session clusters with re-evaluation is done as follows. Assume we have a trained classifier $\hat{f}(X_i, X_{i'})$, which returns the probability of X_i and $X_{i'}$ originating from the same user. First, like in [190], we find for each point X_i the K nearest neighbors, which gives us a set $\mathcal{X}$ of all nearest neighbor session pairs. Second, we compute $-\log(\hat{f}(X_i, X_{i'}))$ for all $(X_i, X_{i'}) \in \mathcal{X}$, and fill this into a (sparse) $n \times n$ distance matrix W. For all pairs $(X_i, X_{i'}) \notin \mathcal{X}$, we assume the distance is some large value $\delta_{\max}$, which allows us to store W efficiently, and greatly speeds-up computations compared to evaluating all pairwise same user probabilities. Distance matrix W can subsequently be used as distance measure δ in the algorithms discussed in Section 4.3.2 to obtain new session clusters.

To train classifier $\hat{f}$, we first cluster a training set according to one of the models discussed in Section 4.3.2, using Euclidean distance for δ. Second, for each cluster we add all unique session pairs into some training set $\mathcal{X}_{\text{clust}}$. Next, we start using the observed cookies: we treat each cookie as a cluster and determine all session pairs in these cookie clusters, where this set of pairs is denoted by $\mathcal{X}_{\text{cookie}}$. To determine for each session pair $(X_i, X_{i'})$ the correct label, we use the information from the observed cookie. If X_i and $X_{i'}$ have the same observed cookie, we set $y_{i,i'} = 1$, while $y_{i,i'} = 0$ otherwise. The final training set $\mathcal{X}_{train}$ is obtained by undersampling from $\mathcal{X}_{\text{clust}} \cup \mathcal{X}_{\text{cookie}}$.

Note that obtaining negative labeled training pairs from a point its K nearest neighbors follows the assumption that these are indeed more likely to be negatives than positives. If this assumption holds, sampling negatives from the nearest neighbors would intuitively help the classifier to learn more subtle patterns. I.e., the K nearest neighbors are close in terms of the common distance measure, but not according to the classifier.

DBSCAN* with random clusters

To benchmark the clustering approaches just discussed, we consider the following benchmark. We first cluster the sessions using the ordinary DBSCAN* algorithm, in which way we obtain initial clusters $\mathcal{B}_1^0, \ldots, \mathcal{B}_m^0$. Next, for each cluster $\mathcal{B}_j^h$ ($h \in \mathbb{N}$, with initially $h = 0$), if $|\mathcal{B}_j^h| > \beta$, we iteratively select $\min\{s_{j,h}, |\mathcal{B}_j^h|, \beta\}$ points uniformly at random from $\mathcal{B}_j$ to form a new cluster $\widetilde{\mathcal{B}}$, and update $\mathcal{B}_j^{h+1} \leftarrow \mathcal{B}_j^h \setminus \widetilde{\mathcal{B}}$. Here, $s_{j,h}$ is drawn from a geometric distribution with $p = 0.5$. This process continues until for all $j \in \{1, \ldots, m\}$: $|\mathcal{B}_j^h| \leq \beta$ for some h, at which the remaining points in $\mathcal{B}_j^h$ are labeled as one cluster.

Intuitively, we selected this benchmark as it captures the higher level hierarchy clustering of DBSCAN*, but not the low level clusters (as these clusters are picked at random). Therefore, comparing the previous methods with this random clustering approach allows us to assess whether the smaller size clusters reveal more information than the larger ones.

4.3.3 Experimental setup

Simulation parameters

Our experimental design consist of two steps. First, we consider a simulation base case on which we evaluate the clustering approaches discussed in Section 4.3.2. In this base case, users' first query arrival follows a Poisson process with rate $\gamma = 0.2$ (minutes), after which subsequent behavior over time of a particular user is modeled according to F^{abs}, F^{cookie}, F^{user}, F^{device}, P, and $\boldsymbol{\pi}$, of which the parameters were already given in Section 4.3.1. We used $U = 20,000$ users with $U_{\text{warm-up}} = 2,000$ (10%). Furthermore, we removed the first 250 sessions (not part of the first $U_{\text{warm-up}}$ users, who were only used to estimate the overall item popularity), as these would likely all be first sessions from new arriving users, and therefore including them may lead to a bias in the data. Likewise, we removed all observations after 43,200 minutes (30 days) to avoid the opposite bias: not having any new users. Users could pick from $V = 100$ items, and we choose as maximum list size $T = 10$.

For parameters that could not be adopted from the literature, we tried several parameter values and looked at three characteristics. First, we considered whether the click probability is decreasing in list position. Second, whether the attraction/satisfaction is centered around 0.5, with a standard deviation of approximately 0.1 to 0.2. Third, whether all sessions are somewhat spread out over time. This lead us to choosing users' preference for nearby items $q = 1$, user lifetime phases geometric parameter $\rho = 0.5$, and salience parameters $\nu^{(R)} = \nu^{(S)} = 5$. Figure 4.1 shows the three base case characteristics for the resulting simulated base case used in further inference. In the second step of the experimental design, we made adjustments to the latter parameters, that is, those not adapted from the literature. These adjustments will be discussed in Section 4.4.2.

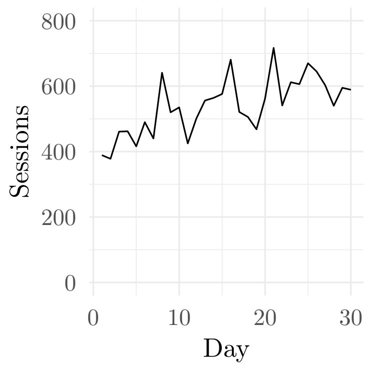
Features and MS-(H)DBSCAN* hyper-parameter settings

The simulated dataset was split into a training and test set according to a 70/30 split over the users. I.e., users always are entirely in the training set, or entirely in the test set. For each session, we used the session's start time, observed session count (as observed by the cookie), number of clicks, and whether the session's SERP has at least one click as features. Furthermore, to obtain a vector representation of the items and interactions with the SERP, we first computed a bin-count table. This table contains per item the total number of clicks, skips (no click), and the log-odds ratio between clicks and skips over 30 percent of all sessions, which combined were used as item vector representations.

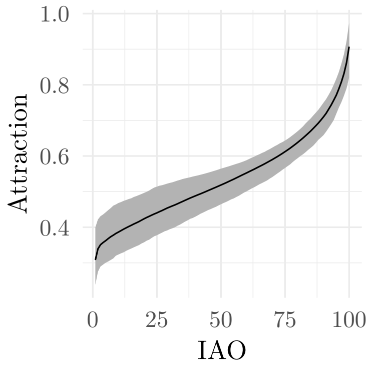
Next, for each session i, we concatenated all item vectors $\boldsymbol{\psi}_v$, $v \in \mathcal{S}_i$, in order of their position, resulting in some vector $\boldsymbol{a}_i$ with $3T$ elements. Additionally, we created four more session vectors. The first of these vectors is obtained by multiplying $\boldsymbol{a}_i$ with a vector containing ones at those positions where a click occurred, whereas for the second vector, $\boldsymbol{a}_i$ is multiplied with a vector containing ones at positions where the item was skipped (=not clicked). The third vector is obtained by multiplying $\boldsymbol{a}_i$ with a vector containing ones at the last clicked position. To obtain the fourth vector, $\boldsymbol{a}_i$ is multiplied with a vector of list positions for each item. In all cases, the vector multiplication is element-wise. Next, all five session vectors were concatenated to obtain one session vector representation.

The resulting concatenated session vector was further treated by computing all second order polynomial features, after which we normalized and applied the Yeo-Johnson [242] power scaler to make the distribution of each feature more Gaussian-like. We reduced the vector's dimension using a principle component analysis using seven principle components, the latter was chosen using the elbow method.

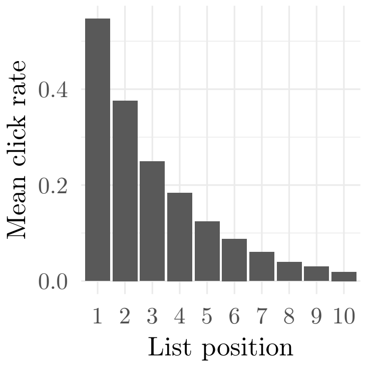
For each method, we experimented with $k \in \{1, 3, 5\}$ (here k as in κ_k, the distance to the k-th nearest neighbor). For DBSCAN*-like algorithms, we



(a) Sessions per day



(b) Mean attraction and the area between 0.05 and 0.95 quantile over the users' item attraction order (IAO)



(c) Mean click rate per list position

Figure 4.1: Summary of base case simulation

tried $\epsilon \in \left\{ \left(q_{\max} \left(q_{\min}/q_{\max} \right)^{\ell/N} \right) | \ell \in \{1, \ldots, N\} \right\}$, with $N = 9$ and $q_{\min}$, $q_{\max}$ the minimum and maximum Euclidean distance, obtained by computing all pair-wise distances over 1,000 sampled session vectors. For HDBSCAN*-type algorithms, we set minimum cluster size $M = 2$.

For re-evaluation models, we took the approach already explained in Section 4.3.2. To train classifier $\hat{f}(\mathbf{a}_i, \mathbf{a}_{i'})$, we first run MS-DBSCAN* with the best found values for k and ϵ from earlier validation of MS-DBSCAN* on the training set to, together with the cookie clusters, obtain $\mathcal{X}_{\text{train}}$. Next, we computed the Manhattan, Euclidean, and infinity norm between $\mathbf{a}_i$ and $\mathbf{a}_{i'}$, $(i, i^{'}) \in \mathcal{X}_{\text{train}}$ that were used as feature vector to train a logistic regression model. We selected for each point the $K = 1,000$ nearest neighbors to evaluate classifier $\hat{f}$ on. All non-evaluated pairs received distance $\delta_{\max} = -\log\left(10^{-6}\right)$. Next, the MS-(H)DBSCAN algorithms were evaluated using the new distance matrix W, where we experimented again with $k \in \{1, 3, 5\}$, and $\epsilon \in \left\{ q_{\min} + \frac{\ell(q_{\max} - q_{\min})}{N_{\text{re-eval}}} | \ell \in \{1, \ldots, N_{\text{re-eval}}\}, N_{\text{re-eval}} = 5 \right\}$.

All algorithms excluding HDBSCAN* (i.e., including DBSCAN*) were trained using the `sklearn` package in Python [187] (version 0.22.1). `sklearn` was also used to compute error scores (see Section 4.3.3). We used the `hdbscan` package [172] (version 0.8.26) to obtain the dendogram and condensed tree, based on which we could impose the maximum cluster size in the way described in Section 4.3.2. For both packages, the default parameters were used unless indicated otherwise.

Error metrics

We considered error metrics from two perspectives. First, we consider error measures with respect to overall website performance. More precisely, given some final clustering $\{\mathcal{B}_1^{\text{final}}, \ldots, \mathcal{B}_m^{\text{final}}\}$, the following error measures are computed. 1) We compute the APE (absolute percentage error) between the real and estimated number of unique users (the latter being equal to m), 2) the Kullback-Leibler divergence (KL-divergence) between the real and estimated user session count distribution (the latter being equal to the cluster size distribution), and 3) the KL-divergence between the real and estimated user conversion distribution. Here, user conversion is defined as the fraction of items clicked per user over all shown (but not necessarily evaluated) items.

The second perspective is on the level of the clusters themselves, where we consider two error measures. To determine the quality of the clusters, we computed the adjusted Rand index (ARI) [108] between computed and real session clusters. Besides ARI, we also measure how well the model distinguishes whether each new session originates from an existing or already observed user, which is measured using the accuracy score.

Since ARI measures the overlap between the computed and real session clusters, we consider ARI to be our main error measure, using the other error measures to study possible side-effects when optimizing for ARI.

4.4 Results

4.4.1 Results on base simulation case

Table 4.2 shows how the different models perform in terms of several error measures on both the training and test set. For each method, the shown results are the best results obtained under the different hyper parameters tried for that method under that dataset. I.e., in theory the hyper parameters might be slightly different between training and test, though in practice we found this was rarely the case.

The *OBS* model in the table are the scores one would obtain if the observed cookies would be used as clusters. Models using the classifier as distance measure are indicated using subscript p. What immediately becomes apparent is that compared to these observed cookie clusters, all methods perform considerably worse. Hence, in the scenario we consider: a single query where the true location $\left(\eta_1^{(u)}, \eta_2^{(u)}\right)$ is only revealed by clicked and skipped item locations, our approaches do not come near what one would obtain if one would simply take the observed cookies.

However, the scores do reveal some interesting patterns. First, approaches using a probabilistic distance measure seem to overfit the data: they perform relatively well (compared to the other approaches) on various measures on the training set, but on the test set these results are mitigated. Here, MS-DBSCAN* seems to work best when considering multiple error measures. Looking at the results from different hyper-parameter settings for MS-DBSCAN* (Table 4.3), we observe that selecting $k = 1$ performed best. Furthermore, due to our maximum size constraint the clusters did not alter for $\ell \geq 4$ ($\epsilon \geq 6.33$).

Furthermore, methods without a probabilistic distance measure do outperform the DBSCAN*-RAND method on most measures. I.e., they perform better at picking sessions originating from the same user from a given cluster

$\mathcal{B}_j$ produced by DBSCAN*, than if we would pick session pairs at random. Although it is difficult to draw a firm conclusion, these findings might be an indication that the same user signal we try to infer from the click data is somewhat weak: if our methods would not pick up a signal at all, we would expect them to have the same result as the DBSCAN*-RAND method.

Table 4.2: Results on the base case

Model	Dataset	ARI	KL-div. session count	KL-div conversion	APE unique user	New user accuracy
MS-DBSCAN*	train	0.0012	**0.55**	0.13	15	**0.56**
MS-DBSCAN*$_p$	train	**0.14**	0.74	**0.092**	**77**	0.5
DBSCAN*-RAND	train	0.0002	1	0.096	**0.011**	0.42
MS-HDBSCAN*$^+$	train	0.0007	0.75	0.15	10	0.52
MS-HDBSCAN*$^-$	train	0.0007	0.75	0.15	10	0.52
MS-HDBSCAN*^+_p	train	0.092	0.9	0.11	**0.011**	0.46
MS-HDBSCAN*^-_p	train	0.1	0.9	0.11	**0.011**	0.46
OBS	*train*	*0.91*	*0.017*	*0.0032*	*15*	*0.95*
MS-DBSCAN*	test	**0.0022**	**0.11**	**0.0026**	60	**0.56**
MS-DBSCAN*$_p$	test	0.0015	1.4	0.13	**6.8**	0.4
DBSCAN*-RAND	test	0.0004	0.32	0.015	40	0.5
MS-HDBSCAN*$^+$	test	0.002	0.16	0.0042	53	0.55
MS-HDBSCAN*$^-$	test	0.002	0.16	0.0042	53	0.55
MS-HDBSCAN*^+_p	test	0.0015	1.4	0.13	7.2	0.4
MS-HDBSCAN*^-_p	test	0.0015	1.4	0.13	7.2	0.4
OBS	*test*	*0.91*	*0.1*	*0.0076*	*51*	*0.95*

Table 4.3: ARI of MS-DBSCAN* on the training set of the base case

		k		
ℓ	ϵ	1	3	5
1	0.013	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
2	3.44	0.0008	0.0004	0.0001
3	4.84	0.0011	0.0005	0.0004
4	6.33	0.0013	0.0006	0.0004
5	8.20	0.0013	0.0006	0.0004
6	10.76	0.0013	0.0006	0.0004
7	14.57	0.0013	0.0006	0.0004
8	20.94	0.0013	0.0006	0.0004
9	30.45	0.0013	0.0006	0.0004

Table 4.4: Simulation scenarios

Variable	**Values**
User distance sensitivity (USER_DIST_SENS_[q])	$\{1, 2, 5, 10, 25, 50\}$
Number of items (ITEM_COUNT_[V])	$\{10, 100\}$
Lifetime phases (LIFETIME_PHASES_[ρ])	$\{.15, .29, .43, .5, .57, .71, .85\}$
Salience (SALIENCE_[ϕ]_[ϕ'])	$\in \{1, 2, 5, 10\}^2$

4.4.2 Results on multiple simulation scenarios

In order to judge the sensitivity of our findings on the parameter settings of the simulation model, we permuted the simulation settings to see if this would alter our results. The different settings are indicated in Table 4.4. Whenever one parameter was permuted, the rest of the parameters remained fixed at their base case value.

As re-running all models on all simulation settings would be computationally rather expensive, we only re-evaluated the best performing models on the simulation cases. Since in our base case we found that the parameters $k = 1$, $\epsilon = \left(q_{\max} \left(q_{\min}/q_{\max}\right)^{2/3}\right)$ worked reasonably well, these parameters were used for MS-DBSCAN* and DBSCAN*-RAND. The maximum cluster size remained the same as in the base case.

Figure 4.2 shows how the models perform over the different simulation settings in terms of ARI, which is our main response variable of interest. The figure suggests that all cluster models do stochastically dominate DBSCAN*-RAND. Furthermore, MS-DBSCAN* seems to outperform the other clustering methods in terms of ARI. As assumptions like homogeneity of variance or normality do not hold in this case, we used a Kruskall-Wallis test, which rejects in this case that all median ARI scores over the different methods are the same (using significance level $\alpha = .01$, $p < 10^{-4}$). Pairwise (between MS-DBSCAN* and all other methods) one-sided pairwise Wilcoxon signed rank tests also indicate MS-DBSCAN* performed significantly better than the other methods (all p-values are smaller than 10^{-4}).

Table 4.7 shows how MS-DBSCAN* performs on the various simulation cases.

The rows in boldface have ARI $\geq$ 0.0025. The results suggest that when strengthening the signal, that is increasing click probabilities, leads to some improvement in ARI. The most obvious way to do so is by decreasing the number of items (which, as we use bin counting, ensures each item has sufficient data for bin counting). However, these improvements remain small.

Table 4.5 shows how the different error measures correlate, using the error scores from all clustering algorithms on the various simulation cases. ARI seems to be weakly correlated with most other error measures, with the sign being in the desired direction (i.e., decrease in KL-divergence for both session count and conversion, but an increase in the new user accuracy). However, both ARI and the new user accuracy show a positive correlation with the percentage error in the number of unique users.

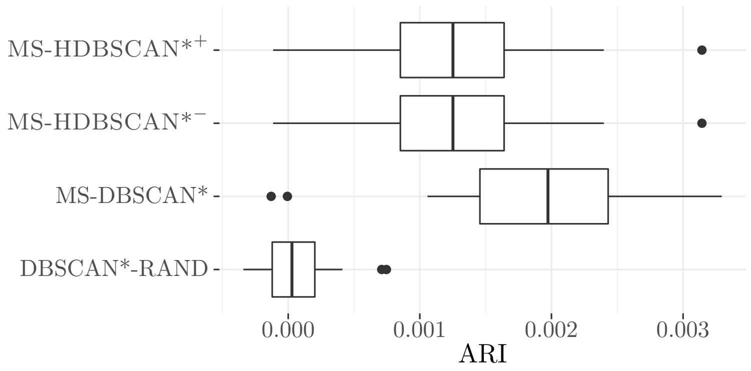


Figure 4.2: Scores over all simulations

4.5 Conclusion and discussion

In this chapter, we presented a homogeneous query click simulation model, and illustrated its usage to the problem of uncovering users from their web sessions. The simulation model is composed of several models from which previous literature suggests that these models work well in explaining typical patterns observed in click data, while remaining relatively simple. Such patterns include the position bias, cookie censoring, and users' utility over multiple products. Furthermore, we illustrated the simulation model on the problem of (partially observed) session clustering, that is, identifying unique

Table 4.5: Correlation matrix error measures

	ARI	KL-div. conversion	KL-div session count	APE unique user	New user accuracy
ARI	1.00				
KL-div. conversion	-0.15	1.00			
KL-div. session count	-0.16	0.60	1.00		
APE unique user	**0.38**	-0.52	-0.92	1.00	
New user accuracy	0.39	-0.59	-0.85	**0.95**	1.00

users from their query-sessions. To solve the latter problem, we tested several mutations of (H)DBSCAN*, where these mutations differ from HDBSCAN* or DBSCAN* as they allow for incorporating a maximum cluster size. Furthermore, we consider both a Euclidean and probabilistic distance measure to determine whether a pair of sessions originated from the same user. The probabilistic distance measure was obtained using a pre-trained classification model.

Given a simulated dataset, we considered solving the problem of uncovering users from their web sessions by using (H)DSCAN*-type clustering algorithms. Comparing (H)DSCAN*-type algorithms with clusters one would obtain by using cookies, we found the accuracy of using cookies largely outperformed that of not using or partially using cookie data. This considerable difference seems to be due to two reasons. 1) The simulated censored cookies turned out to be rather accurate, implying that, assuming the parameters used for cookie censoring adapted from previous literature are accurate, censoring in cookie data does not impose that much of a problem in accurately measuring the metrics studied in this chapter. These metrics being the number of unique users, user sessions count distribution, user conversion distribution, the quality of session clusters (in terms of adjusted Rand index (ARI)), and estimating whether the next session originates from a new or existing user. 2) As we only consider a homogeneous query, the users' preferences are only revealed from the items users clicked, a signal the various (H)DBSCAN*-type algorithms find difficult to detect. Strengthening this signal, e.g., by increasing the number of clicks, leads to a small but significant improvement in ARI.

Other interesting observations include the difference between using Euclidean distance and a probability distance measure in the (H)DBSCAN*-type algorithms, the latter being obtained from training a classifier on detecting whether session pairs originate from the same user. The results show that the probabilistic classifier tends to overfit. Where some methods using probabilistic distance measures performed quite reasonably on the training set, they were outperformed by methods using Euclidean distance on the test set.

By studying the correlations between the various error metrics considered in this chapter, we observe that some error measures show contradictory correlations. In particular, the positive correlation between cluster ARI and average percentage error in the number of unique users (.38), and between the accuracy in estimating whether the next session originates from a new user and the new user average percentage error (.95), indicate that optimizing for one of these error measures may lead to decreased performance in the other.

Although our findings suggest that the practicality of session clustering from single query click data is limited, the usage of the simulation model did allow for studying the sensitivity of the clustering algorithms on different click behavior, something that would not easily have been possible with real click data. It also allowed us to study the effects of user censoring caused by cookie churn or the usage of multiple devices. This showed that if we adopt models for cookie churn behavior found in the literature, this censoring only has a small effect on the accuracy of the website metrics discussed in this chapter, with an exception for estimating the number of unique users.

Given our findings, a number of questions remain. First, it would be interesting to extend the simulation model to allow for multiple queries. As the solutions to the (multi-query) CIKM 2016 and ICDM 2015 cross-device matching competitions were quite successful, a logical hypothesis would be that incorporating multiple queries into the simulation model would improve the results obtained from (H)DBSCAN*-type algorithms. Apart from multiple queries, it could also be interesting to consider alternative models for representing user-item relevance, as the Fleder-Hosanagar model . I.e., modeling users and items using On the other hand, more diversity also causes clicks to be more spread across items, which as we have seen, may lead to decreasing clustering performance.

Second, in this study, we only used a logistic regression model to approximate the probability of two sessions originating from the same user. Given the limited success of this approach so far, it would be interesting to consider other approaches. As the limited results seem to be due to overfitting, including regularization or using bagging could lead to better results.

Third, there is still limited knowledge on how cookie censoring occurs. Currently, multiple models exist in the literature, but most models only consider a specific type of censoring (e.g., only censoring by cross-device usage or cookie churn), from which one cannot infer how these different types of censoring interact. Also, as discussed in Section 4.3.1, literature providing parametric models for cookie churn, user lifetime and absence time (the time between two sessions) seems to be contradictory in terms of tail probabilities. Hence, click simulation models that incorporate cookie censoring would benefit from studies taking a more holistic view on cookie censoring.

4.6 Simulation procedure

Table 4.6: List of notation

Variable	**Interpretation**
$\{1, \ldots, n\}$	Set of query-sessions, indexed by i.
$\mathcal{S}_i = \{1, \ldots, T\}$	Set of items in SERP of session i, indexed by t.
$\mathcal{V} = \{1, \ldots, V\}$	Set of all items, indexed by v.
$\mathcal{U} = \{1, \ldots, U\}$	Set of all users, indexed by u.
$r_i(t)$	Item $v \in \mathcal{V}$ at position t in the SERP of query-session i.
$r_i^{-1}(v)$	Position of item v in the SERP of query-session i, zero if $v \notin \mathcal{S}_i$.
$r_i^{\max}$	Largest position of a clicked item $v \in \mathcal{S}_i$, zero if no items were clicked.
$R_v^{(i)}$; $\phi_{u,v}^{(R)}$	Attraction of user i for item v, with $\mathbb{P}(R_v^{(i)} = 1) = \phi_{u,v}^{(R)}$, given $v \in \mathcal{S}_i$.
$S_v^{(i)}$; $\phi_{u,v}^{(S)}$	Satisfaction of user i for item v, with $\mathbb{P}(S_v^{(i)} = 1) = \phi_{u,v}^{(S)}$, given $v \in \mathcal{S}_i$.
$E_t^{(i)}$	Whether item at position t in SERP i was evaluated.
$y_t^{(i)}$	Whether item at position t in SERP i was clicked.
$\boldsymbol{\eta}_u$	Vector denoting the position of an user u in the user-item space.
$\boldsymbol{\psi}_v$	Vector denoting the position of an item v in the user-item space.

Table 4.6 – *Continued*

Variable	**Interpretation**
$\nu^{(R)}, \nu^{(S)}$	Salience parameters for attraction and satisfaction.
q	Users' preference for nearby items.
$\omega_{u,v}$	Distance between user u and item v in the user-item space.
$\hat{\phi}_v$	Overall estimated popularity of item $v \in \mathcal{V}$.
F^{cookie}	Cookie lifetime distribution (hyper-exponential) with parameters $\boldsymbol{\lambda}$ and $\mathbf{p}$.
$\mathcal{T}^{\text{cookie}}_{u,o,d} \sim F^{\text{cookie}}$	Random variable denoting the cookie lifetime for the o-th cookie of user u on device d.
F^{abs}	User absence distribution (Pareto-I) with parameters α (shape) and m (scale).
$\mathcal{T}^{\text{abs}}_{i,u} \sim F^{\text{abs}}$	Random variable denoting the time between the i-th and $i+1$-th session of user u.
F^{user}	User lifetime distribution (sum of N_u hyper-exponentials) with parameters $\boldsymbol{\lambda}$, $\mathbf{p}$ and ρ (geometric parameter for N_u).
$\mathcal{T}^{\text{user}}_{u} \sim F^{\text{user}}$	Random variable denoting the user lifetime of user u.
P, $\boldsymbol{\pi}$	Device transition matrix and initial device probabilities.

Algorithm 4: Simulate_clicks

```
Simulate_clicks ($\mathcal{U}$, $\mathcal{V}$, $\hat{\boldsymbol{\phi}}$, $\boldsymbol{\pi}$, $P$, $\mathbf{p}$, $\boldsymbol{\lambda}$, $\rho$, $\boldsymbol{\phi}^{(R)}$, $\boldsymbol{\phi}^{(S)}$)
    inputs : Users $\mathcal{U}$, items $\mathcal{V}$, item popularity estimates $\hat{\boldsymbol{\phi}}$, device probabilities $\boldsymbol{\pi}$
             and $P$, $F^{\text{cookie}}$ parameters $\mathbf{p}$ and $\boldsymbol{\lambda}$, user lifetime geometric phases
             parameter $\rho$, attraction and satisfaction parameters $\boldsymbol{\phi}^{(R)}$ and $\boldsymbol{\phi}^{(S)}$
    output: Simulation realization $\mathcal{D}$

    for $u \in \mathcal{U}$ do
        /* Draw initial device and cookie lifetime, and draw the user's
           lifetime                                                     */
        $D \leftarrow \text{dic}()$;   $i \leftarrow 1$;   $o \leftarrow 1$;   $t \leftarrow 0$;
        Draw device $d$ from Multinom($\boldsymbol{\pi}$); $D[d] \leftarrow o$;
        Draw $\mathcal{T}^{\text{cookie}}_{u,o,d}$ from HyperExp($\boldsymbol{\lambda}, \boldsymbol{p}$); $\mathcal{T}^{\text{user}}_{u}$ from RepHyperExp($\rho, \boldsymbol{\lambda}, \boldsymbol{p}$);

        /* Simulate new query-sessions while the user's lifetime has not
           elapsed                                                      */
        while $t \leq \mathcal{T}^{user}_{u}$ do
            $\mathcal{D} \leftarrow$ Get_session_clicks($v, i, \mathcal{V}, \boldsymbol{\phi}^{(R)}, \boldsymbol{\phi}^{(S)}$);
            /* 2) Draw the time until the next session and update $t$
               accordingly                                              */
            Draw $\mathcal{T}^{\text{abs}}_{i,u}$ from Pareto-I($m, \alpha$);
            $t \leftarrow t + \mathcal{T}^{\text{abs}}_{i,u}$; $i \leftarrow i + 1$;
            /* 3) Update the device for the next session                */
            Draw $d'$ from MultiNom($I_d P$);
            if $d' \neq d$ then
                if not $D.exists(d)$ then
                    $o \leftarrow o + 1$;
                    Draw $\mathcal{T}^{\text{cookie}}_{u,o,d}$ from HyperExp($\boldsymbol{\lambda}, \boldsymbol{p}$);
                    $\mathcal{T}^{\text{cookie}}_{u,o,d} \leftarrow \mathcal{T}^{\text{cookie}}_{u,o,d} + t$;
                else
                    $o \leftarrow D[d']$;
                $d \leftarrow d'$;
            /* 4) Simulate cookie churn                                 */
            if $t > \mathcal{T}^{cookie}_{u,o,d}$ then
                $o \leftarrow o + 1$;
                Draw $\mathcal{T}^{\text{cookie}}_{u,o,d}$ from HyperExp($\boldsymbol{\lambda}, \boldsymbol{p}$);
                $\mathcal{T}^{\text{cookie}}_{u,o,d} \leftarrow \mathcal{T}^{\text{cookie}}_{u,o,d} + t$;
                $D[d] \leftarrow o$;
    return $\mathcal{D}$
```

Algorithm 5: GET_SESSION_CLICKS

```
1 Get_session_clicks ($v$, $i$, $\mathcal{V}$, $\boldsymbol{\phi}^R$, $\boldsymbol{\phi}^{(S)}$, $\mathcal{D}$)
      inputs : User $v$, session $i$, attraction and satisfaction parameters $\boldsymbol{\phi}^{(R)}$ and
               $\boldsymbol{\phi}^{(S)}$, simulation realization $\mathcal{D}$
      output: Simulation realization $\mathcal{D}$
2     Draw $\mathcal{S}_i$ in its respective order by repetitively drawing from
        $\text{MULTINOM}(\hat{\phi}_v / \sum_{v' \in \mathcal{V}} \hat{\phi}_{v'}; v \in \mathcal{V} \setminus \mathcal{S}_i)$;
3     Draw $R_v^{(i)}$, $S_v^{(i)}$ from $\text{BERNOULLI}(\phi_{u,v}^{(R)})$ and $\text{BERNOULLI}(\phi_{u,v}^{(S)})$ resp. for all
        $v \in \mathcal{S}_i$;
4     Compute $E_t^{(i)}, y_t^{(i)}$, and recompute $S_v^{(i)}$ according to Equations (4.1) to (4.6);
5     Append $(i, u, o, \mathcal{S}_i, \mathbf{y}_i)$ to $\mathcal{D}$;
6     return $\mathcal{D}$
```

4.7 Results on multiple simulations

Table 4.7: Results MS-DBSCAN* on other simulation cases

Simulation case	ARI	KL-div. conversion	KL-div. session count	APE unique user	New user accuracy
base_case	0.0021	0.0044	0.095	62	0.53
item_count_10	**0.0025**	**0.0006**	**0.049**	**67**	**0.57**
item_count_100	0.0015	0.0053	0.098	59	0.54
lifetime_phases_.15	0.0014	0.014	0.14	56	0.56
lifetime_phases_.29	0.0016	0.0076	0.1	59	0.55
lifetime_phases_.43	0.0021	0.008	0.11	58	0.56
lifetime_phases_.5	0.0021	0.0044	0.095	62	0.53
lifetime_phases_.57	0.0019	0.0049	0.11	61	0.55
lifetime_phases_.71	0.0019	0.0054	0.084	60	0.56
lifetime_phases_.85	**0.0028**	**0.005**	**0.092**	**61**	**0.53**
salience_1_1	0.0012	0.0018	0.07	64	0.58
salience_1_2	0.0022	0.0019	0.059	65	0.57
salience_1_5	0.0014	0.0015	0.085	62	0.59
salience_1_10	**0.0026**	$\mathbf{< 10^{-4}}$	**0.084**	**62**	**0.58**
salience_2_1	0.0011	0.0026	0.15	53	0.55
salience_2_2	0.002	0.0033	0.19	51	0.55
salience_2_5	**0.0027**	**0.0005**	**0.12**	**56**	**0.57**
salience_2_10	0.0018	$< 10^{-4}$	0.094	60	0.58
salience_5_1	$< 10^{-4}$	0.011	0.25	44	0.52
salience_5_2	$< 10^{-4}$	0.021	0.2	51	0.53
salience_5_5	0.0021	0.0044	0.095	62	0.53
salience_5_10	0.002	$< 10^{-4}$	0.079	62	0.57
salience_10_1	0.0016	0.0049	0.051	64	0.57
salience_10_2	0.0014	0.0034	0.065	66	0.57
salience_10_5	0.0018	0.0045	0.1	60	0.56
salience_10_10	0.0012	0.0001	0.042	71	0.63
user_dist_sense_1	0.0021	0.0044	0.095	62	0.53
user_dist_sens_2	**0.0029**	**0.012**	**0.17**	**49**	**0.54**
user_dist_sens_5	**0.0033**	**0.0092**	**0.15**	**49**	**0.53**
user_dist_sens_10	**0.0026**	**0.002**	**0.12**	**57**	**0.56**
user_dist_sens_25	0.0024	$< 10^{-4}$	0.13	59	0.57
user_dist_sens_50	**0.0032**	**0.0002**	**0.09**	**64**	**0.56**

Chapter 5

Predicting Tenure from Unstructured Resumes

This chapter explores to what extent job seekers' future job tenures can be predicted using only the information contained in their own résumés. Here, job tenure is interpreted as the time spent in a single job occupation. To do so, we compare the performance of several machine-learned survival models in terms of multiple error measures, including the Brier score and the C-index. The results suggest that ensemble methods, such as random survival forest and Cox boosting, work well for this purpose. We further find that in particular time-related features, such as the time a person has already worked in a particular field, are predictive when predicting the person's future tenure. However, the results also show that this prediction task is difficult. There is substantial subjectivity in both how job seekers define their jobs, and at what level of granularity they indicate their job tenures. As a result, the best performing models (survival ensemble methods) only perform marginally better than the used benchmark (a Kaplan-Meier estimate).

5.1 Introduction

Given the high internet penetration of job seekers, one could expect it to become easier for recruiters to find and select potential candidates. The reality, however, sometimes turns out to be different. Early studies on online recruitment reported profitable benefits for recruiters, including an increased speed of hiring, or an improved quality applicants. However, they also reported the problem of having to sift through a sometimes overwhelming number of candidates [216].

Apart from this challenge of information overload, the mere digitization of vacancies and résumés also seems to have a substantial impact on how recruiters search and select candidates. A study from the Ladders, a large résumé database provider, shows that recruiters seem to only “read" a résumé for as brief as 7.4 seconds on average [220]. This would be in agreement with studies showing how, in general, digital documents are more scanned than read [202]. Although such a brief résumé scan may be a necessity in dealing with a large number of candidates, at the same time, recruiters are likely to neglect useful information from the résumé.

Many of the methods proposed in the literature that assist in matching job seekers and vacancies online use the semantic overlap between the résumé and the vacancy as a proxy for the quality of this match [84]. This, however, neglects other types of information contained in résumés that could provide information about the quality of the match. In this paper, we will instead use the temporal data often contained in résumés. Most job seekers indicate their job history in their résumé, in which their previous occupations are listed, along with a start and end date for each job. Our aim is to predict job tenures, defined as the time difference between these start and end dates, using other data that is contained in the résumé. This data includes features such as the type of job, education history, and the number of years of experience.

Predicting future job tenure from résumés is not a new problem. In fact, it has been the subject of many studies in personnel psychology in the last few decades [36, Ch. 12]. Many of these studies use Cox regression without (or with small order) interaction terms [105]. This simplifies the interpretation of cause-effect in these models. The problem we consider, however, differs from these studies in three ways. 1) We automatize the processes of extracting features from the résumé, both using a pre-trained résumé parser, and by using word2vec. 2) Job tenure is determined through the résumé, rather than by extracting this data from an HRIS. 2) We focus on methods that emphasize on making accurate predictions, mostly by incorporating a large number of second or larger order interaction terms, rather than models that emphasize on explanations.

This chapter has the following structure. Section 5.2 discusses related work. Section 5.3 introduces the survival models used in this study, and discusses how to measure the error of these models. Section 5.4 discusses properties of the résumé dataset, with a focus on properties of such unstructured datasets that may lead to biased results. It further considers how to avoid such bias. Section 5.5 presents the outcomes of the survival model comparison from different perspectives. Section 5.6 draws a final conclusion and provides directions for further research.

5.2 Related work

The usage of biographic data (such as a résumé) for the purpose of personnel selection has been the subject of many studies in personnel psychology, for one part as it remains one of the most common procedures in personnel selection [54]. And perhaps not without reason, as various meta-studies on the validity of using biographic data when predicting job performance show r^2-values between .33 and .37 [36, p. 261].

With the digitization of these biographic sources, predicting future job performance based on this data comes with both opportunities and challenges. The development of new state of the art data mining techniques and the increasing volume and diversity of stored data related to candidates, jobs, and the labor market, allows for revealing new patterns that were not possible to analyze before [214]. On the other hand, the lack of explainability of models trained by machine learning algorithms and their possible consequences should not be taken lightly. One can easily imagine machine learning algorithms picking-up patterns in the data that are unethical to act upon [151].

Machine learning techniques have been commonly applied to résumés or other biographic sources for various applications. As unstructured résumés consist for the most part out of textual data, it should not be a surprise that most literature focuses on applying Natural Language Processing (NLP) techniques. One common application is information extraction from résumés, which can be used to assist a recruiter while searching through the vast quantity of résumés in a résumé database (e.g., [244, 133]). Or job recommendation, in which the semantic overlap between the résumé and vacancy can be used as a proxy to evaluate the quality of the match [84]. In both applications, missing data is a common issue [118].

Few studies consider the sequential and time-based elements in a résumé, which in particular presents itself in the job history section. In résumés, it is common to write down one's previous jobs in chronological order and indicate the start and end date of each job. This information could be used to infer a job seeker's most likely next job, given a sequence of previous jobs. For this problem, several models have been suggested in the literature [185, 118, 145, 147]. Li et al. [147] compared several sequential models for this task, where in particular a Long Short-Term Memory (LSTM) recurrent neural network [104] with additional contextual data performed well.

From the start and end dates, one could infer how long job seekers will be likely to remain in their jobs. For this problem, survival analysis has been a frequently used method in personnel psychology, in particular in the form of Cox regression [105]. The fact that survival models usually allow for censored data makes these models attractive for studying turnover. Participants of the study might still be "alive", or in other words still occupying the job under study, before the end of the study. Since in such cases the full tenure is not observed, these observations are right-censored.

Even though there has been a substantial increase in the number of studies applying machine learning methods in the field of HR, and in particular for the application of predicting employee turnover [214], only a few consider combining machine learning methods with survival analysis. Wang et al. [230] propose a survival model that is fitted using a Bayesian model. The Bayesian model was chosen to cope with the high dispersion in the number of observations for a job transition from some job a to job b. I.e., most transitions have little to no observations, whereas some self-transitions may be very frequent. The authors show that if one is indifferent about to which job the job seeker switches, and only considers how long the job was occupied, the Bayesian model outperforms a model ignoring covariates in terms of perplexity. However, the difference between the with/without covariates models evaporates when considering more passive job seekers.

Li et al. [145] discretize time and predict a value proportional to the survival function using a squared loss function. As the predicted values are only proportional to the probability of remaining in a job, the study considers the correct order of turnover events, rather than predicting tenure. The method outperformed typical parametric or semi-parametric survival methods such as a Cox regression or the log-logistic model.

5.3 Methods

5.3.1 Job transitions modeled as a survival model

Before we discuss the machine-learned survival methods, we will first introduce some notation. This notation is also summarized in Table 5.1. Consider we have job seekers $j = 1, \dots, J$; each having a job history indexed by $h = 1, \dots, n_j$; where n_j is the job that job seeker j occupies at the time of uploading his/her résumé to the résumé database. I.e., it is the number of jobs held by job seeker j at the time of observing his/her résumé. Observe that in modeling job transitions, we assume jobs are non-overlapping.

Let $T_{j,h} \in \mathbb{R}^+$ be the observed elapsed time between the start of job h, and the time at which the job seeker switches to job $h+1$, which we will also refer to as the *tenure* of the h-th job of job seeker j. The "observed" part is crucial here: in reality, the time between the start of job h and the end of job $h+1$ may be partially censored, which is not reflected in $T_{j,h}$. Since in survival analysis $T_{j,h}$ is more commonly referred to as the *failure time*, we will use both the terms tenure and failure time somewhat interchangeably in this paper. Though, both have the same interpretation.

Likewise, let $\tilde{T}_{j,h} \in \mathbb{R}^+$ be the uncensored tenure. That is, if observation (j,h) is censored, $\tilde{T}_{j,h}$ is the tenure that would have been observed if (j,h) had not been censored. If no censoring occurred, we have $\tilde{T}_{j,h} = T_{j,h}$. We assume $\tilde{T}_{j,h}$ to be a random variable, and since we consider the problem from a survival analysis perspective, our interest lies mostly in estimating the survival function

$$S_{j,h}(t) = \mathbb{P}(\tilde{T}_{j,h} > t | \mathbf{x}_{j,h}), \tag{5.1}$$

where $\mathbf{x}_{j,h}$ is the observed covariate matrix containing P elements. We assume both $\tilde{T}_{j,h}$ and $T_{j,h}$ to be independent of all previous jobs and other jobs seekers, given $\mathbf{x}_{j,h}$ (do note that $\tilde{T}_{j,h}$ and $T_{j,h}$ are dependent). Since this implies that the job seeker and job index are reflected in $\mathbf{x}_{j,h}$, we will somewhat simplify notation by also using the index $i = 1, \dots, I$ to index jobs, where each i has a one-to-one correspondence with some pair (j,h) and vise versa. I.e., the index pair (j,h) will further only be used in case we want to be specific about a job seeker, whereas index i will be used otherwise.

To estimate 5.1, we will frequently use the cumulative hazard function

$$\Lambda_i(t) = \int_{\tau=0}^{t} \lambda_i(\tau) d\tau, \tag{5.2}$$

with

$$\lambda_i(t) = \lim_{\Delta t \to 0} \mathbb{P}(t \leq \tilde{T}_i \leq t + \Delta t | \tilde{T}_i > t, \mathbf{x}_i), \tag{5.3}$$

being the hazard rate. From the hazard rate, the survival function can be obtained via the relationship [116, p. 16].

$$S_i(t) = \exp(-\Lambda_i(t)). \tag{5.4}$$

Estimates of the survival function, or other estimates, are denoted by adding a hat (e.g., $\hat{S}_i(t)$).

Although multiple types of censoring may exists in the data, we only model right censoring explicitly. Other types of censoring are dealt with by transforming the data itself. We will return to exactly how this (transforming the data) is done in Section 5.4.2. For now, we assume only right censoring exists, which is denoted by δ_i (0 if right censoring occurred, 1 otherwise).

Last, for validation, we will discretize time into intervals $\{u_1, \dots, u_R\}$ of equal length $\rho \in \mathbb{R}$, where we truncate time at some point ρR. We do note that, since employee tenure is right-skewed and overdispersed (see Figure 5.2), we plan to truncate already for relatively small ρR (i.e., equivalent to 3-5 years). We wish to select a small ρ, such that long tenures do not have a disproportionate effect on our analysis. I.e., we choose R and ρ such that the equal time intervals do not cause sparsity problems within the bin (which would happen for large t and/or small ρ), but also do not cause the bins to be too imbalanced (which would happen for large ρ).

5.3.2 Survival estimation methods

Benchmark models

All machine-learned survival models presented in this paper are benchmarked against three benchmark methods. 1) A Kaplan-Meier (KM) estimate [116, Ch. 4], 2) Cox proportional hazard model with an elastic net penalty [85], where to estimate the baseline hazard the Brewlow estimator [152] is used. 3) A binary survival tree using the log-rank splitting rule [111].

Variable	**Interpretation**
$j = 1, \ldots, J$	Job seeker index.
$h = 1 \ldots, n_j$	Job indices for jobs occupied by job seeker j; n_j being the number of jobs held by j.
$i = 1, \ldots, I$	Job index corresponding to a unique pair (j, h).
$\mathcal{I}$	Set of observations put aside as validation set, i.e., $\mathcal{I} \subseteq \{1, \ldots, I\}$.
$r = 1, \ldots, R$; ρ	Time is discretized in bins of length ρ, with each bin being indexed by r.
$T_{j,h}$	Time between the start of the h-th job of job seeker j and the latest time at which j still occupied job h with certainty (i.e., this is either the end of the tenure, or the censoring time).
$\delta_{j,h}$	0 if the h-th job of job seeker j was right-censored, 1 otherwise (which implies no censoring occurred).
$\tilde{T}_{j,h}$	The uncensored tenure of the h-th job of job seeker j.
$\mathbf{x}_{j,h}$, P	Covariate vector for the h-th job of job seeker j containing P elements.
$S_i(t)$, $\lambda_i(t)$, $\lambda_i(t)$	$\tilde{T}_i$ is distributed according to the survival function $S_i(t)$, with $\lambda_i(t)$, and $\Lambda_i(t)$ being the hazard rate and cumulative hazard rate respectively.
$\gamma_{i,r}$	Whether T_i switched jobs during interval u_r.
$v_{i,r}$	Equals one if job i has churned before the start of time interval r.

Table 5.1: Notation used

Ensemble survival models

A common approach to improve the quality of predictions from weak learners is by using model ensembles. In this study, we consider two approaches: the random survival forest introduced by [111], and a Cox boosting approach [91].

Random survival forest [111] extends the single survival tree by aggregating multiple survival trees. Each tree is constructed from a bootstrap sample including a different set of randomly drawn covariates. Let $\hat{\Lambda}_{i,b}(t)$ be the estimated cumulative hazard rate of bootstrap sample $b \in \{1, \ldots, B\}$ for observation i at time interval t. The averaged cumulative hazard rate is obtained via

$$\hat{\Lambda}_i(t) = \frac{1}{B} \sum_{b=1}^{B} \hat{\Lambda}_{i,b}(t), \tag{5.5}$$

from which using Expression (5.4) the survival curve can be obtained.

To employ boosting, we use the boosting procedure by Friedman [86]. As the method employs Cox's partial likelihood, the method does not provide an estimate of the baseline hazard. To find the baseline hazard, the same procedure as for the Cox model was applied. That is, we use the Breslow estimator to estimate the baseline hazard, though using the output from the boosting model instead of the linear link function.

Neural survival models

Feedforward neural survival models Neural survival models have been discussed by various authors, taking different approaches. One approach is to use the Cox-PH model as an activation function, and use the log-likelihood of the Cox model as loss function [243, 126]. However, as noted by [87], the Cox-PH log-likelihood is problematic. One requires the full risk set, that is, the set of non-censored observations still at risk of failure at some time t, which is not available when naively splitting the dataset into batches. Alternatively, we may use that the log-likelihood function of discrete-time data can be reduced to that of a Bernoulli distribution [116, pp. 71-73], when only taking into account those time intervals during which the job seeker was either still employed or transitioned with certainty.

We will instead use a similar approach as Gensheimer & Narasimhan [87] to fit neural survival models. More precisely, this study models the neural survival model as a feedforward neural network, only adjusting the output layer to produce a survival curve. To rewrite the problem to discrete time, let $\gamma_{i,r} = 1$

if $T_i \in u_r, \delta_i = 1$ (zero otherwise), and $v_{i,r} = 1$ if $T_i \leq (r-1)\rho$ (zero otherwise).

The output layer of the neural network is modeled in two ways. In the *flexible* variant, a simple feedforward neural network is used with one or multiple hidden layers, applying a sigmoid activation to each output $r \in \{1, \ldots, R\}$ to end up with estimates of the hazard rate.

The *proportional hazard* (PH) variant uses the proportional hazard assumption. I.e., it assumes the hazard rate has the form $\lambda_i(t) = \lambda_0(t)\exp(\mathbf{x}_i^T\boldsymbol{\beta})$, $\boldsymbol{\beta} \in \mathbb{R}^P$ being a weight vector. Following [116][p. 43], when assuming intrinsically discrete time, the (now also discrete) estimated hazard rate $\hat{\lambda}_i(r)$ can be written as

$$\hat{\lambda}_i(r) = \frac{1}{1 + \exp(\alpha_i(r) + z_i)}, \tag{5.6}$$

with z_i and $\alpha_i(r)$ being outputs of different feedforward neural networks. Here, $\alpha(r)$ has as input the covariate vector $\mathbf{x}_i$, followed by one or multiple hidden layers. The variable z_i is obtained by a weighted average over the elements in the last hidden layer. Note that (5.6) is a sigmoid activation function, which simplifies the implementation in for example Keras.

For both the PH and flexible approach, we find a loss function in the form of the binary cross-entropy

$$\mathcal{L} = \sum_{i=1}^{I}\sum_{r=1}^{R}[\gamma_{i,r}\log(\hat{\lambda}_i(r)) + (1-\gamma_{i,r})\log(1-\hat{\lambda}_i(r)v_{i,r})]. \tag{5.7}$$

Note that when $r\rho > T_i$ and $\delta_i = 0$, we have $y_{i,r} = 0$ and $\hat{\lambda}_i(r) = 0$. Therefore, these predictions do not contribute to the log-likelihood.

Recurrent neural networks In addition to the two feedforward models, we also consider a recurrent neural network. The architecture of the model is depicted in Figure 5.1. We either use a standard recurrent neural network with a Gated Recurrent Unit (GRU) [46], or with a Long short-term memory (LSTM) unit [104]. As we do not include time-varying covariates, only at $r = 1$ an input vector is inserted. At the other time periods, a vector containing only zeros, denoted by $\mathbf{0}$, is fed into the network. Also here we multiply (element-wise) the output vector $(\hat{\lambda}_i(1), \ldots, \hat{\lambda}_i(R))$ by the censoring vector $(v_{i,1}, \ldots, v_{i,R})$ to exclude observations after censoring.

$$\begin{tikzcd}
\mathbf{x}_i \arrow[r] & \boxed{\text{GRU/LSTM}} \arrow[r] \arrow[d] & \hat{\lambda}i(1) \arrow[r, "\cdot v_{i,1}"] & \hat{\gamma}_{i,1} \\
\mathbf{0} \arrow[r] & \boxed{\text{GRU/LSTM}} \arrow[r] \arrow[d] & \hat{\lambda}i(2) \arrow[r, "\cdot v_{i,2}"] & \hat{\gamma}_{i,2} \\
& \cdots \arrow[d] & & \\
\mathbf{0} \arrow[r] & \boxed{\text{GRU/LSTM}} \arrow[r] & \hat{\lambda}i(R) \arrow[r, "\cdot v_{i,R}"] & \hat{\gamma}_{i,R}
\end{tikzcd}$$

Figure 5.1: Sequential architecture

5.3.3 Model evaluation

Brier score and C-index

To evaluate the survival models, we use the Brier score and the C-index, and we consider a conditional integrated C-index, which will be explained in Section 5.3.3. As discussed briefly in Section 5.3.1, evaluation is performed by considering the number of failures in binned time intervals $u_1, \ldots, u_R$. We define the Brier score at some interval u_r over some validation set $\mathcal{I} \subseteq \{1, \ldots, I\}$ as [175]

$$B_{\mathcal{I}}(r) = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} [y_{i,r} - \hat{S}_i(r\rho)]^2, \tag{5.8}$$

where $y_{i,r} = \mathbb{1}_{\{T_i > r\rho\}}$. Hence, the Brier-score is evaluated at the end of each time interval.

The C-index considers the correct ordering of estimated failure times, rather than predicting the survival curve correctly. The C-index, again over a validation set $\mathcal{I}$, is defined as [232]

$$C_{\mathcal{I}}(r) = \frac{\sum_{i_1, i_2 \in \mathcal{I}} w_{i_1,i_2} \hat{w}_{i_1,i_2}(r) \delta_{i_1}}{\sum_{i_1, i_2 \in \mathcal{I}} w_{i_1,i_2} \delta_{i_1}}, \tag{5.9}$$

with $\hat{w}_{i_1,i_2}(r) = \mathbb{1}_{\{\hat{S}_{i_1}(r\rho) \leq \hat{S}_{i_2}(r\rho)\}}$, and $w_{i_1,i_2} = \mathbb{1}_{\{T_{i_1} < T_{i_2}\}}$. I.e., such that $\hat{w}_{i_1,i_2} w_{i_1,i_2}(r)\delta_{i_1} = 1$ if the estimated order of failure is correct, and observation i_1 (which failed first) was not censored, zero otherwise.

We choose not to incorporate IPCW weights [234], as the number of observations for the censoring distribution at certain points in time was rather limited. As the number of combinations (i_1, i_2) can grow large in large datasets, we sample 10,000 jobs at random, over which the C-index is computed. Ties are broken at random. Both the Brier score and C-index can be averaged over all $r \in \{1, \ldots, R\}$ to obtain the integrated Brier score and C-index. These are denoted by $\bar{B}_{\mathcal{I}}$ and $\bar{C}_{\mathcal{I}}$ respectively.

Integrated Conditional Concordance Index (ICCI)

Although the C-index provides some insight into the ability of survival models to correctly assess which candidate will switch jobs first, it may be less suited in practice. It assesses failure times at the start of both jobs, and all candidate pairs in the data are considered, even candidates occupying very different job types. In practice, however, we expect to encounter candidates who have been already occupying their current job for some time τ. Furthermore, when comparing candidates, we expect recruiters will be more frequently comparing candidates with similar backgrounds. Therefore, we considered an alternative error metric, which we refer to as the Integrated Conditional Concordance Index (ICCI).

The computation is similar to that of the C-index. However, we only consider pairs in some validation set $\mathcal{I}_c$ that satisfy some condition c, and we use a truncated conditional expectation of the remaining survival time instead of $\hat{w}_{i_1,i_2}(r)$ in (5.9). We used three conditions of selecting the pairs in $\mathcal{I}_c$. The first is a stratified sample over the *function_group* and *transitionlustrum* (both will be defined in Section 5.4). The second is a sample in which the same number of items is drawn from each combination of *function_group* and *transitionlustrum*, and the third approach is a random sample over all possible pairs.

More formally, we condition the observations T_{i_1} and T_{i_2} on that already a duration of length τ_1 and τ_2 has passed for these two jobs. I.e., the remaining tenure equals $(T_{i_1} - \tau_1)^+$ and $(T_{i_2} - \tau_2)^+$ respectively, where $a^+ = \max(a, 0)$. Also, let

$$Q_i(t, \bar{T}) = \int_{\tau=t}^{\bar{T}} S_i(\tau) d\tau. \tag{5.10}$$

I.e., this is the truncated (at time $\bar{T}$) expected remaining tenure for job i, given that job i has already been occupied for a time of length t.

Hence, following the same logic as for the C-index, we now define

$$w_{i_1,i_2}(\tau_1,\tau_2) = \mathbb{1}_{\{(T_{i_1}-\tau_1)^+ < (T_{i_2}-\tau_2)^+\}}, \tag{5.11}$$

$$\hat{w}_{i_1,i_2}(\tau_1,\tau_2) = \mathbb{1}_{\{Q_{i_1}(\tau_1,R\rho) < Q_{i_2}(\tau_2,R\rho)\}}, \tag{5.12}$$

from which the ICCI for pairs following condition c is defined as

$$\text{ICCI}_{\mathcal{I}_c}(\tau_1,\tau_2) = \frac{\sum_{i_1,i_2\in\mathcal{I}_c} w_{i_1,i_2}(\tau_1,\tau_2)\hat{w}_{i_1,i_2}(\tau_1,\tau_2)\delta_{i_1}}{\sum_{i_1,i_2\in\mathcal{I}_c} w_{i_1,i_2}(\tau_1,\tau_2)\delta_{i_1}}. \tag{5.13}$$

The ICCI can further be averaged over different values of τ_1 and τ_2 to obtain one ICCI.

5.4 Data preparation

5.4.1 Engineering of existing features

The data used in this study was extracted from résumés, which were uploaded to the Dutch job board Gus [98]. The jobs to which these applicants applied are temporary jobs. Although the dataset contained some non-Dutch résumé, these were removed as they complicate the NLP procedures, while only being a negligible part of the dataset. The job seekers applied between 2005-01-01 and 2016-10-17. In total, the dataset contains 50,000 unique job seekers, which we split into a training, validation and test set according to a 70/10/20 split. In total, the dataset encompasses 131,059 unique jobs. To avoid data dredging, all statistics presented in this section are based on the training set.

Since the résumés are plain text documents, we used technology from Textkernel [218] to extract information from the text. We use the common convention in résumés to include the job history in a table. In this table, each record includes a (textual) description of the previous job, and the start and end dates of the job. From this data, we extracted the variables *transition lustrum* (year in which job seeker j started job h, grouped into clusters of 5 years), *order* (number of previous jobs job seeker j had occupied just after starting job h), and *expdays* (total observed work experience of candidate j just before the start of job h in days).

We also extracted the *edu_ lvl* (the candidate's highest education level, mapped to the Dutch education system), *age* (candidate's age at the start of the job), *gender*, and the *job description* given by the candidate. The job description was mapped to a vector space in two ways. The first approach used a classification model from Textkernel, which maps the job to a three-layer hierarchical classification (of which the upper two were used as covariates) and classifies the *industry* of the job. The second approach is unsupervised, in which we trained a *word2vec* [173] model on the candidate's previous job description.

5.4.2 Computing tenure from parsed résumé data

From analyzing the job seekers' start and end dates, one can observe that job seekers tend to indicate the start and end date of each job at different levels of granularity. Some indicate the start and end dates on a monthly level, whereas the majority indicate these dates on a yearly level. In case candidates indicate their start and end dates on a yearly level, we considered this a case of interval-censored data. Besides the interval censoring, the start and end dates also may incorporate other types of censoring. The percentage of job occupations having a particular type of censoring is indicated between the brackets in the remainder of this section.

Year interval (42.20%) indicates candidates have rounded the start and end date of their occupation to entire years. *No censoring* (38.00%) implies the candidate has indicated his/her occupation in months. *Right NLJ* (Right censoring, Not Last Job; 10.3%) indicates the start date is either rounded to a monthly or yearly level, but the end date of the job is missing. However, the candidate does have a job in his/her résumé with a start date later than the job under consideration. Hence, this type of censoring can still be seen as interval censoring, as we do know the job ended before the start date of the next job.

Right LJ (Right censoring Last Job; 5.87%) indicates the start date is on a monthly or yearly level, but the end date is missing. Also, the candidate has no other job where the start date is after the start date of the job under consideration. In this case, we assume the occupation of the candidate did not end before the moment he/she applied to the vacancy, and is therefore considered as a case of right censoring. Likewise, *Left NFJ* (Left censoring, Not First Job; 0%) and *Left FJ* (Left censoring, First Job; 0.03%) indicate cases of left censoring. Here, if the job is not the first job, the start date can be guessed by using the end date of the previous job. Last, *Complete* (1.62%) indicates both the start and end date are missing.

To avoid our estimates to be biased by these different types of censoring, we applied the following procedure. First, all observations with *Complete* and *Left FJ* censoring were removed. The former were removed because they are non-informative. The latter, as these only encompass a very small number of jobs. In the case of *Right NLJ*, the end date was inferred by considering the start date of the next job. If this start date was rounded to entire years, the observation was relabeled as *Year interval*, after which we processed these observations in the same way as for the other *Year interval* observations. Although the same procedure could be applied to observations having *Left NFJ* censoring, using the start date of the previous job, we had no observations with this type of censoring.

In case the candidate was only censored in the start and/or end year of his/her occupation (*Year interval*), we added a random number of months to the start and/or end year of the occupation. These random number of months followed the overall distribution of monthly turnover. Because of the year interval censoring, the probabilities for January and December were inflated. To correct for this, we used the monthly distribution of the start date, and removed the probabilities for these months. After removal, the probabilities for January and December were estimated by fitting a cubic spline between the probabilities of the remaining months. Observations without censoring remained in the dataset as-is. The new start and end dates are further referred to as the adjusted start and end dates.

Besides removing and correcting censored data, we also removed observations having occupations with tenures lasting longer than 50 years, occupations that started before the candidate's 18th birthday, occupations that started after the candidate's 67th birthday, and observations with negative tenures. To reduce the number of unique values for categorical attributes, we reassigned categorical values with fewer than 30 observations to a category "other". Missing data was imputed using another random forest algorithm as described by Ishwaran et al. [111], which is implemented in the `RandomForestSRC` R package [110]. To fit this random forest model, the package's default parameters were used.

5.4.3 Transforming job descriptions to a vector space

Recently, *doc2vec* [138] and *word2vec* [173] have grown in popularity, and have shown to outperform other state-of-the-art word and document embedding procedures [33]. Although we originally tried both methods, we found that the document vectors from *doc2vec* were little discriminatory, and therefore we continued using only *word2vec* for document embedding.

Before training the *word2vec* model, we removed (Dutch) stopwords, and the words were stemmed using the *Snowball* stemmer [27]. We used a vocabulary of 20,000 unique words having the largest *tf-idf* values. Both models were trained using negative sampling with a sampling factor of 10^{-5}, from which word pairs were constructed using skip grams with a window size of 3, as larger window sizes did not improve the results.

The *word2vec* model was trained using Keras with a Tensorflow backend [79, 1]. We used an embedding size of 64; 100 training epochs; a batch size of 65,536; an initial learning rate of 0.1; and we used *rmsprop* [221] to update the learning rate in subsequent epochs. To obtain document vectors from word vectors, we computed a weighted average over the word vectors for each job description. As weights we used the *tf-idf* value of each word. We did experiment with different epochs, batch sizes, and initial learning rates; these did not improve the results.

5.4.4 Summary of the dataset

A numerical summary of the training data is shown in Table 5.2. Perhaps the most important implication of the data is the number of missing values. For some variables, the most frequent category is "NA". The summary further shows that, although the dataset contains heterogeneous jobs, the majority of jobs relates to administrative and secretary jobs, and commerce-related jobs.

Figure 5.2 shows the number of previous jobs candidates have held (a), and the estimated survival function of tenure using the Kaplan-Meier estimate over all jobs (b). Most job histories are relatively short: 52% of the job seekers have fewer than four previous jobs. Furthermore, when candidates switch from one job to another, this job was classified as having the same *function_group* in approximately 31% of the cases. Hence, most job transitions are between similar jobs. From examining the raw job descriptions, one also finds that job seekers can have different definitions of a job. In particular, some candidates split different jobs for the same employer into multiple jobs in their résumé, whereas others consider this as one job, with multiple responsibilities.

Table 5.2: Data summary

Feature	Type	Summary
citycluster	cat	4 cat., most frequent: "cluster4" (3.47%), NA: 95%
education_ level	cat	5 cat., most frequent: "Beroepsonderwijs" (22.53%), NA: 45%
function_ class	cat	25 cat., most frequent: "Administratie en klantenservice" (19.50%), NA: < 1%
function_ group	cat	287 cat., most frequent: "Administratief medewerkers" (6.24%), NA: < 1%
gender	cat	2 cat., most frequent: "2" (=female) (13.63%), NA: 75.41%
month_ of_ startdate	cat	12 cat., most frequent: "12" (=December) (62.86%), NA: < 1%
sector	cat	23 cat., most frequent: "Handel" (12.11%), NA: < 1%
transitionlustrum	cat	12 cat., most frequent: "2005-2010" (34.57%), NA: < 1%
year_ of_ birthlustrum	cat	9 cat., most frequent: "1960-" (11.86%), NA: 43.94%
expdays	Num	mean= 3679.88, standard dev.= 3705.06, NA: 20.34%
expdaysfunctiongroup	Num	mean= 583.61, standard dev.= 1531.59, NA: 10.59%
order	Num	mean= 4.13, standard dev.= 3.05, NA: < 1%
orderperfunctiongroup	Num	mean= 1.77, standard dev.= 1.43, NA: < 1%

We also noted that it is quite common for job seekers to occupy (at least) two jobs at the same time somewhere during their career: 52% of the job seekers have at least one overlapping job pair. For these jobs, we determine the job order h by the start date of the job (ties were broken at random), and features related to experience (*expdays*,*expdaysfunctiongroup*, *order*, and *orderperfunctiongroup*) were computed using only completed jobs.

The survival function in Figure 5.2 shows that most candidates have a rather short tenure. After one year, approximately 50% of the candidates already switched jobs. Although this is not representative for the Dutch workforce as a whole, it is common for temporary workers to have shorter tenures [38]. We also note that we do not take into account whether the turnover was voluntarily or involuntarily, and we did not include data related to gaps in the résumé.

5.5 Results

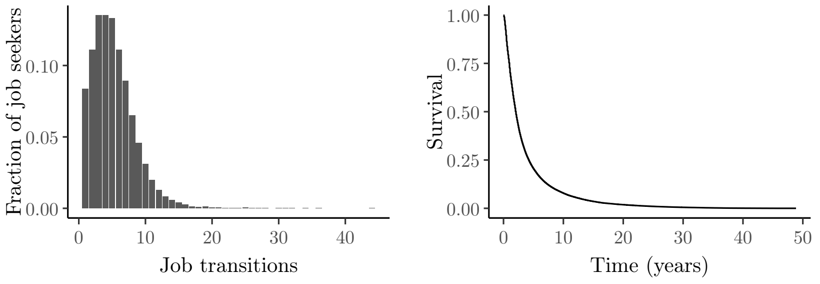
5.5.1 Overall performance

Table 5.3 gives an overview of the grid search we applied to the validation set to find appropriate values for the models' hyperparameters. The obtained best parameter values are given in the last column of Table 5.3. We choose periods of 3 months (i.e., $\rho = 0.25$) as an appropriate trade-off between model accuracy and practical usage. I.e., although recruiters would prefer short periods

Figure 5.2: Number of previous jobs (a) and the estimated overall survival curve (b).

(a) Number of previous jobs held by the job seekers in the training set.

(b) Kaplan-Meier estimate of the survival curve.



(of say one month), this would lead to reduced accuracy, making the models unreliable. Hence, three months was considered as a proper trade-off between the two.

As Table 5.2 suggests, our dataset contains some variables that could introduce unwanted discrimination in terms of *gender* and *age*. Instead of removing these attributes upfront, we include them while training the model, but impute them by their overall average value (in case of categorical values, we impute after dummification) during validation. However, it should be noted that this procedure was only partially effective, due to the many missing values for both *year_ of_ birthlustrum* and *gender* (Table 5.2).

Figure 5.3 shows the resulting Brier and C-index on the test set, whereas Table 5.4 shows the integrated and normalized scores. Note that the different types of sampling to compute the ICCI (stratified, equal per group, and random) refers to the sampling method described in Section 5.3.3. Since the Kaplan-Meier estimate is the model with the least complexity (i.e., it does not take into account any covariates), the results of the KM model are emphasized in Figure 5.3.

The abbreviations have the following interpretation: *NN-Flex*, *NN-PH*, *NN-GRU*, and *NN-LSTM* represent the flex, proportional hazard, Gated Recurrent Unit, and Long Short-Term Memory neural networks respectively (Section

Table 5.3: Hyperparameter grid search

Model	Hyperparameters	Best found parameters
Kaplan-Meier	NA	NA
Cox-PH with elastic net penalty Survival tree	$\alpha \in \{0 \text{ (Ridge)}, 0.5, 1 \text{ (Lasso)}\}$ penalty weight as in [209] term. node size $= 6$	$\alpha^* = 1$ (Lasso), penalty$^* = 0.0107$
Random survival forest	trees $\in \{100, 500, 1000\}$, term. node size $= 6$, depth $\in \{6, 12\}$, random split points $= 5$, tree feature sample size $= \sqrt{P}$	trees$^* = 500$, depth$^* = 12$
Cox boosting	trees $\in \{1000, 2500, 5000\}$, shrinkage $\in \{0.001, 0.05, 0.01, 0.1\}$ depth $\in \{3, 6\}$	trees$^* = 2,500$, shrinkage$^* = 0.01$ depth$^* = 3$
Neural survival non-sequential	hidden units $\in \{64, 128\}$, hidden layers $= 2$, epochs $= 100$, batch size $= 65,536$, learning rate $\in \{0.001, 0.01, 0.1\}$	hidden units Flex $^* = 128$ hidden units PH $^* = 64$ learning rate Flex$^* = 0.01$ learning rate PH$^* = 0.001$
Neural survival sequential	time periods $= 21$, hidden layers $\in \{1, 4, 16\}$, epochs $= 100$, batch size $= 9,292$, learning rate $\in \{0.001, 0.01, 0.1\}$, drop out $= 0.1$	hidden layers GRU$^* = 16$ hidden layers LSTM$^* = 16$ learning rate GRU$^* = 0.001$ learning rate LSTM$^* = 0.001$

5.3.2). *RSF*, and *GBM* represent the Random Survival Forest and Gradient Boosted trees (Section 5.3.2). *Cox Lasso*, *Surv. tree* and *KM* represent the Cox proportional hazard model with a Lasso penalty, a single survival tree, and the Kaplan-Meier estimate (Section 5.3.2). All models are fitted using the best hyperparameters found during training (Table 5.3). For readability of the figures and tables, we only include the best performing sequential (*NN-GRU*) and feedforward (*NN-Flex*) neural network.

Gradient boosted trees and random survival forests produce the best results, with a slight preference for GBM. Interestingly, the neural models and the Cox model barely outperform the Kaplan-Meier estimate both in terms of the Brier score and C-index. The single survival tree shows a trade-off between the Brier score and C-index. For $t < 3$, it shows reasonable performance in

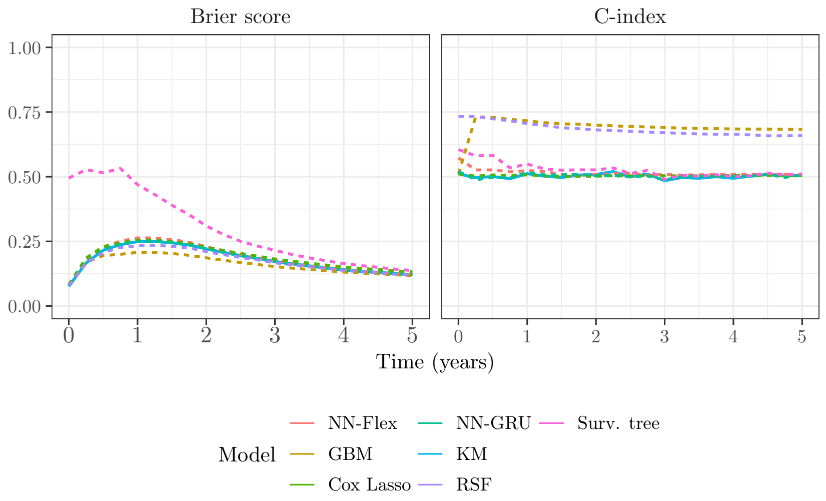


Figure 5.3: Brier score and C-index over time

terms of the C-index, but the results are poor in terms of the Brier score.

The good performance of random survival forest and GBM seems to diminish when we include conditional survival times, as shown in Table 5.4: although in absolute terms the ICCI for GBM and random survival forest are somewhat comparable to their C-index, the values are also closer to the results of a KM-estimator. Furthermore, taking different kinds of samples only had a marginal impact on the ICCI. Hence, when predicting the correct order, the advantage of using more complex models seems to diminish.

5.5.2 Performance on sub-datasets

Next, we split the results per *function_group* in order to study differences in predictive ability for different job types. As GBM has the best results over the combined score (Table 5.4), we use this model for further inference. The results over the five largest job types in the dataset are shown in Figure 5.4. As we may have expected from the Brier scores in Figure 5.3, which are somewhat similar to those of a Kaplan-Meier estimate, the fitted survival curves for the different job types are also rather similar.

Table 5.4: Integrated Brier score, C-index and ICCI

Model	Integrated Brier	Integrated C-index	ICCI stratified	ICCI eq. per group	ICCI random
GBM	**0.16**	**0.69**	**0.66**	**0.66**	**0.67**
RSF	0.17	0.68	0.65	0.65	0.66
NN-Flex	0.19	0.51	0.64	0.64	0.63
NN-PH	0.18	0.51	0.64	0.65	0.63
NN-LSTM	0.18	0.51	0.63	0.64	0.64
NN-GRU	0.18	0.50	0.63	0.62	0.63
KM	0.18	0.50	0.64	0.64	0.64
Cox Lasso	0.19	0.50	0.64	0.65	0.63
Surv. tree	0.30	0.53	0.56	0.55	0.54

We also considered the effect of excluding certain attributes from the model. To do so, we construct four sub-datasets: 1) a dataset in which age and gender were not imputed, 2) a dataset without the *word2vec* word embedding, 3) a dataset in which we exclude attributes derived from the job classifier (i.e., excluding *function_ class*, *function_ group*, *sector*, *expdaysfunctiongroup*, and *orderperfunctiongroup*), and 4) a dataset including only features related to time-dependent variables (i.e., including *transitionlustrum*, *order*, *expdays*, *month_ of_ startdate*, *orderperfunctiongroup*, and *expdaysfunctiongroup*). A comparison between the performance of GBM on these datasets and the full dataset is shown in Figure 5.5. Especially inclusion of the time variables caused a substantial improvement in both the Brier sore and C-index. Inclusion/exclusion of other types of attributes has a negligible effect.

5.6 Conclusion and further research

In this paper, we considered to what extent machine-learned survival models are capable of predicting (job) tenure, extracted from job seekers their résumé. Here (job) tenure is defined as the time difference between the start date and the end date of a job, where this job is indicated by a job seeker in the job history section of his/her résumé. To predict these tenures, several attributes were extracted from the job seekers' résumés. These include attributes obtained from candidates' own job descriptions, which were extracted using an existing text classifier and a trained word2vec model, and several attributes related to work experience. To make the predictions, we compared multiple machine-learned survival models in terms of the Brier score and C-index to determine which of these models performs best.

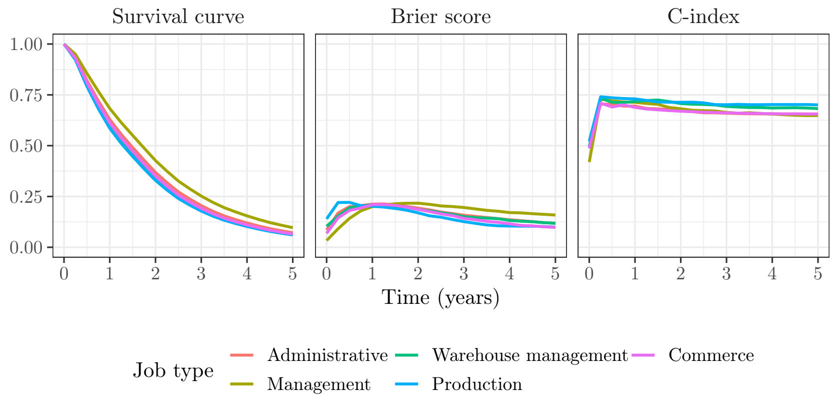


Figure 5.4: Results over the 5 most common job types

From our comparison of machine-learned survival models, we find that especially tree-based ensembles, such as a random survival forest and Gradient Boosting Machines, work well to predict job tenure from unstructured résumés. They outperformed benchmark models in terms of the Brier score and C-index. These benchmarks included a Kaplan-Meier estimator and Cox regression, but also more complex models such as neural survival models and recurrent neural networks. Especially the importance of time-related variables in these models is interesting. Matching jobs to vacancies is often done using semantic overlap. E.g., by comparing skill overlap between the vacancy and job. Our results suggest that including time-related variables in these matching algorithms may improve their performance.

Although tree-based ensembles outperformed benchmark models, still the prediction problem remains difficult. The difference with benchmark models are relatively small, and if one takes into account conditional survival times, and compares more similar job pairs, the error scores between the tree-based ensembles and benchmark models become more similar.

This limited performance may be explained in several ways. First, it should be acknowledged that predicting job tenure from résumés is a difficult prediction problem. Previous work (using mostly Cox-PH models [105]) finds only weak correlations between predictors and tenure, (r^2 between 0.33 and 0.37) [36, p. 261]. Second, as illustrated in this study, résumés come with considerable

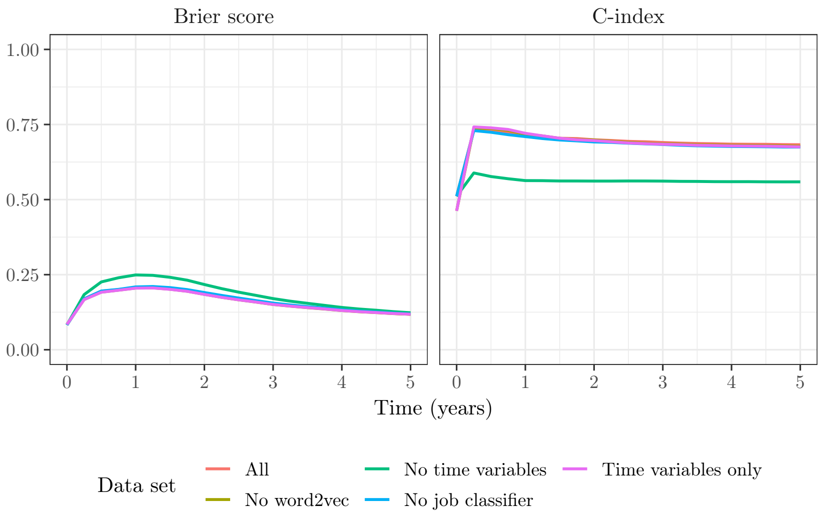


Figure 5.5: Scores on other datasets

fuzziness. Turnover itself may be indicated at different levels of granularity. Missing data is a considerable problem, as the résumé parser has to deal with a variety of formats. Also, job seekers may have different definitions of a job. E.g., one might define two positions at the same employer as one job, whereas another will consider these as two jobs. Naturally, such fuzziness complicates interpreting models derived from résumés.

Given these results, we are in particular interested in two directions for further work. Given the fuzziness of résumé data, an interesting direction would be to study whether survival analysis on résumés could benefit from models trained in different contexts, i.e., transfer learning. One could think of applying language models trained on larger corpora. But also training survival models on corporate turnover data, for which we expect to have more precise measurements, would be an interesting direction.

A second direction is with regards to the practical implications of predicting tenure from résumés for candidate recommendation. It would be interesting to consider how these models compare with semantic matching methods, using more application-directed error scores such as NDCG. From a practical perspective, further research could also consider whether the model benefits from asking the user for additional data when uploading one's résumé.

Chapter 6

Predicting Applications on Recruitment Websites

The Internet has substantially changed how organizations market their vacancies and how job seekers look for a job. Although this has many benefits, such as simplifying the communication, it can also cause problems. Some vacancies are obtaining more applications than can be handled by the recruitment department, while other vacancies may remain unfulfilled for a long time. Data analysis might reveal insights into what strategies are effective to solve these problems. To analyze these problems, we therefore consider the predictability of the number of applications per vacancy per week, and to what extent this can be controlled using online marketing campaigns. After testing the predictive quality of several machine learning methods on a dataset from a large Dutch organization, we found that a Random Forest model gives the best predictions. Although these predictions provide insights into what recruiters and hiring managers can expect when publishing a vacancy, the error of these predictions can be quite large. Also, although the effect of online marketing campaigns on the number of applications is significant, predicting the effect from historic data causes problems due to collinearity and bias in the usage of these campaigns. Nevertheless, these predictions are insightful for both recruiters and hiring manager, to manage their expectations when publishing a vacancy.

6.1 Introduction

The internet revolution has substantially changed how job seekers look for a job and how organizations attempt to attract job seekers [225]. Already in 2003, 94% of the global Fortune 500 companies were using a corporate recruitment website to attract job seekers [90]. Also, online sources such as social media, online professional networks, and company websites are being used for effective employer branding [2]. Furthermore, the percentage of job seekers who are using the internet is growing steadily [216].

Advantages of using corporate recruitment websites have been discussed in previous studies, showing benefits including cost effectiveness, speeding up the hiring process, and ease of use both for recruiters and job seekers [255, 140, 89]. There is, however, yet another benefit of using corporate recruitment websites that has not been explored by previous research. That is, it enables tracking the behavior of job seekers on the website using e-commerce software. By tracking this job seeker behavior, recruitment departments can obtain valuable insights on how to attract or repulse applications. This might lead to strategies for reducing recruitment lead time and cost.

To take a first step into exploring how vacancies attract job seekers and how this might be controlled, this study considers the predictability of the number of job seekers that will apply to an online vacancy per week. This metric is referred to as the application rate. In order to predict this metric, multiple machine learning techniques including Random Forest, Support Vector Regression, and Artificial Neural Networks were applied. The data used to predict the application rate included characteristics of the vacancy such as work location, required education level, and job title. Furthermore, also data describing whether the vacancy was used in online marketing campaigns such as Google Adwords, other vacancies on the website that might compete with the vacancy, and time related attributes such as the current recruitment lead time and application rates in weeks prior to the predicting period were used.

This chapter has the following structure, Section 6.2 provides an overview of previous literature on the effectiveness of online recruitment websites. Section 6.3 will discuss how data was obtained and prepared for this analysis. In Section 6.4, the findings from preliminary data analysis will be discussed that affect the choice of predictive models. Section 6.5 will give an overview of the methods that were used to predict the application rate. Finally, Section 6.6 provides an overview of the predictive quality of these methods along with its implications.

6.2 Related work

The ability of corporate recruitment websites to attract job seekers has been considered in multiple studies, often by sending questionnaires to either job seekers or employers. The results of these studies differ. Some show the potential in terms of cost effectiveness, reducing recruitment leadtimes, and ease of use for both recruiters and job seekers [255, 140, 89]. Other studies, however, show a more modest perception. Brown [29] found that 75% of all job seekers find recruitment websites too complicated. This perception is shared by Maurer and Liu [170], who identified the management of potential information excess on corporate recruitment websites as being one of the key design issues for recruitment managers. Besides the excessive information organizations might send to potential job seekers, the opposite also holds. Vacancies might receive a large number of applications, including many unsuitable ones. Parry and Tyson [186] found that this is one of the reasons why a quarter of the organizations they examined who were using internet recruitment methods found it unsuccessful.

Machine learning can play a role in managing the information spread by both the employer and job seeker. In particular, it can be used to investigate the relationship between recruitment efforts and recruitment outcomes. These relationships can be used to control the quality and quantity of applicants, and the quality of the employer's brand.

Previous research has not paid much attention to how machine learning could be applied to manage the information spread by employers and job seekers, apart from application selection [214], and résumé parsing [39]. Although these methods automate part of the recruitment job, thereby enabling recruiters to handle a large number of applications, being able to control the quality and quantity of applicants would also decrease the workload of recruiters. Furthermore, fewer but better qualified applicants also means fewer rejections, which is beneficial for both job seekers and employers.

6.3 Data gathering and preparation

Data gathering To study whether the number of applicants who apply to a vacancy can accurately be predicted and controlled, data was gathered from a large Dutch company that employs over 30,000 people and has on average 150 vacancies on its corporate recruitment website.

Data was gathered from three systems. First, from an Application Tracking System (ATS), in which vacancy characteristics are stored such as work location, required education level, and working hours. Second, data was gathered from the corporate recruitment website's Google Analytics account. In particular, how many job seekers visited the corporate recruitment website per week, and how frequent job seekers followed different paths from the website's landing page to the application submit page (the webpage visited after having submitted an application). Also, Google Analytics is capable to keep track of whether job seekers visited the website via a paid hyperlink that was part of an online marketing campaign. This data was used to determine which vacancies had been used in online marketing campaigns. Third, the number of weekly tweets the recruitment department published via their recruitment Twitter account was gathered, along with whether certain vacancies were mentioned in a tweet.

Combining these three data sources gives per vacancy v, per time period t (in weeks), the vacancy characteristics of v, whether v was used in certain online marketing campaigns, and how many job seekers navigated from the landing page to the vacancy's submit page during time period t. This dataset was extended with time-related data such as the recruitment lead time at time t, and application rates of a vacancy in weeks prior to week t.

The dataset was split into a test and training set. The training set contained all values between 2013-08-26 and 2015-09-31, whereas the test set contained all values between 2015-10-01 and 2015-12-31. This split was chosen for two reasons. First, at the time of splitting the dataset there was no knowledge of possible time dependency in the data. If the application rate would include this time dependency, then validating the predictive model on the last period of the total dataset would produce the most realistic evaluation. Second, three months is the maximum period for which it is safe to assume that the vacancy portfolio over that period is known.

Data preparation To improve the quality of the dataset, multiple operations were performed. Attributes related to work location and job title contained many possible categorical values, which was not practical for analysis. To reduce the number of categorical values the locations were clustered based on similarities in their application rate probability density. These probability densities were clustered using K-means clustering. To find an appropriate number of clusters, the Akaike Information Criteria (AIC) was used, which was computed for $K = 1, \ldots, 10$ clusters. If a cluster had fewer than 100 observations, the observations were assigned to the cluster closest to the overall mean application rate. Besides location, attributes the job title had even more unique values, making the usage of the probability density unpractical. As an alternative, similar job titles were identified and clustered manually.

In order to identify attributes having a small variance, the frequency cut-off from the `nearZeroVar` function of the `caret` package was used [134]. Since all predictors are either binary, categorical or discrete, it was possible to apply this procedure on all predictors. Let $N_{i,j}$ be the frequency of a value i of category j. Furthermore, let $N_{(l),j}$ be the lth order statistic of $N_{1,j}, \ldots N_{n,j}$, then we have frequency ratio: $F_j = \frac{N_{(n),j}}{N_{(n-1),j}}$. Thus F_j gives the ratio of the most frequent and second most frequent value of attribute j. Attributes were removed from the dataset if $F_j > 19$.

During the last data preparation step, categorical attributes were dummified into binary vectors. The feature values x_{ij} were normalized using $\tilde{x}_{ij} = \frac{x_{ij} - \bar{x}_j}{s(x_j)}$. Here $\bar{x}_j$ and $s(x_j)$ are the mean and standard deviation over the values of attribute j respectively.

6.4 Exploratory data analysis

The application rate When considering possible probability distributions of the application rate, a Poisson distribution would come first to mind. However, as Figure 6.1 suggests, the Poisson distribution does not seem to fit the data well. The application rate's distribution is more zero-inflated and overdispersed than a Poisson distribution. Dependent on the nature of the vacancy, a log-normal or negative binomial distribution is more appropriate. The distribution also confirms previous research, stating that some vacancies can attract a large number of applications [186]. In fact, 10% of the rates account for 53% of all applications.

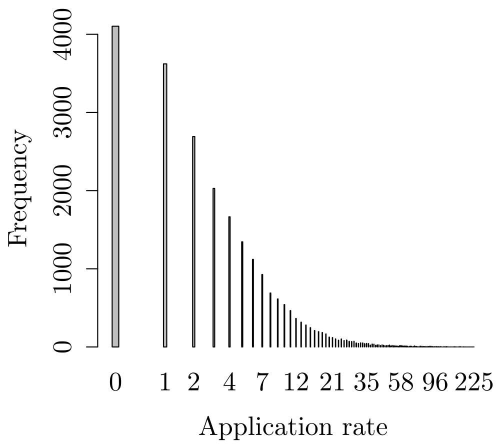


Figure 6.1: Histogram application rate

The total number of applications and sessions Besides considering the distribution of the application rate, also the predictability of the total number of applications per week was considered. To predict these metrics, the structure of the vacancy portfolio and the used online marketing campaigns were used as predictors. This analysis might already give an indication of how online marketing campaigns can affect both the traffic to the website and the number of applications. Furthermore, if the residuals of this model would be highly correlated, this could be an indication of time dependency, which would effect the selection of predictive models for the application rate.

To predict the number of applications and sessions per week, the status of the online vacancy portfolio and the number of vacancies subject to certain online marketing campaigns were used as predictors. The status of the vacancy portfolio was determined by counting the number of online vacancies having certain characteristics such as the same location, work area, job description, and required education level. A linear regression model was used to determine the effect of the online vacancy portfolio and online marketing campaigns on the total number of sessions and applications. This linear regression model did not include any interaction effects. Although also more complex methods can be applied, the number of observations was relatively small compared to the number of predictors. Therefore, using more complex methods was likely to cause overfitting. A backwards AIC algorithm was used to include only those predictors having the most predictive value.

Applying these methods gave R^2 values of 0.68 and 0.56 when predicting the weekly number of sessions and weekly number of applications respectively. The models also indicate that it is difficult to determine the exact effect of online marketing campaigns on the total number of weekly sessions and applications.

Although the marketing campaigns are significant, the campaigns are frequently used in combination with each other, which makes it difficult to distinguish the effect of a single campaign. This can be seen if we compute the condition indices and variance decomposition proportions. If we add up the variance decomposition proportions obtained from the number of vacancies subject to Facebook, Indeed and Google campaigns over the largest condition indices (85, 102 and 128 resp.), this sum becomes 0.692, 0.797 and 0.91 respectively, which are larger than our threshold value of 0.5. A possible remedy for this collinearity is to add more characteristics of the campaigns to the dataset, such as the profiles used in a Facebook campaign. This was however not considered in this study.

Besides collinearity, an increase in online marketing campaigns can also be a response to a small number of applications, which makes the estimated effect of online marketing campaigns on the number of session and applicants biased.

When considering the residuals of the models predicting the number of weekly sessions and applications, a Box-Pierce test showed that these residuals were correlated. However, when examining the autocorrelation and partial autocorrelation functions this correlation turns out to be small. Both the autocorrelation and partial autocorrelation show a maximum absolute correlation of 0.21, at lag 2 and 1 respectively. Therefore, for simplicity, it was found acceptable to assume that the residuals were uncorrelated. As a result it was assumed that the total number of applications per week is independent of the date of the measurement.

Best sources Also the relationship between the number of sessions originating from different websites via different devices, and the number of weekly applications was considered. Again, a linear regression model was used to avoid overfitting. Since visitors to the corporate recruitment website can originate from many different sources, only the top four sources causing most traffic were considered. Smaller sources were combined in an 'other' category. The source - device combinations causing most traffic to the website did not produce most applications. Where visitors originating from Google on a desktop produced most traffic to the website, changes in direct traffic on either desktop, mobile or tablet and traffic from the corporate website were the main drivers for changes in the number of weekly applications.

6.5 Methods

6.5.1 Method selection

To determine which methods would be most suited to predict the application rate, a number of considerations were taken. First, since the application rate is count data, its prediction is considered to be a regression problem. Second, exploratory data analysis found that the data is more zero-inflated and overdispersed than a Poisson distribution. Therefore, predictive models that incorporate zero inflation and overdispersion are preferred. Third, during exploratory data analysis, it was found that when predicting the total number of applications per week the residuals of this model are only slightly correlated. As a result, it was assumed that the total number of applications per week is independent of the date of the measurement. However, it still can be dependent on other time indicators, such as the current recruitment lead time.

Fourth, the dataset still contained a large number of attributes, some of which might not be useful for the predictive model. To reduce the number of attributes, methods that included some form of variable selection were preferred. Fifth, since a grid search was applied to find good model parameters, methods that were able to produce good results within reasonable time were preferred. Sixth, methods that have been applied successfully in other regression application were preferred.

Using these criteria seven methods were identified: Linear elastic net, Poisson elastic net, Tweedie elastic net, Classification And Regression Trees (CART), Random Forest, Support Vector Regression (SVR), and Artificial Neural Networks (ANN).

6.5.2 Method overview

Linear elastic net

Linear elastic net is a method that attempts to minimize the sum of squared error plus a linear combination of the Lasso and Ridge penalty. Let $PSSE(\lambda, \beta, \alpha)$ be the penalized sum of squared error with λ the weight of the penalty. α indicates to what extent either the Ridge or Lasso penalty is taken into account, and β is the effect vector to be estimated. $PSSE(\lambda, \beta, \alpha)$ is given by:

$$\begin{aligned} PSSE(\lambda, \boldsymbol{\beta}, \alpha) = & \tfrac{1}{2N} \textstyle\sum_{i=1}^{N} (y_i - \beta_0 - x_i^T \boldsymbol{\beta})^2 \\ & + \lambda \left[(1-\alpha) \tfrac{1}{2} ||\boldsymbol{\beta}||^2 + \alpha ||\boldsymbol{\beta}||_1 \right] \end{aligned} \tag{6.1}$$

To minimize (6.1), the `glmnet` R package was utilized, which applies a coordinate descent algorithm to estimate β [85]. To determine good values for λ and α a grid search was applied. For λ, $K = 100$ uniformly spread values between $\lambda_{max} = \frac{max_l |\langle x_l, y \rangle|}{N\alpha}$ and $\lambda_{min} = \epsilon \lambda_{max}$ were used. To find a good value for α, values from 0 up to 1 with increasing steps of 0.2 were used.

Poisson elastic net

Poisson elastic net is a combination of a generalized linear regression model and elastic net using the link function $g(\mu_i) = log(\mu_i)$, where $\mu_i = \mathbb{E}(y_i | \mathbf{x}_i)$. Instead of the sum of squared error, the log-likelihood is used to estimate $\boldsymbol{\beta}$. Let PLL be the penalized log-likelihood, then $\boldsymbol{\beta}$ is found by maximizing (6.2).

$$\begin{aligned} PLL(\lambda, \boldsymbol{\beta}, \alpha) = & \tfrac{1}{2N} \textstyle\sum_{i=1}^{N} \left[y_i \mathbf{x}_i^T \boldsymbol{\beta} - \exp\left(\mathbf{x}_i^T \boldsymbol{\beta} \right) \right] \\ & - \lambda \left[(1-\alpha) \tfrac{1}{2} ||\boldsymbol{\beta}||^2 + \alpha ||\boldsymbol{\beta}||_1 \right] \end{aligned} \tag{6.2}$$

To maximize (6.2), again the `glmnet` R package was used. In case of Poisson regression, `glmnet` iteratively creates a second order Taylor expansion of (6.2) without the penalty, using current estimates for $\boldsymbol{\beta}$. This Taylor expansion is then used in a coordinate descent algorithm to update $\boldsymbol{\beta}$ [85, 102]. To find appropriate values for λ and α, the same grid search as in linear elastic net was applied.

Tweedie elastic net

To incorporate that the application rate is more zero-inflated and overdispersed than a Poisson distribution, the Tweedie compound Poisson model was used. The Tweedie compound Poisson model can be represented by $Y = \sum_{i=1}^{n} X_i$, where Y is the responds vector, n a Poisson random variable, and X_i are i.i.d. Gamma distributed with parameters α and γ. The penalized negative log-likelihood is given by (6.3) [195].

$$
\begin{aligned}
PLL(\lambda, \boldsymbol{\beta}, \alpha) = & \textstyle\sum_{i=1}^{n} \left[\frac{y_i \exp\left[-(\rho-1)(\mathbf{x}_i^T \beta)\right]}{\rho-1} + \frac{\exp\left[(2-\rho)(\mathbf{x}_i^T \beta)\right]}{2-\rho} \right] \\
& + \lambda \left[(1-\alpha)\tfrac{1}{2}||\boldsymbol{\beta}||^2 + \alpha ||\boldsymbol{\beta}||_1 \right]
\end{aligned}
\tag{6.3}
$$

To minimize (6.3), the `HDTweedie` R package was used. This method applies an iterative reweighted least squares (IRLS) algorithm combined with a blockwise majorization descent (BMD) [195]. To find appropriate values for λ, the standard procedure from `HDTweedie` was used, which first computes λ_{max} such that $\beta = 0$, and then sets $\lambda_{min} = 0.001\lambda_{max}$. The other $m-2$ values for λ are found by projecting them uniformly on a log-scale on the range $[\lambda_{min}, \lambda_{max}]$. For α the values from 0.1 to 0.9 with an increase of 0.2 were used. For ρ we used $\rho = 1.5$.

Classification And Regression Trees (CART)

To construct a regression tree the `rpart` implementation in R was used [10]. This implementation first constructs a binary tree by maximizing in each node $SS_T - (SS_R + SS_L)$, where SS_T is the sum of squared error of the entire tree, and SS_R and SS_L are the sum of squared errors of the left and right branch respectively. The tree constructions stops when further splits would violate a constraint on the minimum number of observations in each node.

Second, the constructed tree is split into m sub-trees. Let $R(T)$ be the risk of tree T, which is the sum of squared error in the terminal nodes of T. CART computes the risk of each sub-tree tree, which is defined by $R_\alpha(T) = R(T) + \alpha|T|$ using K-fold cross validation. The term $\alpha|T|$ is an additional penalty on the size of the tree. The final tree is the sub-tree that minimizes the average sum of squared error over the K-fold cross validation. The method described here is referred to as the “ANOVA" method.

Alternatively, rpart also has the option to maximize the deviance $D_T - (D_L + D_R)$, where D is the within node deviance assuming that the response originates from a Poisson distribution. To find an appropriate value for α, a grid search was applied using $\alpha \in \{0.001, 0.01, 0.1, 0.3\}$, both for the ANOVA and Poisson models.

Random Forest

A Random Forest model was produced using the `RandomForest` package in R [150]. Random Forest constructs T unpruned regression trees T_i, where in each split only d randomly chosen predictors are considered. A prediction $\hat{y}_i$ is then created by $\hat{y}_i = \frac{1}{T}\sum_{i=1}^{T} T_i(x)$, thus the average over all trees. To find the appropriate number of trees, a grid search was applied using 50, 100 and 500 trees. Furthermore, at each split, 61 randomly sampled attributes were considered.

Support Vector Regression

Support Vector Regression is the regression alternative of Support Vector Machines. Given the linear regression problem: $y_i = \mathbf{w}^T\mathbf{x}_i + b + \epsilon$, SVR attempts to find the flattest hyperplane $\mathbf{w}^T\mathbf{x}_i + b$ such that for all data points $i = 1, \ldots, N$ we have $|y_i - (\mathbf{w}^T\mathbf{x}_i + b)| < \epsilon$. Also incorporating slack variables ζ_i and ζ_i^*, the problem can be described as (6.4).

$$\begin{array}{lrcll} \min & \multicolumn{4}{l}{\frac{1}{2}|\mathbf{w}||^2 + C\sum_{i=1}^{n}(\zeta_i + \zeta_i^*)} \\ \text{s.t.} & y_i - \mathbf{w}^T\mathbf{x}_i - b & \leq & \epsilon + \zeta_i, & i = 1, \ldots, N \\ & \mathbf{w}^T\mathbf{x}_i + b - y_i & \leq & \epsilon + \zeta_i^*, & i = 1, \ldots, N \\ & \zeta_i, \zeta_i^* \geq 0 & & & \end{array} \tag{6.4}$$

Since the solution to (6.4) only depends on inner products between vectors $\mathbf{x}_i$, the problem can be transformed into a higher dimension without much extra computation using kernels [211]. For the computation of the SVR, the R `kernlab` package was used [125]. Although in this study initially both a linear kernel (hence no kernel), and the RBF kernel: $\kappa(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right)$ were considered, not using a kernel had a large negative effect on the runtime and was therefore disregarded. To find appropriate values for ϵ and C a grid search was applied using: $\epsilon \in \{0.01, 0.1, 1\}$ and $C \in \{1, 10\}$

Artificial Neural Networks

In this study we considered a feed-forward Artificial Neural Network with a single hidden layer. To find the weights the `nnet` R package was used, which utilizes an L-BFGS algorithm to find the appropriate weights [155, 200]. A grid search was applied to find an appropriate number of units in the hidden layer. During the grid search 1, 5, 10, 30, and 50 hidden units were considered.

6.5.3 Method evaluation

To evaluate the quality of predictive models, two scenarios were distinguished. The first scenario assumes that application rates in weeks prior to the predicting period are known, which is comparable to predicting one week ahead. The second scenario assumes these application rates to be absent, and is more comparable to predicting 2 to 12 weeks ahead. These two scenarios are indicated by *including PAR*, and *excluding PAR*.

To evaluate the quality of the predictions, two error measures are used: the Root Mean Squared Error (RMSE), and the determination coefficient (R^2). A 10-fold cross validation was applied to obtain accurate estimates for the quality of the predictions over the training set in both scenarios. Furthermore, a final prediction over the test set was made using the model showing the best results over the training set to estimate out of sample performance.

6.6 Results

Method comparison Table 6.1 shows the best results per model when applying a 10-fold cross validation on the training set. The table indicates that Random Forest produced the best results, both when predicting with and without PAR. Table 6.1 also indicates that multiple methods such as artificial neural networks without PAR, Poisson elastic net and Tweedie elastic net with PAR did not produce accurate results. Furthermore, Table 6.1 indicates that the added value of including previous application rates into the model is relatively small. Therefore, the predictive model would only produce slightly better results when predicting short term (1 week), in comparison to prediction long term (2 to 12 weeks).

Table 6.1: Results 10-fold cross validation

Method	Best $RMSE$ including PAR	Best R^2 including PAR	Best $RMSE$ excluding PAR	Best R^2 excluding PAR
Linear elastic net	11.82	0.35	11.87	0.34
Poisson elastic net	15.53	0	NA	NA
CART ANOVA	10.72	0.46	11.12	0.42
CART Poisson	10.17	0.52	10.75	0.46
Random Forest	9.38	0.59	9.93	0.54
Tweedie elastic net	13.86	0.03	11.62	0.37
SVR	10.58	0.46	11.28	0.41
ANN	11.14	0.42	17.35	0

When considering the quality of the models indicated in Table 6.1 we note that the $RMSE$ is largely influenced by some large application rates that are difficult to predict. This can also be derived from the errors the Random Forest model makes on the test set (Figure 6.2). In fact, 90% of the errors are smaller than 9.63, and the average absolute error over this 90% is 2.43.

While predicting the application rate, also the predictive value of online marketing campaigns was considered. To determine the importance of these campaigns, the following procedure was used. For each tree of the forest, the reduction in the sum of squared error when splitting on one of the D randomly selected attributes was computed. These reductions are summed up over all trees for each variable to obtain an overall picture of the decrease in residual sum of squares per predictor.

By comparing the reduction in the sum of squared error for each variable, a comparison can be made between the effect of online marketing campaigns and other attributes. From this comparison, we found that the effect of online marketing campaigns on the the application rate is small. Most variance is explained by predictors related to the application rate in prior weeks, the job title, the current recruitment lead time, and the contractual hours required. However, in contrast to the model predicting the total number of applicants,

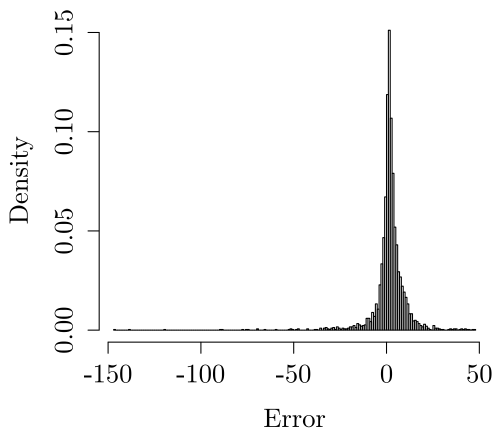


Figure 6.2: Test set error with PAR

the online marketing campaigns do show a positive effect on the application rate.

Test set evaluation Since Random Forest produced the most promising results when applying 10-fold cross validation this model was evaluated on the test set. The results are shown is Table 6.2, whereas the distribution of the error on the test set is given in Figure 6.2. The quality of the prediction was slightly worse than the average error obtained from 10-fold cross validation. Furthermore, just as the training set also the test set contained some large application rates that had a large negative effect on the $RMSE$.

6.7 Conclusion

This chapter considered the predictability of the number of weekly applications per vacancy, and to what extent this metric can be controlled. To study this question, a dataset from a large Dutch organization employing more than 30,000 employees was used. To predict the number of weekly applications per vacancy, multiple machine learning methods were applied. Random Forest returned the best results, with a root mean squared error of 9.38 and 9.93

Table 6.2. Results applying Random Forest on test set

Performance metric	Value including PAR	Value excluding PAR
MAE	5.25	6.35
MSE	100.44	123.99
RMSE	10.02	11.13
Residual mean	1.53	1.81
Residual sd	9.90	10.98
R^2	0.44	0.32

when the predictors included and excluded application rates in weeks prior to the predicted week.

From closer examination of the errors two conclusions can be drawn. First, even though the predictions are quite accurate in most situations, i.e., have an error of less than 5 applicants, some vacancies can attract a large number of job seekers that the model finds hard to predict. As a result, it will be more difficult to act on these predictions. On the other hand, both the predictions and insights into the variability of these predictions are helpful to manage the expectations recruiters and hiring managers might have when publishing a vacancy. In particular, recruiters should manage vacancies expecting a large number of applications carefully to avoid an excess of applications.

Second, from analyzing the effect of online marketing campaigns on the number of applications per vacancy per week, we found this effect to be small but positive. Also, it is likely that when estimating the effect of online marketing campaigns from historic data, this effect will be biased. Vacancies that do not attract many applications are more likely to be used in online marketing campaigns, and there might be collinearity between the campaigns. Therefore, we were not able to draw a clear conclusion on the effect of online marketing campaigns, and how this can be used to control the number of applications.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, and Google Brain. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 265–283, 2016.

[2] Lydia Abbot, Ryan Batty, and Stephanie Bevegni. *Global Recruiting Trends 2016*, 2015. URL: `https://business.linkedin.com/content/dam/business/talent-solutions/global/en_us/c/pdfs/GRT16_GlobalRecruiting_100815.pdf`, last accessed: October 2021.

[3] Fabian Abel, András Benczúr, Daniel Kohlsdorf, Martha Larson, and Róbert Pálovics. Recsys challenge 2016: Job recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 425–426. ACM, 2016.

[4] Fabian Abel, Yashar Deldjoo, Mehdi Elahi, and Daniel Kohlsdorf. RecSys challenge 2017: Offline and online evaluation. In *Proceedings of the eleventh ACM Conference on Recommender Systems*, pages 372–373, 2017.

[5] Charu C. Aggarwal. *Recommender systems.* Springer, 2016.

[6] Shibbir Ahmed, Mahamudul Hasan, Md. Nazmul Hoq, and Muhammad Abdullah Adnan. User interaction analysis to recommend suitable jobs in career-oriented social networking sites. In *2016 International*

Conference on Data and Software Engineering (ICoDSE), pages 1–6. IEEE, 2016.

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

[8] Nikolaos D Almalis, George A Tsihrintzis, and Evangelos Kyritsis. A constraint-based job recommender system integrating FoDRA. *International Journal of Computational Intelligence Studies*, 7(2):103–123, 2018.

[9] Dhruv Arya, Viet Ha-Thuc, and Shakti Sinha. Personalized federated search at Linkedin. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1699–1702, 2015.

[10] Elizabeth J. Atkinson and Terry M. Therneau. An introduction to recursive partitioning using the rpart routines. *Rochester: Mayo Foundation*, 2000.

[11] Raju Balakrishnan and Subbarao Kambhampati. Optimal ad-ranking for profit maximization. In *Proceedings of the 11th International Workshop on the Web and Databases*. ACM, 2008.

[12] Jack Bandy. Problematic machine behavior: A systematic literature review of algorithm audits. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW1):1–34, 2021.

[13] Shivam Bansal, Aman Srivastava, and Anuja Arora. Topic modeling driven content based jobs recommendation engine for recruitment industry. *Procedia computer science*, 122:865–872, 2017.

[14] Mathieu Bastian, Matthew Hayes, William Vaughan, Sam Shah, Peter Skomoroch, Hyungjin Kim, Sal Uryasev, and Christopher Lloyd. LinkedIn skills: large-scale topic extraction and inference. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 1–8, 2014.

[15] Zeynep Batmaz, Ali Yurekli, Alper Bilge, and Cihan Kaleli. A review on deep learning for recommender systems: challenges and remedies. *Artificial Intelligence Review*, 52(1):1–37, 2019.

[16] Yoshua Bengio and Paolo Frasconi. An input output hmm architecture. In *Advances in neural information processing systems (NIPS)*, pages 427–434, 1995.

[17] Alan Mark Berg. The learning analytics architectural lifecycle. 2018.

[18] Shuqing Bian, Wayne Xin Zhao, Yang Song, Tao Zhang, and Ji-Rong Wen. Domain adaptation for person-job fit with transferable deep global match network. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4812–4822, 2019.

[19] Shuqing Bian, Xu Chen, Wayne Xin Zhao, Kun Zhou, Yupeng Hou, Yang Song, Tao Zhang, and Ji-Rong Wen. Learning to match jobs with resumes from sparse interaction data using multi-view co-teaching network. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 65–74, 2020.

[20] Mattia Bianchi, Federico Cesaro, Filippo Ciceri, Mattia Dagrada, Alberto Gasparin, Daniele Grattarola, Ilyas Inajjar, Alberto Maria Metelli, and Leonardo Cella. Content-based approaches for cold-start job recommendations. In *RecSys Challenge '17: Proceedings of the Recommender Systems Challenge 2017*, pages 1–5. 2017.

[21] Christopher M. Bishop. *Pattern recognition and machine learning.* Springer, 2006.

[22] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. A neural click model for web search. In *Proceedings of the 25th International Conference on World Wide Web*, pages 531–541, 2016.

[23] Alexey Borisov, Martijn Wardenaar, Ilya Markov, and Maarten de Rijke. A click sequence model for web search. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 45–54, 2018.

[24] Fedor Borisyuk, Liang Zhang, and Krishnaram Kenthapadi. LiJAR: A system for job application redistribution towards efficient career marketplace. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1397–1406, 2017.

[25] Paul Boselie. *Strategic human resource management: A balanced approach.* McGraw-Hill Education, 2010.

[26] Svetlin Bostandjiev, John O'Donovan, and Tobias Höllerer. LinkedVis: exploring social and semantic career recommendations. In *Proceedings*

of the 2013 international conference on Intelligent user interfaces, pages 107–116, 2013.

[27] Milan Bouchet-Valat. Package ‘SnowballC’. `https://cran.r-project.org/web/packages/SnowballC/index.html`, 2019. retrieved: September 2021.

[28] Phillip W Braddy, Adam W Meade, and Christina M Kroustalis. Organizational recruitment website effects on viewers’ perceptions of organizational culture. *Journal of Business and Psychology*, 20(4):525–543, 2006.

[29] David Brown. Unwanted online job seekers swamp HR staff. *Canadian HR reporter*, 17(7):1–2, 2004.

[30] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[31] Ricardo J.G.B. Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 160–172. Springer, 2013.

[32] Ricardo J.G.B. Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(1):1–51, 2015.

[33] Michal Campr and Karel Ježek. Comparing semantic models for evaluating automatic document summarization. In *International Conference on Text, Speech, and Dialogue*, pages 252–260. Springer, 2015.

[34] CareerBuilder. *Website CareerBuilder*, 2021. Retrieved from: `https://www.careerbuilder.com`, last accessed: October 2021.

[35] Tommaso Carpi, Marco Edemanti, Ervin Kamberoski, Elena Sacchi, Paolo Cremonesi, Roberto Pagano, and Massimo Quadrana. Multi-stack ensemble for job recommendation. In *RecSys Challenge ’16: Proceedings of the Recommender Systems Challenge*, pages 1–4. 2016.

[36] Wayne F. Cascio and Herman Aguinis. *Applied psychology in human resource management.* Prentice Hall, 1998.

[37] Wayne F. Cascio and John Boudreau. *Investing in people: Financial impact of human resource initiatives.* FT Press, 2010.

[38] CBS. *Maatwerk - Banen van werknemers naar baanduur*, 2018. Retrieved from: `https://cbs.nl/nl-nl/maatwerk/2018/17/maatwerk-banen-van-werknemers-naar-baanduur`, last accessed: October 2021.

[39] Duygu Çelik and Atilla Elçi. An ontology-based information extraction approach for résumés. In *Joint International Conference on Pervasive Computing and the Networked World*, pages 165–179. Springer, 2012.

[40] Olivier Chapelle and Ya Zhang. A dynamic bayesian network click model for web search ranking. In *Proceedings of the 18th international conference on World wide web*, pages 1–10. ACM, 2009.

[41] Le Chen, Ruijun Ma, Anikó Hannák, and Christo Wilson. Investigating the impact of gender on rank in resume search engines. In *Proceedings of the 2018 chi conference on human factors in computing systems*, pages 1–14, 2018.

[42] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[43] Weijian Chen, Xingming Zhang, Haoxiang Wang, and Hongjie Xu. Hybrid deep collaborative filtering for job recommendation. In *2017 2nd IEEE International Conference on Computational Intelligence and Applications (ICCIA)*, pages 275–280. IEEE, 2017.

[44] Wenbo Chen, Pan Zhou, Shaokang Dong, Shimin Gong, Menglan Hu, Kehao Wang, and Dapeng Wu. Tree-based contextual learning for online job or candidate recommendation with big data support in professional social networks. *IEEE Access*, 6:77725–77739, 2018.

[45] Oualid Chenni, Yanis Bouda, Hamid Benachour, and Chahnez Zakaria. A content-based recommendation approach using semantic user profile in e-recruitment. In *International Conference on Theory and Practice of Natural Computing*, pages 23–32. Springer, 2015.

[46] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[47] Andrew Cholakian. *A Gentle Intro to Function Scoring*, 2014. Retrieved from: `https://www.elastic.co/blog/found-function-scoring`, last accessed: October 2021.

[48] Yi-Chi Chou and Han-Yen Yu. Based on the application of AI technology in resume analysis and job recommendation. In *2020 IEEE International Conference on Computational Electromagnetics (ICCEM)*, pages 291–296. IEEE, 2020.

[49] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. Click models for web search. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 7(3):1–115, 2015.

[50] CIKM. *CIKM Cup 2016 Track 1: Cross-Device Entity Linking Challenge*, 2016. Retrieved from: `https://competitions.codalab.org/competitions/11171`, last accessed: October 2021.

[51] Clickmodels. *Clickmodels Github repository.* `https://github.com/varepsilon/clickmodels`, last accessed: October 2021.

[52] Dominic Coey and Michael Bailey. People and cookies: Imperfect treatment assignment in online experiments. In *Proceedings of the 25th International Conference on World Wide Web*, pages 1103–1111. International World Wide Web Conferences Steering Committee, 2016.

[53] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 international conference on web search and data mining*, pages 87–94, 2008.

[54] Françoise Dany and Véronique Torchy. Recruitment and selection in Europe policies, practices and methods 1. In *Policy and practice in European human resource management*, pages 68–88. Routledge, 2017.

[55] Anirban Dasgupta, Maxim Gurevich, Liang Zhang, Belle Tseng, and Achint O. Thomas. Overcoming browser cookie churn with clustering. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 83–92. ACM, 2012.

[56] Vachik S. Dave, Baichuan Zhang, Mohammad Al Hasan, Khalifeh Al-Jadda, and Mohammed Korayem. A combined representation learning approach for better job and skill recommendation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1997–2005, 2018.

[57] Toon De Pessemier, Kris Vanhecke, and Luc Martens. A scalable, high-performance algorithm for hybrid job recommendations. In *RecSys Chal-*

lenge '16: Proceedings of the Recommender Systems Challenge, pages 1–4. 2016.

[58] Corné de Ruijt and Bhulai. Detecting users from website sessions: A simulation study and results on multiple simulation scenarios. *International Journal On Advances in Internet Technology 14, Number 1 & 2, 2021 (forthcoming)*, 2021.

[59] Corné de Ruijt and Sandjai Bhulai. Detecting users from website sessions: A simulation study. In *DATA ANALYTICS 2020, The Ninth International Conference on Data Analytics*, pages 35–40, 2020.

[60] Corné de Ruijt and Sandjai Bhulai. *GCM Github repository*, 2021. `https://github.com/cornederuijtnw/GCM`, last accessed: October, 2021.

[61] Corné de Ruijt and Sandjai Bhulai. *gecasmo package*, 2021. `https://pypi.org/project/gecasmo/`, last accessed: October, 2021.

[62] Corné de Ruijt and Sandjai Bhulai. The generalized cascade click model: A unified framework for estimating click models. *arXiv preprint arXiv:2111.11314*, 2021.

[63] Corné de Ruijt and Sandjai Bhulai. Job recommender systems: A review. *arXiv preprint arXiv:2111.13576*, 2021.

[64] Corné de Ruijt and Sandjai Bhulai. A comparison of machine-learned survival models for predicting tenure from unstructured resumes. In *DATA ANALYTICS 2021, The Tenth International Conference on Data Analytics*, 2021. Forthcoming.

[65] Corné de Ruijt, Sandjai Bhulai, Han Rusman, and Leon Willemsens. Predicting candidate uptake for online vacancies. *DATA ANALYTICS 2016, The Fifth International Conference on Data Analytics*, pages 63–68, 2016.

[66] Corné de Ruijt, Sandjai Bhulai, Bram L. Gorissen, Han Rusman, and Leon Willemsens. Predicting candidate uptake on individual online vacancies and vacancy portfolios. *International Journal on Advances in Software Volume 10, Number 1 & 2, 2017*, 2017.

[67] Weiwei Deng, Xiaoliang Ling, Yang Qi, Tunzi Tan, Eren Manavoglu, and Qi Zhang. Ad click prediction in sequence with long short-term memory networks: an externality-aware model. In *The 41st International ACM*

SIGIR Conference on Research & Development in Information Retrieval, pages 1065–1068, 2018.

[68] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[69] Mamadou Diaby and Emmanuel Viennet. Taxonomy-based job recommender systems on Facebook and LinkedIn profiles. In *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*, pages 1–6. IEEE, 2014.

[70] Mamadou Diaby, Emmanuel Viennet, and Tristan Launay. Toward the next generation of recruitment tools: an online social network-based job recommender system. In *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*, pages 821–828. IEEE, 2013.

[71] Mamadou Diaby, Emmanuel Viennet, and Tristan Launay. Exploration of methodologies to improve job recommender systems on social networks. *Social Network Analysis and Mining*, 4(1):227, 2014.

[72] Roberto Díaz-Morales. Cross-device tracking: Matching devices and cookies. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 1699–1704. IEEE, 2015.

[73] Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, Karin Pasini, and Roberto Pasolini. Job recommendation from semantic similarity of LinkedIn users' skills. In *International Conference on Pattern Recognition Applications and Methods (ICPRAM)*, pages 270–277, 2016.

[74] Shaokang Dong, Zijian Lei, Pan Zhou, Kaigui Bian, and Guanghui Liu. Job and candidate recommendation with big data support: A contextual online learning approach. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–7. IEEE, 2017.

[75] Georges Dupret and Mounia Lalmas. Absence time and user engagement: evaluating ranking functions. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 173–182. ACM, 2013.

[76] Georges E. Dupret and Benjamin Piwowarski. A user browsing model to predict search engine click data from past observations. In *Proceedings of*

the 31st annual international ACM SIGIR conference on Research and development in information retrieval, pages 331–338. ACM, 2008.

[77] ELISE. *Website ELISE Job Matching Search and Match Platform*, 2021. Retrieved from: `https://www.wcc-group.com/employment/products/elise-job-matching-search-and-match-platform/`, last accessed: October 2021.

[78] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[79] Daniel Falbel, JJ Allaire, Françios Chollet, RStudio, Google, Yuan Tang, Wouter Van Der Bijl, Martin Studer, and Sigrid Keydana. *Package 'keras'*, 2019. URL `https://keras.rstudio.com`. R package version 2.2.4.1.

[80] Evanthia Faliagka, Lazaros Iliadis, Ioannis Karydis, Maria Rigou, Spyros Sioutas, Athanasios Tsakalidis, and Giannis Tzimas. On-line consistent ranking on e-recruitment: seeking the truth behind a well-formed cv. *Artificial Intelligence Review*, 42(3):515–528, 2014.

[81] Alexander Felfernig, Michael Jeran, Gerald Ninaus, Florian Reinfrank, and Stefan Reiterer. Toward the next generation of recommender systems: applications and research challenges. In *Multimedia services in intelligent environments*, pages 81–98. Springer, 2013.

[82] Peter Flach. *Machine learning: the art and science of algorithms that make sense of data.* Cambridge University Press, 2012.

[83] Daniel Fleder and Kartik Hosanagar. Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity. *Management science*, 55(5):697–712, 2009.

[84] Mauricio Noris Freire and Leandro Nunes de Castro. e-recruitment recommender systems: a systematic review. *Knowledge and Information Systems*, pages 1–20, 2020.

[85] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[86] Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[87] Michael F Gensheimer and Balasubramanian Narasimhan. A scalable discrete-time survival model for neural networks. *PeerJ*, 7:e6257, 2019. URL `https://doi.org/10.7717/peerj.6257`.

[88] Sahin Cem Geyik, Stuart Ambler, and Krishnaram Kenthapadi. Fairness-aware ranking in search & recommendation systems with application to linkedin talent search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2221–2231, 2019.

[89] Philip Gibson and Jennifer Swift. e2c: Maximising electronic resources for cruise recruitment. *Journal of Hospitality and Tourism Management*, 18(01):61–69, 2011.

[90] R Greenspan. *Job seekers have choices*, 2003. URL: `https://www.clickz.com/job-seekers-have-choices/76679/` last accessed: October 2021.

[91] Brandon Greenwell, Bradley Boehmke, and Jay Cunningham. *Package 'gbm'*, 2019. URL `https://github.com/gbm-developers/gbm`. R package version 2.1.5.

[92] Akshay Gugnani and Hemant Misra. Implicit skills extraction using document embedding and its use in job recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13286–13293, 2020.

[93] Huan Gui, Haishan Liu, Xiangrui Meng, Anmol Bhasin, and Jiawei Han. Downside management in recommender systems. In *Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on*, pages 394–401. IEEE, 2016.

[94] Cheng Guo, Hongyu Lu, Shaoyun Shi, Bin Hao, Bin Liu, Min Zhang, Yiqun Liu, and Shaoping Ma. How integration helps on cold-start recommendations. In *RecSys Challenge '17: Proceedings of the Recommender Systems Challenge 2017*, pages 1–6. 2017.

[95] Fan Guo, Chao Liu, and Yi Min Wang. Efficient multiple-click models in web search. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 124–131, 2009.

[96] Xingsheng Guo, Houssem Jerbi, and Michael P. O'Mahony. An analysis framework for content-based job recommendation. In *22nd International Conference on Case-Based Reasoning (ICCBR)*, 2014.

[97] Anika Gupta and Deepak Garg. Applying data mining techniques in job recommender system for considering candidate job preferences. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1458–1465. IEEE, 2014.

[98] Gus. *Website Gus*, 2017. Retrieved from: `https://www.gus.nl`, last accessed: October 2021.

[99] Francisco Gutiérrez, Sven Charleer, Robin De Croon, Nyi Nyi Htun, Gerd Goetschalckx, and Katrien Verbert. Explaining and exploring job recommendations: a user-driven approach for interacting with knowledge-based job recommender systems. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 60–68, 2019.

[100] Rob Hartgers. *Kan een computerprogramma uit duizenden sollicitanten de beste kandidaten selecteren? Unilever denkt van wel.*, 2019. Retrieved from: `https://www.nu.nl/werk/5972368`, last accessed: October 2021.

[101] Christopher J Hartwell. *The use of social media in employee selection: Prevalence, content, perceived usefulness, and influence on hiring decisions.* PhD thesis, Purdue University, 2015.

[102] Trevor Hastie and Junyang Qian. *Glmnet Vignette*, 2014. URL: `http://www.web.stanford.edu/~hastie/Papers/Glmnet_Vignette.pdf`, last accessed: October 2021.

[103] Bradford Heap, Alfred Krzywicki, Wayne Wobcke, Mike Bain, and Paul Compton. Combining career progression and profile matching in a job recommender system. In *Pacific Rim International Conference on Artificial Intelligence*, pages 396–408. Springer, 2014.

[104] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[105] Peter W Hom, Thomas W Lee, Jason D Shaw, and John P Hausknecht. One hundred years of employee turnover theory and research. *Journal of Applied Psychology*, 102(3):530–545, 2017.

[106] Wenxing Hong, Siting Zheng, Huan Wang, and Jianchao Shi. A job recommender system based on user clustering. *Journal of Computers (JCP)*, 8(8):1960–1967, 2013.

[107] Jose Ignacio Honrado, Oscar Huarte, Cesar Jimenez, Sebastian Ortega, Jose R. Perez-Aguera, Joaquin Perez-Iglesias, Alvaro Polo, and Gabriel Rodriguez. Jobandtalent at RecSys challenge 2016. In *RecSys Challenge*

'16: Proceedings of the Recommender Systems Challenge, pages 1–5. 2016.

[108] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[109] ICDM. *ICDM 2015: Drawbridge Cross-Device Connections*, 2015. Retrieved from: `https://www.kaggle.com/c/icdm-2015-drawbridge-cross-device-connections`, last accessed: October 2021.

[110] Hemant Ishwaran and Udaya B Kogalur. *Random Forests for Survival, Regression, and Classification (RF-SRC)*, 2018. URL `https://cran.r-project.org/package=randomForestSRC`. R package version 2.6.0.

[111] Hemant Ishwaran, Udaya B Kogalur, Eugene H Blackstone, and Michael S Lauer. Random survival forests. *The Annals of Applied Statistics*, pages 841–860, 2008.

[112] Leila Jalali, Misbah Khan, and Rahul Biswas. Learning and multi-objective optimization for automatic identity linkage. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4926–4931. IEEE, 2018.

[113] Dietmar Jannach, Gabriel de Souza P. Moreira, and Even Oldridge. Why are deep learning models not consistently winning recommender systems competitions yet? a position paper. *RecSys Challenge'20, Proceedings of the Recommender Systems Challenge 2020*, page 44–49, 2020.

[114] Bernard J. Jansen, Karen J. Jansen, and Amanda Spink. Using the web to look for work: Implications for online job seeking and recruiting. *Internet research*, 15(1):49–66, 2005.

[115] Andrzej Janusz, Sebastian Stawicki, Michał Drewniak, Krzysztof Ciebiera, Dominik Ślęzak, and Krzysztof Stencel. How to match jobs and candidates - a recruitment support system based on feature engineering and advanced analytics. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 503–514. Springer, 2018.

[116] Stephen Jenkins. *Survival Analysis*, 2005.

[117] Junshu Jiang, Songyun Ye, Wei Wang, Jingran Xu, and Xiaosheng Luo. Learning effective representations for person-job fit by feature fusion. In

Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pages 2549–2556, 2020.

[118] Miao Jiang, Yi Fang, Huangming Xie, Jike Chong, and Meng Meng. User click prediction for personalized job recommendation. *World Wide Web*, 22(1):325–345, 2019.

[119] Di Jin, Mark Heimann, Ryan Rossi, and Danai Koutra. node2bits: Compact time-and attribute-aware node representations. In *ECML/PKDD European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2019.

[120] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the ACM Conference on Research and Development on Information Retrieval (SIGIR)*, pages 154–161. ACM, 2005.

[121] Anne Jonker, Corné de Ruijt, and Jornt de Gruijl. Bag & tag'em-a new dutch stemmer. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 3868–3876, 2020.

[122] Kaggle. *Careerbuilder Job Recommendation Challenge*, 2012. Retrieved from: `https://www.kaggle.com/c/job-recommendation`, last accessed: October 2021.

[123] Kaggle. *Kaggle website*, 2021. Retrieved from: `https://www.kaggle.com`, last accessed: October 2021.

[124] Can Karakaya, Hakan Toğuç, Rıdvan Salih Kuzu, and Ali Hakan Büyüklü. Survey of cross device matching approaches with a case study on a novel database. In *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, pages 139–144. IEEE, 2018.

[125] Alexandros Karatzoglou, Alex Smola, and Kurt Hornik. *The kernlab package*, 2007. URL: `https://cran.r-project.org/web/packages/kernlab/kernlab.pdf`, last accessed: October 2021.

[126] Jared Katzman, Uri Shaham, Alexander Cloninger, Jonathan Bates, Tingting Jiang, and Yuval Kluger. Deep survival: A deep cox proportional hazards network. *ArXiv*, abs/1606.00931, 2016.

[127] Krishnaram Kenthapadi, Benjamin Le, and Ganesh 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, 2017.

[128] Rémy Kessler, Nicolas Béchet, Mathieu Roche, Juan-Manuel Torres-Moreno, and Marc El-Bèze. A hybrid approach to managing job offers and candidates. *Information processing & management*, 48(6):1124–1135, 2012.

[129] Sungchul Kim, Nikhil Kini, Jay Pujara, Eunyee Koh, and Lise Getoor. Probabilistic visitor stitching on cross-device web logs. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1581–1589. International World Wide Web Conferences Steering Committee, 2017.

[130] Aseel B. Kmail, Mohammed Maree, and Mohammed Belkhatir. MatchingSem: Online recruitment system based on multiple semantic resources. In *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pages 2654–2659. IEEE, 2015.

[131] Aseel B. Kmail, Mohammed Maree, Mohammed Belkhatir, and Saadat M. Alhashmi. An automatic online recruitment system based on exploiting multiple semantic resources and concept-relatedness measures. In *27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 620–627. IEEE, 2015.

[132] Tanja Koch, Charlene Gerber, and Jeremias J. De Klerk. The impact of social media on recruitment: Are you linkedin? *SA Journal of Human Resource Management*, 16(1), 2018.

[133] Sunil Kumar Kopparapu. Automatic extraction of usable information from unstructured resumes to aid search. In *2010 IEEE International Conference on Progress in Informatics and Computing*, volume 1, pages 99–103. IEEE, 2010.

[134] Max Kuhn. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5):1–26, 2008.

[135] Emanuel Lacic, Markus Reiter-Haas, Tomislav Duricic, Valentin Slawicek, and Elisabeth Lex. 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, 2019.

[136] Emanuel Lacic, Markus Reiter-Haas, Dominik Kowald, Manoj Reddy Dareddy, Junghoo Cho, and Elisabeth Lex. Using autoencoders for session-based job recommendations. *User Modeling and User-Adapted Interaction*, 30(4):617–658, 2020.

[137] Sven Laumer, Fabian Gubler, Christian Maier, and Tim Weitzel. Job seekers' acceptance of job recommender systems: Results of an empirical study. In *Proceedings of the 51st Hawaii International Conference on System Sciences*, pages 3914–3923, 2018.

[138] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

[139] Ran Le, Wenpeng Hu, Yang Song, Tao Zhang, Dongyan Zhao, and Rui Yan. Towards effective and interpretable person-job fitting. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1883–1892, 2019.

[140] In Lee. The evolution of e-recruiting: A content analysis of fortune 100 career web sites. *Journal of Electronic Commerce in Organizations (JECO)*, 3(3):57–68, 2005.

[141] Yeon-Chang Lee, Jiwon Hong, and Sang-Wook Kim. Job recommendation in askstory: experiences, methods, and evaluation. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 780–786, 2016.

[142] Yujin Lee, Yeon-Chang Lee, Jiwon Hong, and Sang-Wook Kim. Exploiting job transition patterns for effective job recommendation. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2414–2419. IEEE, 2017.

[143] Vasily Leksin and Andrey Ostapets. Job recommendation based on factorization machine and topic modelling. In *RecSys Challenge '16: Proceedings of the Recommender Systems Challenge*, pages 1–4. 2016.

[144] Vasily Leksin, Andrey Ostapets, Mikhail Kamenshikov, Dmitry Khodakov, and Vasily Rubtsov. Combination of content-based user profiling and local collective embeddings for job recommendation. In *CEUR Workshop Proceeding, Vol. 1968, No. Experimental Economics and Machine Learning*, pages 9–17, 2017.

[145] Huayu Li, Yong Ge, Hengshu Zhu, Hui Xiong, and Hongke Zhao. Prospecting the career development of talents: A survival analysis perspective. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 917–925. ACM, 2017.

[146] Jia Li, Dhruv Arya, Viet Ha-Thuc, and Shakti Sinha. How to get them a dream job? entity-aware features for personalized job search ranking. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 501–510, 2016.

[147] Liangyue Li, How Jing, Hanghang Tong, Jaewon Yang, Qi He, and Bee-Chung Chen. NEMO: Next career move prediction with contextual embedding. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 505–513. International World Wide Web Conferences Steering Committee, 2017.

[148] Jianxun Lian and Xing Xie. Cross-device user matching based on massive browse logs: The runner-up solution for the 2016 CIKM cup. *arXiv preprint arXiv:1610.03928*, 2016.

[149] Jianxun Lian, Fuzheng Zhang, Min Hou, Hongwei Wang, Xing Xie, and Guangzhong Sun. Practical lessons for job recommendations in the cold-start scenario. In *RecSys Challenge '17: Proceedings of the Recommender Systems Challenge 2017*, pages 1–6. 2017.

[150] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[151] Cynthia C.S. Liem, Markus Langer, Andrew Demetriou, Annemarie M.F. Hiemstra, Achmadnoer Sukma Wicaksana, Marise Ph. Born, and Cornelius J. König. Psychology meets machine learning: Interdisciplinary perspectives on algorithmic job candidate screening. In *Explainable and Interpretable Models in Computer Vision and Machine Learning*, pages 197–253. Springer, 2018.

[152] D.Y. Lin. On the breslow estimator. *Lifetime data analysis*, 13(4):471–480, 2007.

[153] Yiou Lin, Hang Lei, Prince Clement Addo, and Xiaoyu Li. Machine learned resume-job matching solution. *arXiv preprint arXiv:1607.07657*, 2016.

[154] LinkedIn. *About LinkedIn*, 2021. Retrieved from: `https://about.linkedin.com`, last accessed: October 2021.

[155] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[156] Kuan Liu, Xing Shi, Anoop Kumar, Linhong Zhu, and Prem Natarajan. Temporal learning and sequence modeling for a job recommender system. In *RecSys Challenge '16: Proceedings of the Recommender Systems Challenge*, pages 1–4. 2016.

[157] Miao Liu, Zijie Zeng, Weike Pan, Xiaogang Peng, Zhiguang Shan, and Zhong Ming. Hybrid one-class collaborative filtering for job recommendation. In *International Conference on Smart Computing and Communication*, pages 267–276. Springer, 2016.

[158] Yiqun Liu, Xiaohui Xie, Chao Wang, Jian-Yun Nie, Min Zhang, and Shaoping Ma. Time-aware click model. *ACM Transactions on Information Systems (TOIS)*, 35(3), 2016.

[159] Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. Recommender system application developments: a survey. *Decision Support Systems*, 74:12–32, 2015.

[160] Yao Lu, Sandy El Helou, and Denis Gillet. A recommender system for job seeking and recruiting website. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 963–966, 2013.

[161] Malte Ludewig, Michael Jugovac, and Dietmar Jannach. A light-weight approach to recipient determination when recommending new items. In *RecSys Challenge '17: Proceedings of the Recommender Systems Challenge 2017*, pages 1–6. 2017.

[162] Cuauhtemoc Luna-Nevarez and Michael R. Hyman. Common practices in destination website design. *Journal of destination marketing & management*, 1(1-2):94–106, 2012.

[163] Yong Luo, Huaizheng Zhang, Yonggang Wen, and Xinwen Zhang. ResumeGAN: An optimized deep representation learning framework for talent-job fit via adversarial learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1101–1110, 2019.

[164] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(86):2579–2605, 2008.

[165] Saket Maheshwary and Hemant Misra. Matching resumes to jobs via deep siamese network. In *Companion Proceedings of the The Web Conference 2018*, pages 87–88, 2018.

[166] Emmanuel Malherbe, Mamadou Diaby, Mario Cataldi, Emmanuel Viennet, and Marie-Aude Aufaure. Field selection for job categorization and recommendation to social network users. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, pages 588–595. IEEE, 2014.

[167] Jochen Malinowski, Tobias Keim, Oliver Wendt, and Tim Weitzel. Matching people and jobs: A bilateral recommendation approach. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, volume 6, pages 137c–137c. IEEE, 2006.

[168] Jorge Martinez-Gil, Alejandra Lorena Paoletti, and Mario Pichler. A novel approach for learning how to automatically match job offers and candidate profiles. *Information Systems Frontiers*, pages 1–10, 2019.

[169] Annet Maseland. Wie gaat er eerst? *Flexmarkt, februari 2018*, pages 24–27, 2018.

[170] Steven D Maurer and Yuping Liu. Developing effective e-recruiting websites: Insights for managers from marketers. *Business horizons*, 50(4): 305–314, 2007.

[171] Leland McInnes and John Healy. Accelerated hierarchical density clustering. *arXiv preprint arXiv:1705.07321*, 2017.

[172] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2(11), mar 2017. doi: 10.21105/joss.00205. URL `https://doi.org/10.21105%2Fjoss.00205`.

[173] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[174] Sonu K. Mishra and Manoj Reddy. A bottom-up approach to job recommendation system. In *RecSys Challenge '16: Proceedings of the Recommender Systems Challenge*, pages 1–4. 2016.

[175] Ulla B. Mogensen, Hemant Ishwaran, and Thomas A. Gerds. Evaluating random forests for survival analysis using prediction error curves. *Journal of Statistical Software*, 50(11):1–23, 2012.

[176] George D. Montanez, Ryen W. White, and Xiao Huang. Cross-device search. In *Proceedings of the 23rd ACM International Conference on*

Conference on Information and Knowledge Management, pages 1669–1678. ACM, 2014.

[177] Motebang Daniel Mpela and Tranos Zuva. A mobile proximity job employment recommender system. In *2020 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*, pages 1–6. IEEE, 2020.

[178] Kevin P. Murphy and Stuart Russell. *Dynamic bayesian networks: representation, inference and learning.* PhD thesis, 2002.

[179] F. M. Naini, J. Unnikrishnan, P. Thiran, and M. Vetterli. Where you are is who you are: User identification by matching statistics. *IEEE Transactions on Information Forensics and Security*, 11(2):358–372, 2016.

[180] Amber Nigam, Aakash Roy, Hartaran Singh, and Harsimran Waila. Job recommendation through progression of job selection. In *2019 IEEE 6th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pages 212–216. IEEE, 2019.

[181] Lukasz Olejnik, Claude Castelluccia, and Artur Janc. On the uniqueness of web browsing history patterns. *annals of telecommunications-annales des télécommunications*, 69(1-2):63–74, 2014.

[182] Cathy O'neil. *Weapons of math destruction: How big data increases inequality and threatens democracy.* Crown, 2016.

[183] O*Net. *O*Net*, 2021. Retrieved from: `https://www.onetonline.org`, last accessed: October 2021.

[184] Andrzej Pacuk, Piotr Sankowski, Karol Węgrzycki, Adam Witkowski, and Piotr Wygocki. RecSys challenge 2016: job recommendations based on preselection of offers and gradient boosting. In *RecSys Challenge '16: Proceedings of the Recommender Systems Challenge*, pages 1–4. 2016.

[185] Ioannis Paparrizos, B. Barla Cambazoglu, and Aristides Gionis. Machine learned job recommendation. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 325–328. ACM, 2011.

[186] Emma Parry and Shaun Tyson. An analysis of the use and success of online recruitment methods in the uk. *Human Resource Management Journal*, 18(3):257–274, 2008.

[187] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vander-

plas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[188] Caroline Criado Perez. *Invisible women: Exposing data bias in a world designed for men.* Random House, 2019.

[189] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (long papers)*, pages 2227–2237, 2018.

[190] Minh C. Phan, Yi Tay, and Tuan-Anh Nguyen Pham. Cross device matching for online advertising with neural feature ensembles: First place solution at CIKM cup 2016, 2016.

[191] Minh C. Phan, Aixin Sun, and Yi Tay. Cross-device user linking: URL, session, visiting time, and device-log embedding. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 933–936. ACM, 2017.

[192] Marc Poch, Núria Bel Rafecas, Sergio Espeja, and Felipe Navio. Ranking job offers for candidates: learning hidden knowledge from big data. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014).* ACL (Association for Computational Linguistics), 2014.

[193] Mirko Polato and Fabio Aiolli. A preliminary study on a recommender system for the job recommendation challenge. In *RecSys Challenge '16: Proceedings of the Recommender Systems Challenge*, pages 1–4. 2016.

[194] PyClick. *PyClick Github repository.* `https://github.com/markovi/PyClick`, last accessed: October 2021.

[195] Wei Qian, Yi Yang, and Hui Zou. Tweedie's compound poisson model with grouped elastic net. *Journal of Computational and Graphical Statistics*, 25(2):606–625, 2016.

[196] Chuan Qin, Hengshu Zhu, Tong Xu, Chen Zhu, Liang Jiang, Enhong Chen, and Hui 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, 2018.

[197] Chuan Qin, Hengshu Zhu, Tong Xu, Chen Zhu, Chao Ma, Enhong Chen, and Hui Xiong. An enhanced neural network approach to person-job fit in talent recruitment. *ACM Transactions on Information Systems (TOIS)*, 38(2):1–33, 2020.

[198] Michael Reusens, Wilfried Lemahieu, Bart Baesens, and Luc Sels. A note on explicit versus implicit information for job recommendation. *Decision Support Systems*, 98:26–35, 2017.

[199] Michael Reusens, Wilfried Lemahieu, Bart Baesens, and Luc Sels. Evaluating recommendation and search in the labor market. *Knowledge-Based Systems*, 152:62–69, 2018.

[200] Brian Ripley and William Venables. *Package 'nnet'*, 2016. URL: `https://cran.r-project.org/web/packages/nnet/nnet.pdf`, last accessed: October 2021.

[201] Alberto Rivas, Pablo Chamoso, Alfonso González-Briones, Roberto Casado-Vara, and Juan Manuel Corchado. Hybrid job offer recommender system in a social network. *Expert Systems*, 36(4):e12416, 2019.

[202] Jean-François Rouet, Andrew Dillon, Jarmo J Levonen, and Rand J Spiro. *Hypertext and cognition.* Psychology Press, 1996.

[203] Rishiraj Saha Roy, Ritwik Sinha, Niyati Chhaya, and Shiv Saini. Probabilistic deduplication of anonymous web traffic. In *Proceedings of the 24th International Conference on World Wide Web*, pages 103–104. ACM, 2015.

[204] Masahiro Sato, Koki Nagatani, and Takuji Tahara. Exploring an optimal online model for new job recommendation: Solution for RecSys challenge 2017. In *RecSys Challenge '17: Proceedings of the Recommender Systems Challenge 2017*, pages 1–5. 2017.

[205] Thomas Schmitt, Philippe Caillou, and Michele Sebag. Matching jobs and resumes: a deep collaborative filtering task. In *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, volume 41, 2016.

[206] Thomas Schmitt, François Gonard, Philippe Caillou, and Michèle Sebag. Language modelling for collaborative filtering: Application to job applicant matching. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1226–1233. IEEE, 2017.

[207] Walid Shalaby, BahaaEddin AlAila, Mohammed Korayem, Layla Pournajaf, Khalifeh AlJadda, Shannon Quinn, and Wlodek Zadrozny. Help me find a job: A graph-based approach for job recommendation at scale. In *2017 IEEE international conference on big data (big data)*, pages 1544–1553. IEEE, 2017.

[208] Saman Shishehchi and Seyed Yashar Banihashem. Jrdp: a job recommender system based on ontology for disabled people. *International Journal of Technology and Human Interaction (IJTHI)*, 15(1):85–99, 2019.

[209] Noah Simon, Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for Cox's proportional hazards model via coordinate descent. *Journal of Statistical Software*, 39(5):1–13, 2011.

[210] Zheng Siting, Hong Wenxing, Zhang Ning, and Yang Fan. Job recommender systems: a survey. In *2012 7th International Conference on Computer Science & Education (ICCSE)*, pages 920–924. IEEE, 2012.

[211] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

[212] LinkedIn Talent Solutions. *LinkedIn 2020 Global Talent Trends*, 2020. Retrieved from: `https://business.linkedin.com/talent-solutions/recruiting-tips/global-talent-trends-2020`, last accessed: October 2021.

[213] Rongwei Song, Siding Chen, Bailong Deng, and Li Li. eXtreme gradient boosting for identifying individual users across different digital devices. In *International Conference on Web-Age Information Management*, pages 43–54. Springer, 2016.

[214] Stefan Strohmeier and Franca Piazza. Domain driven data mining in human resource management: A review of current research. *Expert Systems with Applications*, 40(7):2410–2420, 2013.

[215] Studo. *Website Studo*, 2021. Retrieved from: `https://studo.com/at`, last accessed: October 2021. Redirected from `https://studo.co`.

[216] Farrukh Suvankulov. Job search on the internet, e-recruitment, and labor market outcomes. Technical report, RAND CORP SANTA MONICA CA, 2010.

[217] Ugo Tanielian, Anne-Marie Tousch, and Flavian Vasile. Siamese cookie

embedding networks for cross-device user matching. In *Companion Proceedings of the The Web Conference 2018*, pages 85–86. ACM, 2018.

[218] Textkernel. *Website Textkernel*, 2017. Retrieved from: `https://www.textkernel.com/`, last accessed: October 2021.

[219] textkernel. *Website Textkernel*, 2021. Retrieved from: `https://www.textkernel.com`, last accessed: October 2021.

[220] theLadders. *Eye-Tracking Study*, 2018. Retrieved from: `https://www.theladders.com/static/images/basicSite/pdfs/TheLadders-EyeTracking-StudyC2.pdf`, last accessed: October 2021.

[221] Tijmen Tieleman and G Hinton. Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012.

[222] Minh-Luan Tran, Anh-Tuyen Nguyen, Quoc-Dung Nguyen, and Tin Huynh. A comparison study for job recommendation. In *2017 International Conference on Information and Communications (ICIC)*, pages 199–204. IEEE, 2017.

[223] Nam Khanh Tran. Classification and learning-to-rank approaches for cross-device matching at CIKM cup 2016. *arXiv preprint arXiv:1612.07117*, 2016.

[224] Jorge Carlos Valverde-Rebaza, Ricardo Puma, Paul Bustios, and Nathalia C. Silva. Job recommendation based on job seeker skills: An empirical study. In *Text2Story@ ECIR*, pages 47–51, 2018.

[225] David L. Van Rooy, Alexander Alonso, and Zachary Fairchild. In with the new, out with the old: Has the technological revolution eliminated the traditional job search process? *International journal of selection and assessment*, 11(2-3):170–174, 2003.

[226] José Vega. Semantic matching between job offers and job search requests. In *COLING 1990 Volume 1: Papers presented to the 13th International Conference on Computational Linguistics*, 1990.

[227] Maksims Volkovs, Guang Wei Yu, and Tomi Poutanen. Content-based neighbor models for cold start in recommender systems. In *RecSys Challenge '17: Proceedings of the Recommender Systems Challenge 2017*, pages 1–6. 2017.

[228] Chao Wang, Yiqun Liu, Meng Wang, Ke Zhou, Jian-yun Nie, and Shaoping Ma. Incorporating non-sequential behavior into click models. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 283–292, 2015.

[229] Hongning Wang, ChengXiang Zhai, Anlei Dong, and Yi Chang. Content-aware click modeling. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1365–1376, 2013.

[230] Jian Wang, Yi Zhang, Christian Posse, and Anmol Bhasin. Is it time for a career switch? In *Proceedings of the 22nd international conference on World Wide Web*, pages 1377–1388. ACM, 2013.

[231] Pengyang Wang, Yingtong Dou, and Yang Xin. The analysis and design of the job recommendation model based on GBRT and time factors. In *2016 IEEE International Conference on Knowledge Engineering and Applications (ICKEA)*, pages 29–35. IEEE, 2016.

[232] Ping Wang, Yan Li, and Chandan K Reddy. Machine learning for survival analysis: A survey. *arXiv preprint arXiv:1708.04649*, 2017.

[233] Qingchen Wang. Recombining customer journeys with probabilistic cookie matching: A supervised learning approach. In *Machine learning applications in operations management and digital marketing*, chapter 6, pages 127–139. 2019.

[234] Marcel Wolbers, Paul Blanche, Michael T Koller, Jacqueline CM Witteman, and Thomas A Gerds. Concordance for prognostic models with competing risks. *Biostatistics*, 15(3):526–539, 2014.

[235] Wenming Xiao, Xiao Xu, Kang Liang, Junkang Mao, and Jun Wang. Job recommendation with hawkes process: an effective solution for recsys challenge 2016. In *Proceedings of the recommender systems challenge*, pages 1–4. 2016.

[236] Xing. *Website Xing*, 2021. Retrieved from: `https://www.xing.com`, last accessed: October 2021.

[237] Peng Xu and Denilson Barbosa. Matching résumés to job descriptions with stacked models. In *Canadian Conference on Artificial Intelligence*, pages 304–309. Springer, 2018.

[238] Ya Xu, Nanyu Chen, Addrian Fernandez, Omar Sinno, and Anmol Bhasin. From infrastructure to culture: A/B testing challenges in large

scale social networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2227–2236, 2015.

[239] Murat Yagci and Fikret Gurgen. A ranker ensemble for multi-objective job recommendation in an item cold start setting. In *Proceedings of the Recommender Systems Challenge*, pages 1–4. 2017.

[240] Rui Yan, Ran Le, Yang Song, Tao Zhang, Xiangliang Zhang, and Dongyan Zhao. Interview choice reveals your preference on the market: To improve job-resume matching through profiling memories. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 914–922, 2019.

[241] Shuo Yang, Mohammed Korayem, Khalifeh AlJadda, Trey Grainger, and Sriraam Natarajan. Combining content-based and collaborative filtering for job recommendation system: A cost-sensitive statistical relational learning approach. *Knowledge-Based Systems*, 136:37–45, 2017.

[242] In-Kwon Yeo and Richard A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000.

[243] Safoora Yousefi, Fatemeh Amrollahi, Mohamed Amgad, Chengliang Dong, Joshua E Lewis, Congzheng Song, David A Gutman, Sameer H Halani, Jose Enrique Velazquez Vega, Daniel J Brat, et al. Predicting clinical outcomes from large scale cancer genomic profiles with deep survival models. *Scientific Reports*, 7(1):11707, 2017.

[244] Kun Yu, Gang Guan, and Ming Zhou. Resume information extraction with cascaded hybrid model. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 499–506. Association for Computational Linguistics, 2005.

[245] Abeer Zaroor, Mohammed Maree, and Muath Sabha. A hybrid approach to conceptual classification and ranking of resumes and their corresponding job posts. In *International Conference on Intelligent Decision Technologies*, pages 107–119. Springer, 2017.

[246] Abeer Zaroor, Mohammed Maree, and Muath Sabha. Jrc: a job post and resume classification system for online recruitment. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 780–787. IEEE, 2017.

[247] Chenrui Zhang and Xueqi Cheng. An ensemble method for job recommender systems. In *RecSys Challenge '16: Proceedings of the Recommender Systems Challenge*, pages 1–4. 2016.

[248] XianXing Zhang, Yitong Zhou, Yiming Ma, Bee-Chung Chen, Liang Zhang, and Deepak Agarwal. Glmix: Generalized linear mixed models for large-scale response prediction. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 363–372, 2016.

[249] Yuchen Zhang, Dong Wang, Gang Wang, Weizhu Chen, Zhihua Zhang, Botao Hu, and Li Zhang. Learning click models via probit bayesian inference. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 439–448. ACM, 2010.

[250] Yuchen Zhang, Weizhu Chen, Dong Wang, and Qiang Yang. User-click modeling for understanding and predicting search-behavior. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1388–1396. ACM, 2011.

[251] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists.* O'Reilly Media, Inc., 2018.

[252] Chen Zhu, Hengshu Zhu, Hui Xiong, Chao Ma, Fang Xie, Pengliang Ding, and Pan 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):1–17, 2018.

[253] Zeyuan Allen Zhu, Weizhu Chen, Tom Minka, Chenguang Zhu, and Zheng Chen. A novel click model and its applications to online advertising. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 321–330, 2010.

[254] Dávid Zibriczky. A combination of simple models by forward predictor selection for job recommendation. In *RecSys Challenge '16: Proceedings of the Recommender Systems Challenge*, pages 1–4. 2016.

[255] Rebecca R. Zusman and Ronald S. Landis. Applicant preferences for web-based versus traditional job postings. *Computers in Human Behavior*, 18(3):285–296, 2002.

Samenvatting

Zoekmachines en aanbevelingssystemen hebben een bepalende invloed op het online wervingsproces van personeel. Voor werkzoekenden bepalen ze welke vacatures worden aanbevolen op vacaturewebsites en in welke volgorde, maar ook zijn ze belangrijk voor recruiters. Deze systemen bepalen namelijk welke kandidaten worden getoond bij het zoeken in een CV-database en in welke volgorde.

Een zoekmachine/aanbevelingssysteem maakt daarbij gebruik van schattingen van de relevantie van een vacature/kandidaat voor een werkzoekende/recruiter. Deze geschatte relevantie bepaalt uiteindelijk welke vacatures/kandidaten in het zoekresultaat voorkomen en in welke volgorde. Om deze relevantie te schatten, moet een zoekmachine/aanbevelingssysteem het doen met beperkte informatie. Vaak zijn alleen de huidige en voorgaande zoekopdrachten bekend, zowel van de persoon die de zoekopdracht uitvoert, als de vorige zoekopdrachten van anderen. De zoekmachine/aanbevelingssysteem kan daarnaast gebruik maken van interactie op zoekresultaten. Oftewel, op welke vacatures/kandidaten er is doorgeklikt en op welke niet. De vraag is nu hoe op basis van deze informatie de relevantie zo goed mogelijk te schatten. Deze dissertatie beschouwt daarvoor verschillende algoritmen, die als doel hebben een betere schatting te maken van de relevantie van een vacature/kandidaat voor een werkzoekende/recruiter.

Hoofdstuk 2 geeft daarbij een overzicht van de huidige literatuur op het gebied van vacature-aanbevelingssystemen. In overeenstemming met andere vakgebieden, laat dit hoofdstuk daarin een stijging zien in het aantal contributies dat gebruik maakt van Deep Neural Networks. Deze modellen zijn in het bijzonder sterk in het automatisch genereren van numerieke (vector)representaties van vacatures en een CV's. Deze representaties kunnen vervolgens worden gebruikt om een schatting te maken van relevantie, bijvoorbeeld door een afstand te bepalen tussen de representatie van een vacature en CV. Dat deze modellen goed in staat zijn om numerieke representaties te maken is bijzonder. Niet alleen kan een vacaturetekst/CV ambigue zijn, ook kunnen recruiters en werkzoekenden gebruik maken van verschillende terminologieën. Vacatures en CV's zijn daardoor vaak lastig tekstueel te matchen.

Mogelijk worden echter toepassingsspecifieke problemen in vacature-aanbevelingssystemen genegeerd. Een vacature kan misschien volgens de numerieke representatie goed matchen met een CV, maar wanneer er al veel sollicitaties op deze vacature zijn, is het dan in het belang van de werkzoekende/recruiter om de vacature aan te bevelen? Contributies die een meer toepassingsspecifiek perspectief hebben laten de voordelen daarvan zien. Zo kan het verlagen van de positie van een zoekresultaat voor vacatures met veel sollicitaties leiden tot een betere spreiding van kandidaten over vergelijkbare vacatures. Daarnaast kan aandacht aan de eerlijkheid van het zoek-/aanbevelingsalgoritme voorkomen dat personen met dezelfde kennis en ervaring toch onterecht verschillende aanbevelingen krijgen.

Hoofdstuk 3 gaat in op het schatten van relevantie op basis van klikgedrag. Sinds de komst van zoekmachines zijn er verschillende modellen geïntroduceerd die klikgedrag op zoekmachines proberen te verklaren. Deze modellen worden ook wel klikmodellen genoemd. De relevantie van een item (zoals een kandidaat/vacature) voor een gebruiker (werkzoekende/recruiter) speelt daarin een belangrijke rol. Relevantie is echter niet het enige wat klikgedrag bepaalt. Ook de positionering van zoekresultaten speelt een rol. Items die hoger in het zoekresultaat staan hebben een grotere kans om op te worden geklikt, ongeacht of deze ook het meest relevant zijn. Dit wordt ook wel het positie-effect genoemd.

Klikmodellen hebben vaak een eigen methode om de parameters uit het model te schatten. Het is echter mogelijk om veel van deze modellen om te schrijven naar een generieker model, het zogenaamde Generalized Cascade Model (GCM). De parameters van een GCM kunnen geschat worden met behulp van de Expectation-Maximization-schattingsmethode voor Input/Output Hidden Markov Models. Dit betekent dat het niet nodig is om voor elk klikmodel apart een schattingsmethode uit te werken. Als het model omgeschreven kan worden naar een GCM, dan biedt GCM direct een methode om de parameters van het klikmodel te schatten.

Hoofdstuk 3 laat zien hoe van Expectation-Maximization voor Input/Output Hidden Markov Models gebruik gemaakt kan worden om de parameters van een GCM, en daarmee het oorspronkelijke klikmodel, te schatten. Daarnaast geeft het een aantal voorbeelden hoe bestaande klikmodellen kunnen worden geschreven als GCM. De schattingsmethode is ook geïmplementeerd in een `Python` package onder de naam `gecasmo`.

Hoofdstuk 4 schat het effect van gebruikerscensurering op het achterhalen van welke gebruiker welk klikgedrag heeft vertoond. Websites, en daarmee zoekmachines, gebruiken vaak browsercookies om te achterhalen welke klikdata bij welke gebruiker hoort. Wanneer de gebruiker echter via meerdere apparaten de zoekmachine benadert, of de browsercookies weggooit, kunnen die browsercookies niet eenvoudig meer bepalen welk klikgedrag van wie afkomstig is.

Om het effect van gebruikerscensurering te bepalen, wordt er in Hoofdstuk 4 een kliksimulatiemodel geïntroduceerd. Ook vergelijkt dit hoofdstuk verschillende clusteringsmethoden op basis van (H)DBSCAN*. Deze clusteringsmethoden proberen te achterhalen welk gedrag van welke gebruiker afkomstig is. Ze maken daarbij niet/beperkt gebruik van informatie uit browsercookies.

Uit de analyse blijkt dat gebruikerscensurering door het gebruik van meerdere apparaten en het verwijderen van browsercookies gering is. Ondanks de censurering zijn browsercookies vrij nauwkeurig in staat om gebruikers te identificeren. Daarmee kan er door browsercookies ook vrij nauwkeurige schattingen worden gegeven van webstatistieken die betrekking hebben op deze gebruikers. Voorbeelden van deze webstatistieken zijn het aantal internetsessies per gebruiker, of het aantal unieke gebruikers op de website. Voor de (H)DBSCAN*-clusteringsmethoden is dit een ander verhaal. Ondanks dat deze methoden significant beter presteren dan een naïve clusteringsmethode, blijven de resultaten van deze methoden ver achter bij de resultaten die gevonden zijn bij het gebruik van browsercookies.

Hoofdstuk 5 gaat in op hoe lang iemand in zijn/haar functie zal blijven, ook wel bekend als baanduur. Dit wordt gedaan door uit de werkervaringssecties van CV's de duur van de daarin genoemde banen te extraheren. Vervolgens worden een aantal modellen vergeleken die deze baanduren proberen te voorspellen. Daarbij wordt gebruik gemaakt van overige informatie uit het CV, zoals het type functie waarvoor een voorspelling wordt gemaakt, of de werkervaring van de persoon tot het moment dat deze persoon aan de functie in het CV begon.

De resultaten laten zien dat voornamelijk tijdselementen uit het CV een voorspellende waarde hebben voor de baanduur. Ondanks dat dit logisch lijkt, worden deze tijdselementen momenteel nauwelijks meegenomen in kandidaataanbevelingssystemen. Wel blijkt dat het voorspellen van baanduur lastig is. De baanduur wordt in de werkervaringssectie vaak afgerond naar hele jaren, waardoor kortetermijnvoorspellingen lastig zijn. De resultaten geven dan ook de voorkeur aan robuuste modellen.

Hoofdstuk 6 richt zich op het voorspellen van het wekelijkse aantal sollicitaties op een recruitmentwebsite. Deze voorspellingen geven daarmee aan welke vacatures meer/minder stimulans nodig hebben. Uit een vergelijking van een aantal Machine Learning-algoritmen geeft vooral Random Forest goede resultaten.

Summary

Search engines and recommender systems have an important influence on the recruitment process, both for recruiters and job seekers. For both, they determine which job seekers/vacancies are recommended in what order, on résumé databases or recruitment websites. In doing so, they influence which vacancies job seeker apply to, or which job applicants are selected for an interview. To determine which vacancies/candidates to show and in what order, the search engine/recommender system uses estimations of the relevance of a vacancy/-candidate for a job seeker/recruiter.

To estimate this relevancy, the search engine/recommender system often only have partial information. In particular, only the search queries are known, and the click interaction on the search result. This interaction data shows which vacancies/candidates have been clicked on by users of the search engine/recommender system. This dissertation considers multiple algorithms, with the goal of improving estimations of relevance in the context of job/candidate recommender systems and search engines.

Chapter 2 provides an overview of the current literature in the field of job recommender systems. In agreement with other application areas, this literature shows an increase in the number of contributions that use Deep Neural Networks. These models are used to create vector embeddings of vacancies and résumés. These embeddings can subsequently be used to estimate relevance, for example, by computing distances between the vector representations of vacancies and résumés. The deep representations seem to resolve some problems of text ambiguity, and the problem of job seekers and recruiters using different terminologies to describe jobs.

Perhaps because of the focus on creating good vector embeddings, other application-related problems in job recommender systems are sometimes neglected.

A vacancy and a résumé may semantically match well, however, if the vacancy already has many applications, is it then beneficial to recommend the vacancy to the job seeker (or vice versa)? Contributions that have a more application-oriented view show the benefits of such a view. This literature suggests that lowering the rank of the vacancy in the search result may lead to a better spread of candidates over multiple similar vacancies. Contributions that put attention to algorithm fairness also show that this may prevent two job seekers with identical backgrounds to unjustly receive different recommendations.

Chapter 3 focuses on estimating relevance based on click data. Several models have been proposed in the literature that try to explain the behavior often observed on search engines and recommender systems. These models are also known as click models. Although the relevance of an item (such as a candidate/vacancy) for a user (recruiter/job seeker) plays an important role in these models, it is not the only variable that determines search behavior. Also the position of the items has an effect, which is often referred to as the position bias.

Click models often have their own method for estimating the parameters of the model. It is, however, possible to write many of these models as a more generic click model, which we call the Generalized Cascade Model (GCM). The parameters of a GCM can be estimated by using Expectation-Maximization for Input/Output Hidden Markov Models. This means that it is not necessary for each click model to have a separate estimation procedure. If the model can be written as GCM, the Expectation-Maximization procedure for Input/Output Hidden Markov Models directly provides a method for estimation the parameters of the GCM, and therefore, for the original click model. Chapter 3 shows how Expectation-Maximization for Input/Output Hidden Markov Models can be used to estimate the parameters of a GCM. Also, it provides a number of examples of how existing click models can be rewritten as GCM. The estimation procedure has also been implemented in a `Python` package called `gecasmo`.

Chapter 4 estimates the effect that user censoring has in detecting which user caused what website traffic. Websites use browser cookies to determine what click data originated from which user. However, when website users use multiple devices, or remove their browser cookies from their browser, the browser cookies cannot easily be used anymore to determine the user.

To determine the effect of user censoring, Chapter 4 introduces a click simulation model. Also, this chapter compares several clustering methods based on the (H)DBSCAN* algorithm. These clustering algorithms try to determine what click data originated from which user, without — or by making limited use of — cookie data. The results suggest that the effect of user censoring due to the usage of multiple devices/removing cookies is small. Despite the user censuring, browser cookies are still quite able to determine which user generated which sessions. As a result, the observed web statistics under user censoring with respect to users, such as the number of sessions per user or the number of unique users, are also quite accurate. Although the (H)DBSCAN* methods perform significantly better than a naive clustering approach, the results are considerably worse compared to the results obtained by using browser cookies.

Chapter 5 compares several machine learning methods that predict how long someone will remain in his/her job, which we refer to as job tenure. This is done by extracting these job tenures from the work experience section often found in résumés. To estimate job tenure, the models use other information found in the résumé, such as the function type, or the work experience up until the moment the person started the job.

The results indicate that in particular time-related features are good predictors in estimating future tenure. Although this may seem trivial, time-related features are typically not taken into account in candidate recommender systems. Predicting job tenure from résumés is, however, a difficult problem. Job tenure is often rounded to entire years, making it difficult to make short-term predictions based on this data. The results therefore also prefer robust models.

Chapter 6 compares several machine learning models in predicting the weekly number of applications that a vacancy on a recruitment website will receive. The predictions can be used to determine which vacancies may require more/-less attention on the recruitment website. From this model comparison, we find that in particular Random Forest works well for this purpose.