Using Implicit Job Seeker Preferences To Improve Job Recommendation Ranking

A.T. Walterbos

Technische Universiteit Delft

Using Implicit Job Seeker Preferences To Improve Job Recommendation Ranking

by

A.T. Walterbos

in partial fulfillment of the requirements for the degree of

Master of Science
in Computer Science

at the Delft University of Technology,
to be defended publicly on Monday May 27, 2019 at 15:00.

Thesis committee:

Chair:	Prof.dr. Alan Hanjalic
University Supervisor:	Cynthia Liem, MMus
Committee Member:	Dr. Nava Tintarev
Company Supervisor:	Rogier Slag, MSc

TU Delft Delft University of Technology

Preface

This thesis focuses on the field of Job Recommendation. Particularly, we focus on using implicit preferences exhibited by the job seeker in interactions with a web platform to propose an improved ranking algorithm for a job recommendation platform called Magnet.me. We also study evaluation of relevance, and evaluation of recommendation sorting algorithms to determine the degree of improvement achieved by the proposed algorithm. Using NDCG with different relevance evaluations, we test performance of the proposed algorithm in an online experiment on the job recommendation platform.

We find that the evaluation of relevance strongly affects the distinguishability of NDCG. The evaluation shows that our sorting algorthm outperforms the original algorithm when using classical binary relevance, or relevance evaluations that consider items with negative feedback less relevant than items with missing feedback. However, when using relevance evaluations for NDCG that punish missing feedback more than negative feedback, NDCG loses its capability of distinguishing between algorithm performance.

Based on baseline sorting algorithm evaluation MRR and the different evaluations using NDCG, we conclude that the proposed recommendation sorting algorithm outperforms the original algorithm.

I would like to thank Cynthia Liem for her supervision; my colleagues at Magnet.me, particularly Rogier and Kolpa, for their help; and finally Mara, for bearing with me.

A.T. Walterbos
Delft, May 2019

Contents

1

Job Recommendations

In this thesis, we study the field of Job Recommendations. Recommending jobs to job seekers is the challenge many e-recruitment platforms like LinkedIn, Xing, Glassdoor, Google for Jobs and others try to solve. After their success in e-commerce, recommender systems made their entry into the e-recruitment field; the vastly increased number of jobs available through the internet, and vice versa the number of applicants for a vacancy creates the need for recommender system technology to help handle this information efficiently [1]. Recommender systems aim to filter out those items that are most interesting to the user within the context of the user's need or aim. Different platforms will have varying definitions of what specifically is 'most interesting' to a user, but the general principle applies everywhere.

1.1. Approaches in Recommender Systems

Like in more general recommender systems, job recommender systems employ collaborative filtering, content-based filtering, or a combination of both [1, 2]. More recently, filtering based on demographic data has been used in recommender systems [3].

Recommender systems that use collaborative filtering apply known preferences of a set of users to predict the preferences for another (new) user. They are based on the assumption that if two users have rated items similarly or express similar behaviour, they will rate other items similarly as well. Collaborative filtering is the most popular approach in general recommender systems, employed successfully by many companies like Amazon, Facebook, Google and Netflix [4]. It excels in domains where it is hard to describe items with an automated process, like sound or movies [1].

Content-based recommender systems use a model of the user's preferences, and a detailed description or list of properties of items. These preference models, or profiles, and item descriptions are then compared, and items that match the user's preferences best are recommended. Content-based filtering does not depend on an overlap between user preferences, like collaborative filtering does. However, it does depend on how well features can be selected from the items to recommend [5].

One of the first job recommender systems was developed by Rafter et al. [6]. Their system employed a user-collaborative filtering approach, and only uses relatively simple user behaviour data such as number of visits, and total time spent reading on pages to build profiles of users. Later, researchers suggested that job recommendations required far more use of properties of the job seekers, such as their level of education, work experience and location [7–9]. These systems also take into account the reciprocal interest of the employer in the job seeker.

Job Recommendation is still a very active field of research: the subject of the RecSys Challenge 2016, part of the ACM Recommender Systems Conference 2016, was Job Recommendations. Contestants were asked to create an offline evaluation method for data from social business network Xing [10]. Xiao et al. [11] won this contest, applying a combination of modern techniques in job recommendation.

1.2. Implicit and Explicit Feedback

Compared to other domains, job recommendations are considered a high-risk recommendation context [12]. High-risk means that providing a 'wrong' recommendation yields a big negative impact: job

seekers will be left unsatisfied, and could abandon the platform. Traditional job recommender systems are therefore more conservative, and rely more on explicit feedback [13] from the user: they only recommend jobs based on what the job seeker explicitly likes.

Alternatively, there are job recommenders that consider implicit feedback [14], or even rely solely on that [15]. Implicit feedback is feedback that a user gives through their natural behaviour, from which preferences can be derived. Consciously entered preferences (explicit feedback) are generally seen as more robust and more intuitively reliable.

1.3. Research Questions and Thesis Structure

In this thesis, we focus on collecting implicit user preferences by analysing user interaction data. We will use those preferences to improve the job recommendation algorithm of job platform Magnet.me. To determine if performance improves, we study methods of evaluating recommender system performance in an online environment. We formulate our Research Questions based on this focus, as follows below.

The main question in this thesis will be: **How can we improve Magnet.me's Job Recommendations algorithm using User Interaction Data?** To answer this question, we define these three research questions:

RQ1 How can we gain a better understanding of job seeker preferences, beyond their explicitly provided preferences?

- (a) What job seeker interaction data can we monitor to derive implicit preferences from?
- (b) How can we aggregate implicit job seeker preferences from job seeker interaction data?

RQ2 How can we use implicit job seeker preferences to improve the opportunity recommendation ordering algorithm?

RQ3 How can we evaluate rank performance, and therewith distinguish between ranking algorithm performance?

- (a) How can we evaluate recommendation relevance based on job seeker feedback?
- (b) How should we handle recommendations for which feedback is missing?

The structure of this thesis is as follows: First, we look into related work on deriving preferences from user behaviour, evaluation of recommender systems, and online recommender system evaluation in Chapter 2. In Chapter 3, we introduce Magnet.me, discuss what job recommendation techniques Magnet.me has employed so far, and look at possible ways to improve upon them.

In Chapter 4 we propose monitoring job seeker behaviour at Magnet.me, and analyse if- and how we can derive an improved understanding of job seekers from that behaviour in the form of implicit preferences. Based on the derived implicit preferences, we propose a new sorting algorithm. We study several aspects of rank evaluation and propose an evaluation method for sorting algorithms in Chapter 5. In Chapter 6, we design and run an online experiment with the alternative recommendation sorting algorithm based on implicit preferences, as well as a comparison of different parameters for our rank evaluation method. We reach conclusions on several facets of rank evaluation, and discuss the performance of the new sorting algorithm. Finally, we discuss the experiment and possible future work in Chapter 7.

2

Related Work

We first look at work related to the focus of this thesis. That focus is bipartial: First, we focus on the collection of implicit preferences from user interactions based on which we want to improve a recommender system; therefore, we look into related work on deriving preferences from user behaviour. Second, we want to be able to evaluate the recommender system to measure improvements from our work. We look into related work in the field of rank evaluation, and specifically highlight some evaluation measures. Finally, we discuss related work on online recommender system evaluation.

2.1. Deriving Preferences from User Behaviour

Part of the focus of this thesis is the aggregation of user preferences from behaviour exhibited on a web platform. Contrary to consciously provided preferences, we seek to extract implicit preferences that can be used to refine the model of a user's preferences that have already been acquired. Being the opposite of implicit preferences, consciously provided preferences are referred to as 'explicit preferences' [16].

Implicit preferences were considered less reliable than explicit preferences [17], but more recently this opinion has changed [18]. When seen as extra data besides explicit preferences, they can greatly improve the understanding of a user's overall preferences.

Within the subject of web design, user interactions are the interactions that users make with web pages in the form of navigation, clicking, scrolling and other ways to consume web page content. In particular, patterns in web browsing can be studied to gain understanding of many facets of web systems. Studying web browsing patterns is no new subject of research: as early as in the year 2000, Strivastava et al. [19] describe scraping browsing history to derive usages of web sites, and using them to improve systems, gain knowledge on how users use the system, and personalisation of content. It has been an active topic of research since [20–22], and is widely applied in modern web platforms.

It is widely assumed that longer gaze times indicate a higher level of attention. Morita et al. [23] report longer reading times for articles marked as interesting, while they find no correlation between reading time and article length and article readability. Though, understandably, Dimpfel et al. [24] postulate that longer reading times could be caused by more complicated content. Researchers have also studied attention in web page usage using eye tracking, pupil size, and electroencephalography (EEG) [25–27].

Peska et al. [28] built a recommender on users interacting with web pages, like clicking on links and buttons to expand text, and thereby access more info. They conclude that this behaviour expresses interest from the user in the displayed information.

In job recommendations, Reusens et al. [15] developed a recommender system for job recommendations, based on implicit preferences. Though the use of implicit preferences in this field is unusual due to the high-risk nature of job recommendations, they have shown that their recommender system outperforms the knowledge-based system that preceded it.

2.2. Rank Evaluation

The aim of this work is to improve the performance of a recommender system. To conclude that improvement is achieved, one must first have means of monitoring recommender system performance.

We discuss the field of recommender systems, and evaluation techniques used to monitor these systems.

Recommender systems are strongly related to the field of Information Retrieval. The challenge in both fields is finding the right items or documents from vast numbers of them, most of which are not relevant to the query at hand. Determining what information should be retrieved depends on the fit between the information need of the user, and the available documents.

The problem is often modelled as optimising the order of the documents or items in the data set on relevancy to the user query, and presenting the user with the items with the highest predicted relevancy. In practice, this results in optimising some rank evaluation of the ordering algorithm output.

2.2.1. Relevance

Performance of a recommender system is therefore determined by how relevant its recommendations are to the user. Relevance is defined as how well the generated recommendation fits the user's (information) need, and a recommendation is marked relevant or non-relevant based on user feedback [29]. To determine this fit, one would ideally have explicit feedback from the user for every item the recommender suggests. Such feedback could, for example, be a (non-)binary rating of the item. Often, such explicit feedback is not available; for example, web search engines only have little feedback, mostly in the form of item clicks. The first implementation of relevance originates from an experiment conducted in 1953, described by Swanson et al. [30]: two teams assessed relevance of several thousand documents to a hundred questions, but could not agree on the relevance of more than half of the documents.

2.2.2. Accuracy Evaluations

Traditionally, evaluation of ranking algorithms are based on the accuracy of the algorithm; how well the algorithm predicts the relevance of an item to a user's information need. Precision and recall are metrics that evaluate this; 'Precision' is the fraction of retrieved documents that are relevant, and 'recall' is the fraction of relevant documents that are retrieved [29]. By definition, recall is hard to accurately determine for data sets with sparse feedback: if one does not know which documents are relevant to a query, one cannot determine what the ratio of retrieved relevant documents to all relevant documents is. Precision and recall are evaluations of unranked sets of recommendations. They are used in broader-scale evaluations of recommender systems by combining and/or aggregating the precision and recall over many generated recommendation sets, such as the F-measure, introduced by Van Rijsbergen [31].

Examples of evaluations that aggregate precision and recall are Precision@k, Mean Average Precision (MAP), and the ROC-curve [29, 32]. These evaluation measures are all only capable of handling binary relevance feedback: an item is either relevant or non-relevant, but there are no multiple grades of relevance possible. Except for Precision@k, these measures factor in precision at all recall levels. This is not applicable at many information problems on web platforms [29]. Precision@k only looks at the top k documents, which does not require any information on the number of relevant documents.

The same applies to the very simple Mean Reciprocal Rank (MRR). The Reciprocal Rank (RR) is calculated as $\frac{1}{r_q}$ where r_q is the rank of the first relevant item in queue q. The MRR is then the mean of the RR for all generated result sets Q:

$$\text{MRR} = \sum_{q \in Q} \frac{1}{r_q} \tag{2.1}$$

This simple metric was introduced by Voorhees et al. [33], and is considered useful because it is closely related to the Mean Average Precision.

NDCG

The Normalised Discounted Cumulative Gain (NDCG) also does not depend on recall, or an estimate of it. NDCG is a measure that is widely used in information retrieval, and machine learning [34]. Because of its popularity, it has also been researched extensively, though mainly in the field of Information Retrieval [35–37] NDCG uses a numeric relevance score for an item as a 'gain', and discounts this relevance for higher ranks. By doing so, the NDCG of a ranked list of recommendations is higher if the more relevant recommendations are on top of the list. It is the normalised form of the Discounted

Cumulative Gain (DCG) of the result set by the DCG of the ideally sorted version of the same result set, based on relevance [38]. Because it uses numeric relevance, it can consider multiple grades of relevance for the recommendations. It can still handle binary feedback, by simply using gain score 0 for non-relevant items and 1 for relevant items [38]

Wang et al. [38] define the DCG of a ranked result set S as:

$$\text{DCG}(S) = \sum_{r=1}^{|S|} y(r)D(r) \tag{2.2}$$

where $D(r)$ is the discount for some rank r in S, and $y(r)$ is the relevance of the item at rank r in S.

They then define NDCG as:

$$\text{NDCG}(S) = \frac{\text{DCG}(S)}{\text{IDCG}(S)} \tag{2.3}$$

Where $\text{IDCG}(S) = \max\left(\sum_{r=1}^{|S|} y(r)D(r)\right)$, i.e. the highest possible DCG score for the result set S. The latter is achieved by sorting S on the relevance of all items $y(r)$ for all $r \in S$.

There are many implementations of NDCG, using different manipulations of the relevance score and the discount function. For example, while Wang et al. [38] simply use the relevance scores $y(r)$ from the result set S, Manning et al. [29] uses $2^{y(r)} - 1$. This adaptation boosts higher relevance stronger and punishes low relevance.

The discount functions vary per implementation as well. Its function is to discount the relevance gain for higher ranks. Wang et al. [38] consider NDCG 'standard' if the discount function used is the inverse logarithmic decay $D(r) = \frac{1}{log(1+r)}$. Another example of a discount function is $D(r) = r^{-1}$, as used by Kanoulas et al. [36].

Wang et al. [38] have shown that NDCG is capable of 'continuous distinguishability', which means to say that NDCG is capable of distinguishing between performance of two recommendation ranking algorithms, and declare which algorithm outperforms the other. For NDCG to be capable to distinguish between performance of two ranking algorithms, one must choose an appropriate discount function that decays fast enough: the standard discount function $D(r) = \frac{1}{\log(1+r)}$ is such a function.

A metric similar to DCG is the half-life utility metric from Breese et al. [39]; it similarly awards a gain score for relevant item feedback, but compared to DCG it uses a faster decaying exponential discount function. Like NDCG, the half-life metric is often used normalised by the maximal possible score, which is the half-life score for the ideally sorted list. Ekstrand et al. [40] state that this metric is hard to apply because its parameters α, which influences the discount function's decay, and d, which influences relevance scoring, have to be chosen appropriately.

Restricted Result Sets

Many implementations restrict the number of results from the result set that they consider when evaluating [4, 38–40]. This is done to reduce the influence of items that the user has not seen, due to them naturally not regarding all results. Ties on the document set considered in measures of information retrieval were first introduces by Cooper [41] and later applied to precision and recall by Raghavan et al. [42]. McSherry et al. [43] propose versions of recall, precision, F1, average precision, reciprocal rank and NDCG that consider document cut-offs, but they do not suggest approaches for choosing the cut-off.

Raghavan et al. [42] suggest that the cutoff point can be based on several different statistics; the number of documents retrieved, the number of *relevant* documents retrieved, etcetera. In lieu with their first suggestion, Kluver et al.'s Half Life metric [4] suggest to set the parameter α so that the item at rank α has a 50% chance of being seen by the user: they set the value based on the number of documents retrieved by the user.

2.2.3. Beyond-accuracy Evaluations

Accuracy is not everything, as has been shown by McNee et al. [44] and Fazeli et al. [45]. The performance of a recommender algorithm can strongly differ whether one measures the user experience or the accuracy [45]. They show that user-centric evaluation results do not confirm results of traditional evaluation. Al-Maskari et al.'s [35] study shows that NDCG correlates poorly with user satisfaction

scores; however, they show that Cumulative Gain does correlate with user satisfaction. While accuracy is likely to remain an important aspect of recommender performance, recent studies have looked at other metrics to consider in evaluation of recommender systems:

Kaminsas et al.[46] describe diversity, serendipity, novelty, and coverage as objectives to optimise for besides accuracy, and propose metrics to achieve this objective. They motivate that instead of simply seeing recommenders as systems that should accurately predict user's ratings for unseen items, the recommended items should also be diverse, and (some) suggestions should be novel to the user.

2.3. Online Recommender System Evaluation

To evaluate work proposed in this thesis, we will run both offline and online experiments. Particularly the online experiment will evaluate a recommender system. Online experiments to evaluate recommender systems are wildly popular [47].

The online experiment is often preceded by offline experiments [4, 48] to see if the experiment is likely to yield good results. When the test succeeds, an online experiment is started. Online recommender systems evaluations are often structured as A/B-tests. Such experiments redirect a small portion of users to an alternative version of the recommender system [4, 32].

As Gunawardana and Shandi [32] state, there are several important considerations to be made to make the outcome of online evaluations valid: First of all, the sampling of users must be made randomly to avoid bias in user groups, affecting the experiment. Second, it is important to test one change at a time: when a new ranking algorithm is tested, the interface should be kept the same; otherwise one might actually be seeing results caused by the new interface, rather than the new ranking algorithm.

Considering this related work, we now move our attention to the job recommendation platform Magnet.me. We will discuss their situation, and relate it to the work discussed in this chapter.

3

Job Recommendations at Magnet.me

In this chapter, we introduce Magnet.me; we shortly outline what the philosophy behind their company is and what they stand for. After that, we focus on their job recommendations; first, we introduce the user interface in which job seekers are offered job recommendations, called the Matches page. Then, we discuss how job recommendations are generated, and how the recommendations are structured. Following that, we discuss how Magnet.me evaluates rank performance. Finally, we propose possible improvements for Magnet.me's job recommendations.

3.1. Introducing Magnet.me

Magnet.me is a job recommendation platform founded by Vincent Karremans, Freek Schouten and Laurens van Nues in Rotterdam, The Netherlands. The idea for Magnet.me came from the founder's frustrations about the job market: the only companies present on the market were the big corporates with big recruitment budgets, while smaller companies were hard to find and hard to reach. Magnet.me is online since 2012 and their mission is still to revolutionise how talent starts- and builds their careers, and to make the process of finding a job easy and personal.

The platform targets students, graduates and young professionals with up to seven years of experience. They offer them opportunities in the form of internships, traineeships and jobs for which they are qualified, and which suit their preferences. Magnet.me is active in The Netherlands and The United Kingdom, with over 2,500 employers and over 200,000 job seekers on the platform. More information about Magnet.me can be found on their press page [49].

In short, the global process a job seeker on Magnet.me goes through is as follows: A job seeker creates a profile that consists of a résumé, and preferences regarding employment. Visa versa, an employer creates a profile with minimal requirements for a position, or for potential employees for their company as a whole. Magnet.me's job recommender system then calculates matches between the job seeker profiles and these requirements. A job seeker is suggested these matches, they engage with the matches that interest them, and eventually apply.

In the rest of this chapter, we specify what the job seeker preferences comprise, followed by generating recommendations based on those preferences. Then we discuss how the job seeker is shown the recommendations, and how they interact with them. Finally, we look at how Magnet.me evaluates performance of the recommendation interface and discuss ways to improve that interface.

3.2. Job Seeker Preferences

Any recommender system needs a basic understanding of user preferences to make recommendations. Magnet.me uses a content-based filtering approach for their recommendations, and besides considering job seeker preferences, they also consider the requirements for an opportunity and the qualifications of the job seeker.

The user model on which recommendations are based consists of a résumé and explicitly provided preferences regarding employment. The résumé consists of followed- or current education, work experiences, extracurricular activities, language proficiency and acquired miscellaneous skills. The employment preferences consist of a number of categories:

- **Location**: where the job seeker wants to work. It can be a country (United Kingdom or The Netherlands); an area around a city; or 'everywhere' which translates to no restriction based on location.
- **Work experience**: years of work experience, denoted by ranges: 0 to 1 year, 1 to 3 years, 3 to 5 years and 5 to 10 years.
- **Job types**: (Graduate) internship, graduate scheme, traineeship, and job.
- **Job functions**: The role of the employment within the company. Examples are 'accounting' and 'management'. There are 33 job functions.
- **Industries**: The industry in which the employer is active. Examples are 'education' and 'government'. There are 42 industries.
- **Company sizes**: Number of employees, denoted by ranges: 1 to 10 employees, 11 to 50 employees, 51 to 200 employees, 201 to 500 employees, 501 to 1000 employees, or more than 1000 employees.
- **Availability**: The job seeker can indicate that they are not available, available per a future date, and available immediately.

For all categorical preferences (work experience, job types, job functions, industries, and company sizes), the user must select at least one option they are interested in.

This preference modal does not allow the user to indicate stronger preference for one property than the other; all chosen options are considered equal. This means that the categorical preferences are binary: for each value, the job seeker indicates whether they are interested in them or not. The reason for that is that Magnet.me's recommendation generation algorithm works with such binary input. In this thesis, we will study ways to improve this 'flat' model by getting a more detailed understanding of the job seeker's preferences. We will try to derive these fine-grained preferences from aggregated behaviour they exhibit on Magnet.me by looking for behaviour patterns that suggest implicit preferences.

3.3. Recommendation Generation by the Talent Matcher

At the core of Magnet.me's service that connects job seekers with their potential employers is the Talent Matcher. The Talent Matcher compares the qualifications and preferences of all job seekers to the requirements and properties of all opportunities and employers on Magnet.me; a content-based job recommender system. It generates a 'match' between a job seeker and an opportunity when the job seeker's qualifications meet the opportunity's requirements, and the opportunity's properties meet the job seeker's preferences.

The Talent Matcher also calculates a so called 'match score' for all matches,. This match score is a remainder from the first implementations of the Talent Matcher, where it was an indication of match quality. It is no longer considered a proper indication of match quality; reasons for that are described in Appendix A.1. However, the match score is still used in the Matches Page content where no better alternative is available.

These matches are suggested to the job seeker as recommendations on the Matches Page. There, the opportunities and employers are compiled to a list of suggestions called the Matches Queue.

3.4. Ranks at Magnet.me: The Matches Queue

In this section, we first explain how job seekers view the Matches Queue, and how they interact with it. Then, we discuss different versions of the Matches Queue, and when they are shown. We explain the setup and dynamics of the Curated Queue, and how performance of the queues is defined and monitored.

3.4.1. Job Seeker Interactions with the Matches Queue

The Matches Page is the landing page for authenticated job seekers. To give the reader an idea of what the job seeker sees, a screenshot of the page is shown in Figure 3.1; it shows a match with the job

'Software Engineer' at Magnet.me. In Figure 3.2 shows the same screenshot, overlaid with indications to explain what is what.

On the left, the Matches Page shows the first match in the Matches Queue as a card (area A, B and C). On the right, a summary of the job seeker's preferences is shown (area D). The Match card shows information about the employer (A); the company logo (A1), the company name (A2), the company industry (A3) and the number of employees (A4). Below that, area B shows information about the opportunity; the opportunity title (B1), the job type and location of employment (B2), several properties of the job like the job function and the level of education, and if available a salary (B3), and finally B4 shows the truncated opportunity description.

The job seeker interacts with the Match by clicking the link or buttons in area C. As shown in Figure 3.2, the job seeker is presented with options to 'like' (C2) or 'ignore' (C3) an opportunity, and via the 'read more or apply' (C1) button they can directly apply to the opportunity. A match with a company offers the options 'connect', which is equivalent to 'like' for opportunities, and companies can be ignored as well. We will refer to these interactions as `apply`, `like`, `connect`, and `ignore` to distinguish the interactions from the English words.

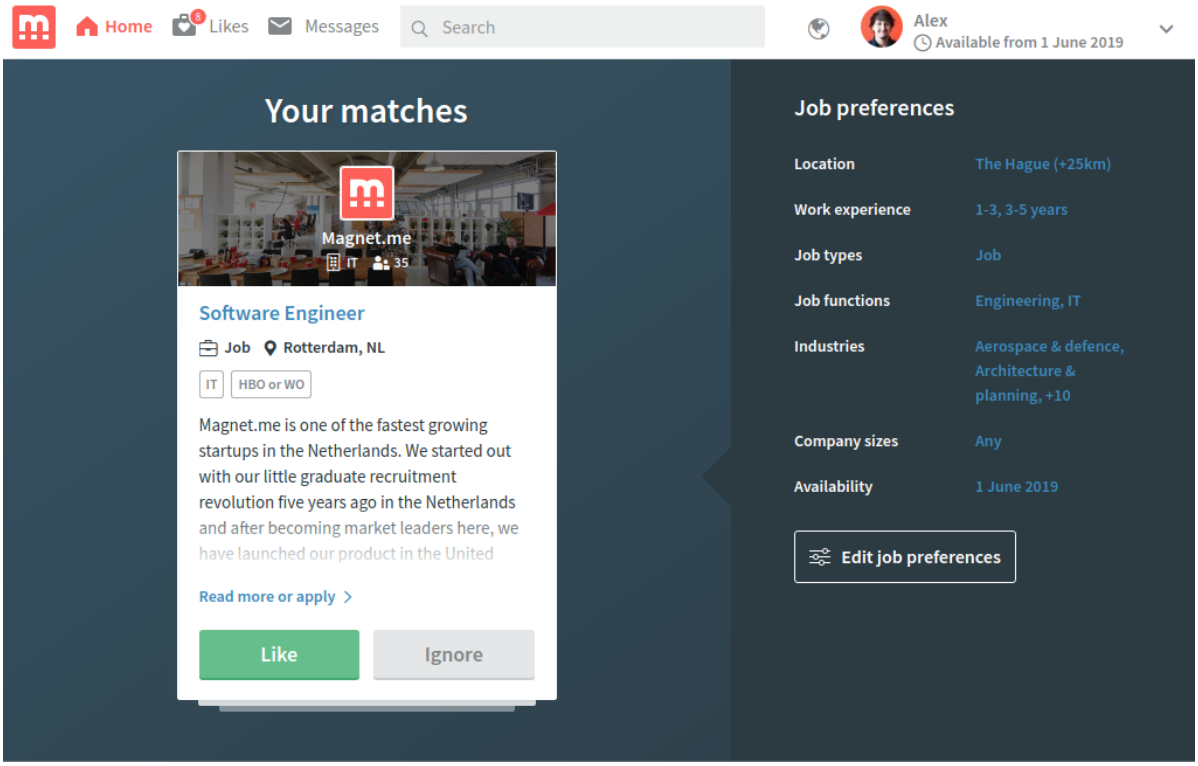

Figure 3.1: A partial screenshot of the Matches Page, showing the user interface elements relevant to Magnet.me's job recommender system. A match with the job 'Software Engineer' at Magnet.me is shown.

The queue-structured interface guides users through the recommendations generated by the Talent Matcher, asking them to `ignore`, `like`, `connect` or `apply` to each match before moving on to the next. This user experience has a number of properties that will prove relevant to our work: since the user cannot skip matches, we have feedback for all matches. However, the interface introduces fatigue: after a number of interactions, job seekers typically drop off by navigating elsewhere on Magnet.me, or ending their session.

3.4.2. Matches Queue Evaluation

Performance of the Matches Queue is currently monitored with two metrics: the like to ignore-ratio (LIR), and number of connections made (NOC).

LIR is a metric that compares the number of `likes`, `nOL`, to the number of `ignores`, `nOI`, and defined as $\text{LIR} = \frac{\texttt{nOL}}{\texttt{nOI}}$ for $\texttt{nOI} > 0$. If $\texttt{nOI} = 0$, $\text{LIR} = 1$. It is used to check whether matches served on

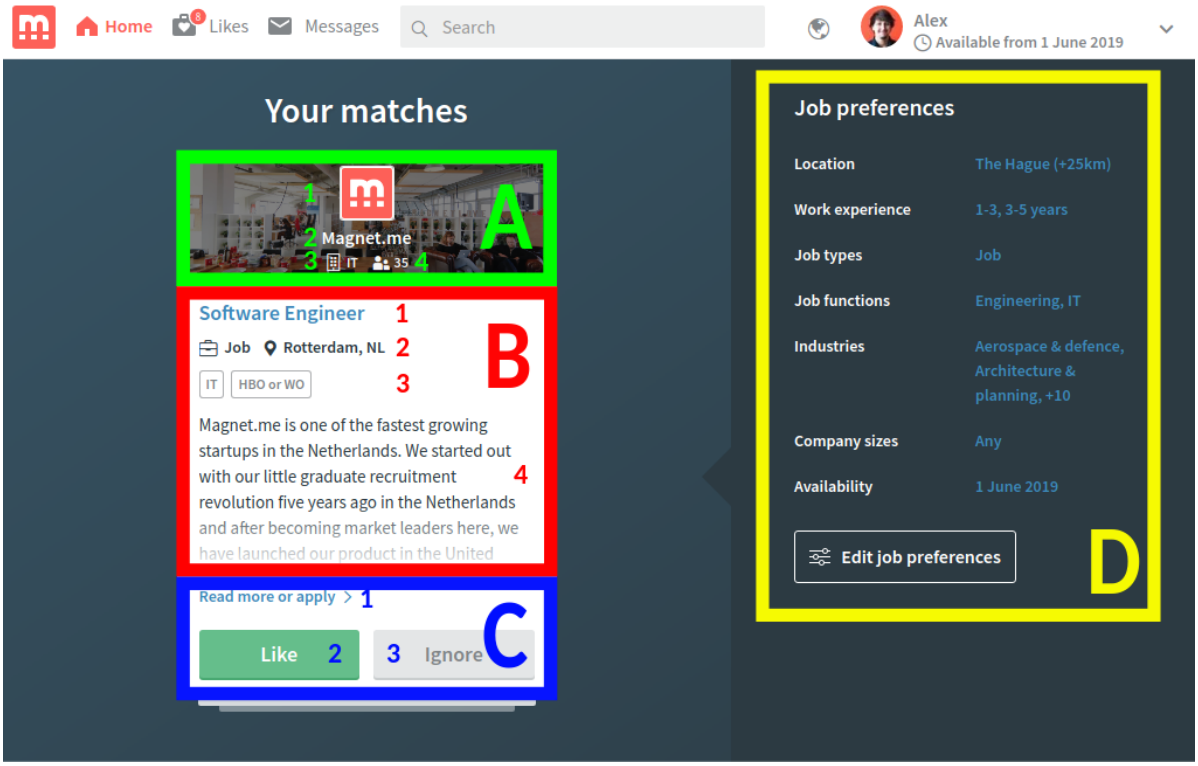

Figure 3.2: The screenshot from Figure 3.1, but overlaid with indicators of what information is where: Area A shows employer information. Information about the opportunity is shown in Area B, and the job seeker can interact with the match via the link and buttons in Area C. Area D shows a summary of the job seekers preferences and a button to edit them.

the Matches Page are relevant.

The number of connections made refers to the connections made when a job seeker engages an opportunity with a `like` or `apply`. The metric is therefore an indication of the absolute number of positive interactions a job seeker has made on Magnet.me, and since most of these connections are created on the Matches Page it can be used as a metric for Matches Queue evaluation.

Although these metrics have sufficed for Magnet.me, they are somewhat rudimentary. To distinguish any improvements made in our research, we require more reliable recommendation evaluation metrics. However, defining rank evaluation in a way that is meaningful for Magnet.me is not trivial. Related to RQ3, we will study ways to evaluate recommender system performance that can be implemented at Magnet.me, aimed at establishing a baseline of performance of the Matches Queue and comparing improvements we will propose.

3.4.3. Queue Generation

There are no pre-generated queues for a job seeker when they land on the Matches Page. Maintaining pre-generated queues for all events that change queues would require far more computational power than Magnet.me currently has available. Instead, queue generation is triggered when the job seeker lands on the page.

The generated queue, called the Curated Queue, consists of the job recommendations provided by the Talent Matcher. Generation of this queue, however, takes too long to make the job seeker wait for its content when loading the Matches Page. To provide the job seeker with content immediately, they are served an intermediate list of recommendations from the Talent Matcher, which is referred to as the Non-curated Queue. This Non-curated Queue is ordered on the Talent Matcher's match score we described in Section 3.3; while deprecated, it is used in the context of the quickly served Non-curated Queue for lack of a better sorting property.

While the job seeker interacts with the Non-curated Queue, a Curated Queue is generated in the background. The first batch of six matches served comprise the Non-curated queue, and by the time the second batch of matches is served, the Curated Queue is available. The second, and following

batches will be retrieved from this Curated Queue.

It is necessary to recalculate the Curated Queue from time to time to include new content posted while the generated queue was stored. Therefore, the Curated Queue is stored for an hour, after which it is removed to allow a recalculation of the queue to be triggered. The curated queue can also be replaced by an updated version during its existence; triggers for such recalculations are discussed later. Figure 3.3 shows the flow of which queue is served to the job seeker.

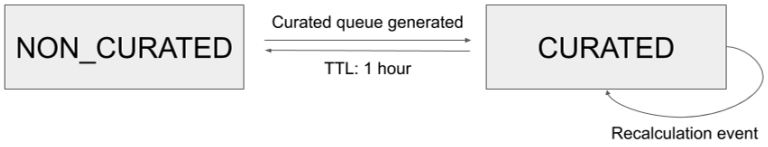

Figure 3.3: State diagram indicating which queue is served to the job seeker

3.4.4. Curated Queue Setup

The Matches Queue is built up from two underlying queues of matches: one consisting of opportunity matches, and one of company matches. We will refer to them as the Opportunity Queue and the Company Queue for easy reference. The Opportunity Queue consists of promoted[‡] opportunities and non-promoted opportunities. Per the business model, the promoted opportunities must be sorted before the non-promoted opportunities in the queue. The company queue consists only of promoted companies; non-promoted companies will not appear in the Matches Queue.

The Matches Queue is constructed by injecting companies from the Company Queue into the Opportunity Queue at set intervals. This is shown Figure 3.4. It shows the Opportunity Queue at the top, which itself consists of promoted opportunities and non-promoted opportunities; sorted in that order. The Company Queue is shown at the bottom of the figure. The Matches Queue is created by injecting companies at index two, and then every sixth rank in the queue. This means the company-to-opportunity-ratio is at most 1 on 5, but in practice we see that the ratio of available opportunities and companies is different; often there are not enough promoted company matches available to maintain this periodicity. In that case, the ranks where companies would have appeared are simply occupied by opportunities.

[‡]'Promoting' an opportunity or company is a paid service of Magnet.me that boosts the match in the Matches Queue.

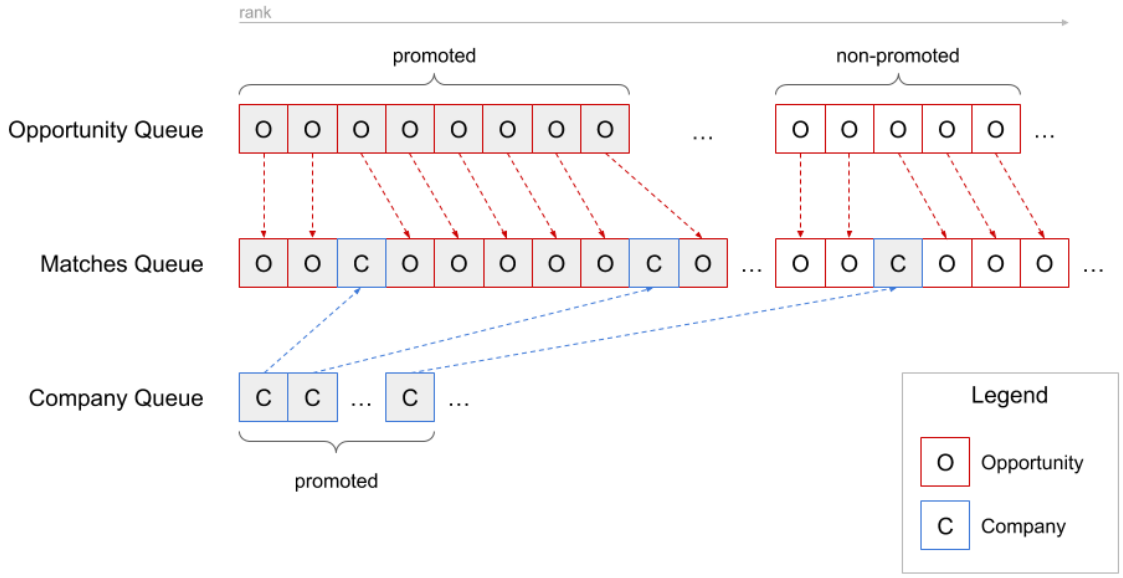

Figure 3.4: Curated queue setup. The Opportunity Queue (top, red) consists of both promoted and non-promoted opportunities that match with the job seeker. The Company Queue (bottom, blue) consists only of promoted, matching companies. The Matches Queue (middle) is acquired by injecting companies into the Opportunity Queue at set intervals.

Queue Order

The order of matches in the Matches Queue is dictated by the order of the matches in the respective underlying queues. When combining the queues to form the Matches Queue, the relative order from the Opportunity Queue and Company Queue is maintained.

As explained before, the business model dictates that promoted opportunities are shown before non-promoted opportunities, and only promoted companies are shown. The Company Queue is ordered by the match score of the Talent Matcher, since the impact of the match score is deemed insignificant on the sparse companies in the Matches Queue. The Opportunity Queue is ordered differently, because the match score provided by the Talent Matcher is not suitable for ordering opportunities in the queue. Instead the opportunities are shuffled. The problem, and the remedying shuffle, are described in Appendix B.

3.4.5. Overlap between Curated Queues

The curated queue can be recalculated during its one-hour lifespan, and it can be subject to small mutations. Several events, based on user interactions with the queue as well as events from elsewhere on Magnet.me, will trigger mutations or a recalculation of the queue. For an overview of events that affect the queue, see Appendix C.

Typical mutations or recalculations effectively remove one, or a small number of matches from the queue. For example, when a job seeker `ignores` a company in the queue, the opportunities posted by that company will not be suggested to the user anymore, assuming that the user would `ignore` these opportunities too.

Generally speaking however, queues remain very similar over these mutations or recalculations: Between the recalculations of the queues within its lifetime of one hour, the matches provided by the Talent Matcher do not change significantly. Furthermore, the deterministic shuffle mentioned above was written to minimise this perceived change in the queue.

This means that two queues generated consecutively, and within a short time span from one another, overlap: They show roughly the same content, in the same order. Effectively, this means that a job seeker that happens to trigger the recalculation of the queue they are watching over time is still interacting with the same content. This is important in evaluation of match relevance later on, because it allows us to obtain information on match relevance for a queue by looking at interactions with other

queues: If a match wasn't interacted with in one queue, it may very well have been interacted with in another.

3.5. Constraints on Matches Queue Improvements

The improvements we suggest related to the recommendations (RQ2), are subject to constraints imposed by Magnet.me, aimed at keeping the core product of Magnet.me unchanged during the experiment and protecting the user from possibly detrimental changes. Furthermore, the part of Magnet.me in which this experiment will be implemented is tied to very complex and old parts of Magnet.me, which restrict the extent to which the experiment can be implemented. These restrictions impact the validity of the experiment and the interpretations of the results. Those consequences will be discussed later on in the thesis, but the constraints will be described here to increase the comprehensibility of the rest of the thesis.

- Promoted opportunities must be sorted before the non-promoted opportunities in the Matches Queue.
- The content of the Matches Queue may not change: a job seeker must be offered the same opportunities and companies, which are provided by the Talent Matcher.
- The structure of the Curated Queue, meaning the periodicity of opportunities and companies, must remain as described in Section 3.4.4.
- The User Interface (i.e. the structure of the visual elements on the page) may not be changed.
- The User Experience, specifically regarding the loading time of the page, may not be changed.

These restrictions still allow us to improve on the *order* of matches in the Matches Queue, while respecting the periodicity of the queue from 3.4.4. This is possible by ordering the matches in the respective Opportunity- and Company Queues before combining both queues into the Matches Queue.

The constraints on user interface and user experience are not an issue when we propose and evaluate improvements to the sorting algorithm; as suggested in Section 2.3, we should keep all aspects of the web platform identical except what we are trying to test; otherwise our results may be affected.

3.6. Focus on Opportunity Queue

As explained in Section 3.4.4, the Curated Queue is built up by combining the Opportunity Queue and Company Queue. In this thesis, we focus on improving (the ordering of) the Opportunity Queue.

The first reason to look at the Opportunity Queue only is the goal to use job seeker interactions with the platform to improve the ranking of the queue. Job seekers look significantly more at opportunities than they look at companies; there were almost six times as many views of opportunity pages compared to company page views, and the average duration of an opportunity page view is three times as long as a company page view. The sum of durations of opportunity page views is 16.5 times longer than that of the company page views. See Table 3.1 for the absolute statistics of company page views versus opportunity page views.

Viewed match type	Number of views	Total duration	Average duration
Company	47283 views	823 hours	1 minute
Opportunity	281005 views	13660 hours	3 minutes

Table 3.1: A comparison of view statistics of company page views and opportunity page views.

The second reason to only focus on the Opportunity Queue is the structure of the Matches Queue: There are five times more opportunities than companies in the Matches Queue due to the interlacing of the Opportunity- and Company Queue.

This means that the feedback that can be gathered on the Company Queue as a part of the Matches Queue is small: Job seekers, on average, interact with between 5 and 15 matches before navigating

to another page on Magnet.me, or dropping off entirely. That means that the average job seeker sees at most three companies in the Matches Queue, yielding very little feedback compared to the up to ten opportunities they view.

Finally, the amount of content for respectively the opportunity- and company queue means that improving the Opportunity Queue ordering has more potential impact than ordering the Company Queue. Due to the user dropping off the Matches Page, they miss a lot of opportunities in the queue. The goal of the Matches Queue is to provide the job seeker with matches that are interesting for them; an improved sorting algorithm could increase the relevance of the content in the front of the queue.

In conclusion, we have introduced the job recommendation platform Magnet.me. We have shown how a job seeker experiences the job recommendations, and interacts with them. In the job seeker's job recommendation experience, we have highlighted areas that can be improved, while abiding to the constraints imposed by Magnet.me

In the next chapter, we will further investigate how the job seeker interacts with Magnet.me, and how we can use these interactions to gain a better understanding of the job seeker's preferences, eventually to improve the job recommender system.

4

Deriving Preferences from User Behaviour

In Chapter 3, we discuss the job seeker's explicit preferences. These consciously provided preferences are used to filter employers and opportunities in the Talent Matcher algorithm described in Section 3.3. While the explicit job seeker preferences suffice to filter content on Magnet.me on, they fail to indicate relative preferences: For example, a job seeker may indicate a preference in industries 'government' and 'education', but which do they like *better*?

During their sessions on Magnet.me, job seekers explore employers and opportunities. They go through the matches on the Matches Page, they visit pages dedicated to the employers, or to their opportunities, and eventually correspond with recruiters and apply to opportunities. Through their interactions with Magnet.me, job seekers express behaviour that may provide a better understanding of their preferences; more fine-grained than the explicit preferences indicated in their profile.

In this chapter, we focus on RQ1: *How can we gain a better understanding of job seeker preferences, beyond their explicitly provided preferences?* To do so, we discuss monitoring user behaviour, and analyse job seeker's interactions with opportunity pages on Magnet.me. We then look for existence of patterns that support the intuitive notion of preferences expressed in their behaviour. Then, we propose an aggregation of behaviour aimed towards a better understanding of their preferences, specifically to predict a stronger affection to one property value than another.

We also focus on RQ2: *How can we use implicit job seeker preferences to improve the opportunity recommendation ordering algorithm?* We do so by using the aggregation of implicit preferences to propose an algorithm to sort the Opportunity Queue for the Matches Page.

4.1. Monitoring User Behaviour

RQ1a asks what job seeker interaction data we can monitor to derive implicit preferences from. Magnet.me historically tracked user behaviour on an abstract event basis: ignores, likes and applications, for example. While raw navigational logs are available, they were not retained for long due to their sheer volume: gigabytes of such data are generated every day. More importantly, combining this data into an understanding on how users interacted is very challenging; getting a complete image of what a user did from arriving on a page until they left is near unfeasible. Even if it were feasible, it was not possible to look far back because navigational data is not available for the more distant history.

To be able to study behaviour for implicit preferences, we need to gain a better understanding of how job seekers interact with pages on Magnet.me. We therefore implemented a more behaviour-oriented monitoring system that tracks page interactions from the moment the page is loaded to the moment the user leaves. The user 'leaving' is defined as one of: the user navigates to another page on Magnet.me, the page is not visible anymore (based on the Page Visibility API [50]), or the user closes the browser (tab). For each page view, we store the following information:

- Time-related details, such as the start- and end time of the page visit and the duration.
- Information on the entity on the page, if applicable.

For example: on the page of a specific opportunity, we can include detailed information on the opportunity, but on the search page for all opportunities such information is not available.

- Information on the relation between the job seeker and the entity on the page, if applicable.

  Like explained above, this is only useful on pages showing an entity the job seeker can relate with, i.e. opportunities and companies.

- Actions and interactions performed by the job seeker during the page visit.

  While on the page, the job seeker interacts with the page in two ways: interactions with the page itself, such as clicking buttons and scrolling the page, and interactions with the entity shown on the page, such as (un)liking and applying to the shown opportunity. These actions are stored with time stamps of their own, as well as information specific to the type of action: e.g. for a scroll event, we store how far the user has scrolled.

These page views and -interactions are tracked for job seekers on all pages on Magnet.me, and are referred to as 'page interactions'. Since we only focus on improving the opportunity queue as explained in section 3.6, we will also focus our study of user behaviour on the opportunity pages in this thesis. Therefore, from here on, when we refer to page interactions we refer to interactions with opportunity pages.

4.2. Implicit Preferences Shown In User Behaviour

Now that job seeker behaviour on Magnet.me is tracked, in the form of page interactions, we can study this data for behavioural patterns that could indicate implicit preferences. To find out whether job seekers show preferences in their behaviour, we will use one of the actions that the job seeker exercises during opportunity page interactions: indicating their interest in the opportunity. In the page interactions, a change in the job seeker's interest in the opportunity is denoted by the property `newInterestedStatus`. This is a clear indicator of affection towards the opportunity, and we will use this as the ground truth for our page interactions analysis.

Using this ground truth, we will analyse if properties of the page interactions are good predictors for a change in their interest status for the viewed opportunity. For example, based on the related work one could expect that when a job seeker shows a longer attention span for an opportunity page, it could indicate a positive affection for the opportunity. We will look for these patterns using a logistic regression. In this section, we discuss how we pre-processed page interactions and derived a feature set.

4.2.1. Pre-processing the Page Interaction Data

We must first pre-process the data to analyse the data for patterns. The interactions already contain the property `newInterestedStatus`, indicating the affection of the user to the opportunity. From the start- and end time stamp, we derive the duration of the session. From the events (i.e. the job seeker's actions), we derive three numerical variables: the number of events, the duration between the start of the of the interaction and the last event, and the duration between the first and the last event.

Below is the full list of variables of the hypothetical feature set for the logistic regression analysis.

- `newInterestedStatus`: Logical. The new status of interest the job seeker expressed during the page interaction. The value is `true` or `false`.
- `duration`: Numerical. The duration of the page interaction in milliseconds.
- `eventCount`: Numerical. The number of page interaction events (click or scroll).
- `startToLast`: Numerical. The time between the start of the page interaction and the last event in milliseconds.
- `firstToLast`: Numerical. The time between the first event and the last event in milliseconds.

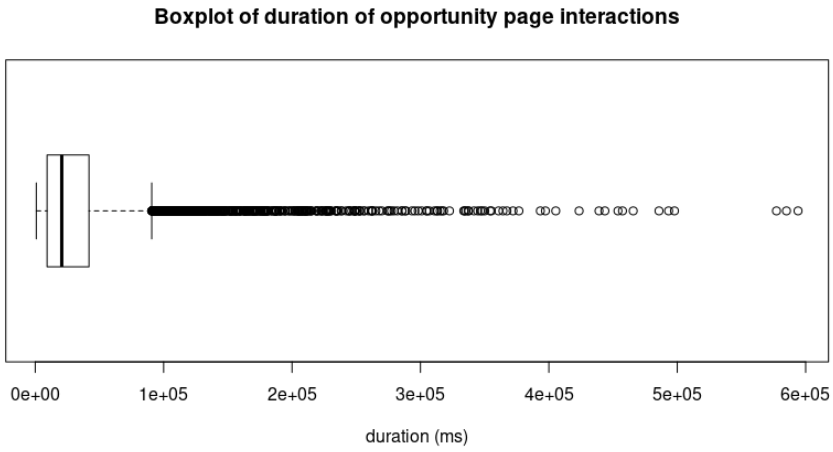

Figure 4.1: Boxplot showing the long-tailed nature of the duration of sessions on dedicated opportunity pages. The median is at 20,643ms, the first quartile is at 9,205ms and the third quartile at 41,859ms. The whisker at 1.5IQR above the median is at 90,726ms.

4.2.2. Selected Page Interaction Data

We take page interactions gathered between [redacted] and filter out page interactions during which the job seeker indicated a new status of interest on a dedicated opportunity page. This resulted in a total of 9,978 page interactions. The duration data is quite long-tailed, as is shown in the boxplot in Figure 4.1. We see that the right whisker in the boxplot at 1.5IQR above the median lies around 100,000 milliseconds, and duration values above that are considered outliers. From these page interactions, we removed the outliers: interactions that have a duration higher than 100,000 milliseconds, or about 1.67 minutes. This is equal to 5.6% of the dedicated opportunity page interactions. The data we use for our analysis consists of the remaining 94.4% or 9,422 page interactions.

4.2.3. Data Exploration

After pre-processing the data and removing outliers, we have a dataset from which we want to derive a feature set for the logistic analysis. We can quickly see what patterns are emergent by studying the correlation matrix shown in Table 4.1, and corresponding scatterplot matrix shown in Figure 4.2.

Both the correlation matrix and the scatterplot matrix clearly show that all duration-based variables; `duration`, `startToLast` and `firstToLast`, strongly correlate. `duration` is nearly perfectly correlated with `startToLast`, the time from the start of the page interaction to the last event in the interaction; of course, this is not entirely surprising. For example: if a user typically ends the page interaction shortly after the last interaction event (e.g. a click), that would explain this correlation.

The same applies to `firstToLast`, but we see something peculiar in the scatterplots between `firstToLast` and two other variables; `duration` and `startToLast`. Both show a strong grouping in the form of a line for very low values of `firstToLast`. These are caused by page interactions for which there were very few events, which were prevalent in the data set: 3,115 of 9,422 page interactions had only a single click or scroll occurring.

The correlation matrix shows a correlation between the `newInterestedStatus` and `duration` of 0.1904546; there is a (weak) positive correlation between the duration of a page interaction and the positive affection of the job seeker towards the opportunity. This is not distinguishable in the scatterplots, because the data points for `newInterestedStatus` overlap very strongly in the plot.

In the correlation matrix, we see a similar correlation between `newInterestStatus` and `eventCount`. The correlation has value 0.1666328. Though even weaker, it is comparable to the correlation of `newInterestStatus` and `duration`.

From the correlations in Table 4.1, we can conclude that the duration of page interactions could be a predictor of the affection of a job seeker towards an opportunity. The `eventCount` shows a similar but weaker correlation.

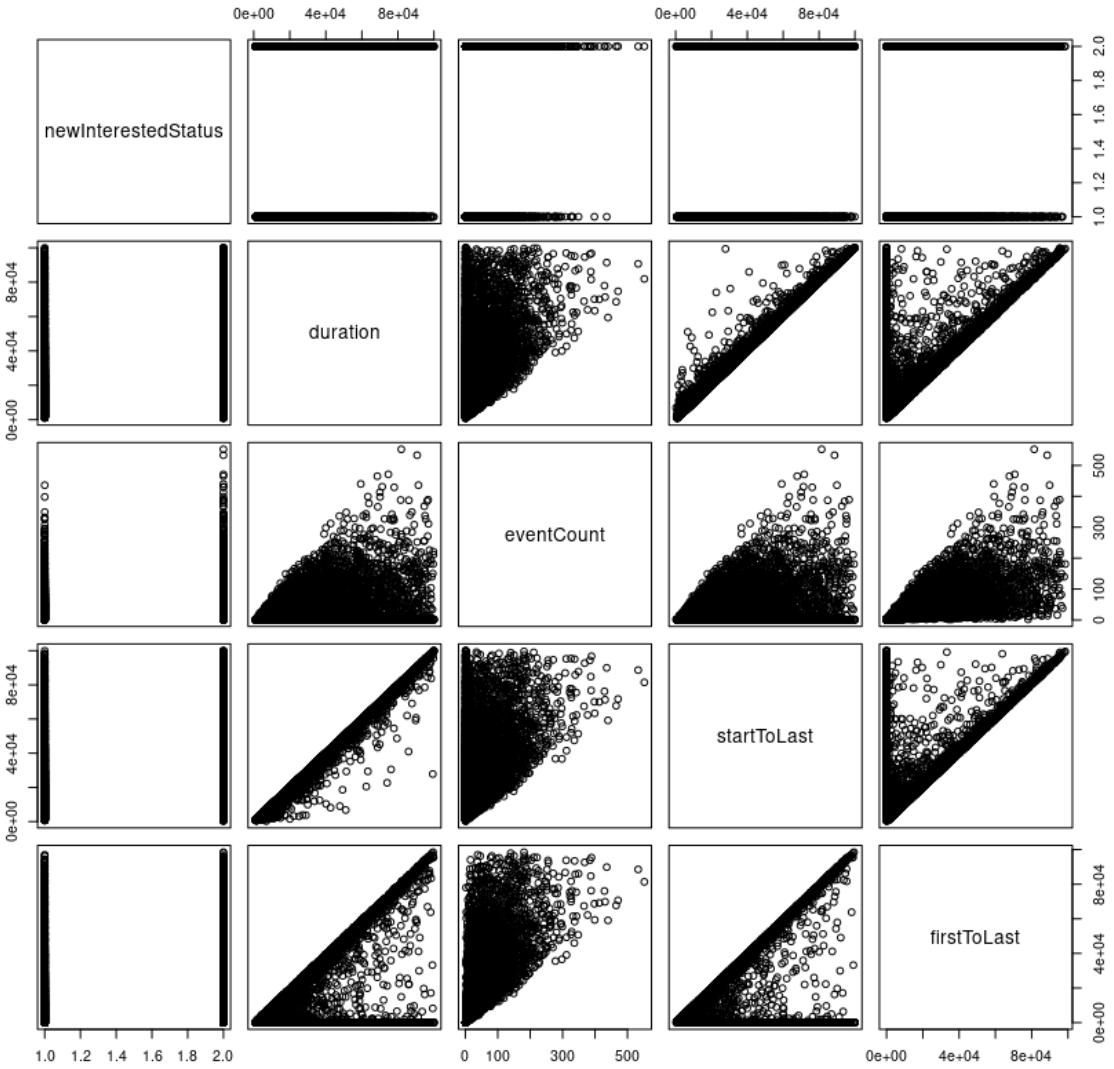

Figure 4.2: Scatterplot matrix showing scatterplots for all variables in the page interactions data. The logical `newInterestedStatus` variable has been converted to a numerical value: 'false' is represented by value `1.0` and 'true' by `2.0`.

	newInterestedStatus	duration	eventCount	startToLast	firstToLast
newInterestedStatus	1.0000000	0.1904546	0.1666328	0.1892302	0.1616041
duration	0.1904546	1.0000000	0.4378709	0.9932156	0.6622901
eventCount	0.1666328	0.4378709	1.0000000	0.4340881	0.7335792
startToLast	0.1892302	0.9932156	0.4340881	1.0000000	0.6569243
firstToLast	0.1616041	0.6622901	0.7335792	0.6569243	1.0000000

Table 4.1: Correlation matrix for the pre-processed page interactions data.

Feature Selection

Due to the high correlation between `duration`, `startToLast` and `firstToLast`, we remove the last two variables from the feature set. The variables used for the logistic regression analysis are therefore:

- `newInterestedStatus` as the dependent, logical variable.
- `duration`: Numerical. The duration of the page interaction in milliseconds.
- `eventCount`: Numerical. The number of page interaction events (click or scroll).

Logistic Regression

To determine whether `duration` and `eventCount` are significant predictors for the `newInterestedStatus`, we use a logistic regression analysis. We construct a model with `newInterestedStatus` as the dichotomous response variable and `duration` and `eventCount` as the predictor variables, and fit it on the data. The regression analysis can be found in Listing 4.1.

```
1
2  Call:
3  glm(formula = newInterestedStatus ~ duration + eventCount, family = binomial,
4      data = pageIntFact)
5
6  Deviance Residuals:
7      Min       1Q   Median       3Q      Max
8  -2.3199  -1.0543  -0.9537   1.2169   1.4337
9
10 Coefficients:
11               Estimate Std. Error z value Pr(>|z|)
12 (Intercept) -6.004e-01  3.318e-02  -18.10   <2e-16 ***
13 duration     1.354e-05  1.078e-06   12.57   <2e-16 ***
14 eventCount   4.548e-03  4.954e-04    9.18   <2e-16 ***
15 ---
16 Signif. codes:  0 '*** 0.001 '** 0.01 '* 0.05 '. 0.1 ' ' 1
17
18 (Dispersion parameter for binomial family taken to be 1)
19
20     Null deviance: 13037  on 9421  degrees of freedom
21 Residual deviance: 12599  on 9419  degrees of freedom
22 AIC: 12605
23
24 Number of Fisher Scoring iterations: 4
```

Listing 4.1: Summary of the model fitted on the page interactions data.

The fitted model shows that `duration` and `eventCount` are both significant predictors of the dependent variable `newInterestedStatus`. However, the model leaves much to be desired. See the analysis of variance in Listing 4.2: the combined predictor variables only explain $\frac{13037-12600}{13037} * 100\% = 3.3\%$ of the deviance in the page interactions data. We can conclude that the model does not fit the data very well. This urges caution in using `duration` and/or `eventCount` of opportunity page interactions as predictors for affection of the job seeker towards the opportunity.

```
Analysis of Deviance Table

Model: binomial, link: logit

Response: newInterestedStatus

Terms added sequentially (first to last)


           Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                        9421      13037
duration    1   347.12       9420      12690 < 2.2e-16 ***
eventCount  1    90.01       9419      12600 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Listing 4.2: Analysis of Variance test for the logistic regression model newInterestedStatus duration + eventCount.

Despite the weak correlation and the poorly fitted model, we have shown a connection between `duration` and `eventCount` of opportunity page interactions by job seekers with the job seekers affection towards the opportunity. This is relevant for RQ1a. We will cautiously use these connections to aggregate behaviour into an improved and more detailed understanding of job seeker preferences towards opportunities.

4.3. Aggregating User Behaviour to Derive Implicit Preferences

Using the conclusions about the connection between page interaction properties and job seeker affection towards opportunities, we look more closely at deriving implicit preferences from the job seeker behaviour. To do so, we will now look away from the page interactions to what is actually displayed on the page: an opportunity. By doing so, we shift our focus to RQ1b: 'How can we aggregate implicit job seeker preferences from job seeker interaction data?'.

To reiterate: an opportunity is for example an internship or a job, and every opportunity is defined by a number of properties. These properties consist of the job type, job function, employment type, minimal education level, salary, location of employment, and several properties of the employer: company size and industry.

In this section, we propose a model of implicit preferences that employs a proposition about the affection of job seekers towards an opportunity: We observe that, if a job seeker is interested in an opportunity, they are interested in the properties of that opportunity as well. To refrain from leaning too heavily on the logistic regression model above, and the proposition we just made, we will be cautious in our data selection.

4.3.1. Conservative Data Selection

If we we gather all opportunity page interactions for a job seeker, we have an overview of what the job seeker has sought out to view, as well as meta-information about the page interactions. We can append the opportunity properties to the page interactions, and analyse this augmented information to derive implicit preferences from job seekers. We will formally propose an algorithm to do so later in this section. However, we want to tread carefully with applying the weak conclusions from last section: even though we've shown page interaction properties can suggest positive affection towards the opportunity, the logistic regression model did not fit well.

Therefore, before we apply the algorithm to distil a usable model of implicit job seeker preferences, we apply a filter on the page interactions from which we derive them. Instead of using all opportunity page interactions for a job seeker, we only use page interactions with opportunity pages for which we know the job seeker is interested in the opportunity. By applying this filter, we build our model of the job seeker's preferences more conservatively, focusing on more explicit feedback from the job seeker.

Because of this conservative approach, we significantly constrain the amount of data available to derive implicit preferences from. If we look at all dedicated opportunity page interactions from the period described in Section 4.2.2, the data is divided as shown in Table 4.2. By selecting only page interactions with `interest=true`, we remove all but 44.1% of available data.

To use the rest of the data in implicit preference derivation, we consider more research is required to be confident about the applicability of the data.

Interest Status	Number of page interactions	Percentage of total
Interested	182.829	44.14%
Not interested	23.779	5.74%
Unknown	207.557	50.11%
Total	414.165	100.0%

Table 4.2: Page interactions on dedicated opportunity pages, grouped by interest status.

4.3.2. Deriving Implicit Preferences

Using a job seeker's opportunity page interactions, and augmenting them with the properties for the displayed opportunities, we can use the positive affection that is suggested by higher page interaction duration. We have decided not to use the `eventCount` due to the complexity it would add to the implicit preference model we propose here; this variable could be considered in future work. To employ the correlation between `duration` and opportunity affection, we apply a conceptual observation on the page interaction data: if a job seeker has read an opportunity page with a duration d, then it has also studied the opportunity's properties with duration d. We then apply the positive affection that is suggested by longer durations, and make the following proposition: *The longer a job seeker has looked at an opportunity, the more interested they are in that opportunity's properties.* These are the implicit preferences of the job seeker for properties of viewed opportunities.

Using that proposition, we can aggregate the job seeker's implicit preferences by summing the duration of each view, per property of the opportunity. The result from this is a collection of all possible values of opportunity properties, and total duration per value. For example, if the job seeker is interested in engineering jobs, and lesser so in jobs in education, we expect to see a higher total duration for the job function 'engineering' than for the industry 'education'. To be able to compare the scores of all different properties, we normalise each property score by dividing the duration for the property by the total duration of the job seeker's opportunity page interactions.

Formally defined, the set of opportunities viewed by the job seeker is O, and the set of all possible opportunity properties is P. $P_o \subset P$ is then the set of properties of opportunity $o \in O$. We define t_o as the time spent watching opportunity $o \in O$. Then we define T as:

$$T = \bigcup_{p \in P} t_p \tag{4.1}$$

where t_p, normalised for the total view time, is:

$$t_p = \sum^{\langle o \in O | p \in P_o \rangle} \frac{t_o}{t_{total}} \tag{4.2}$$

where t_{total} is:

$$t_{total} = \sum^{o \in O} t_o \tag{4.3}$$

The proposed model T will be used in our online experiment to help us answer RQ1. Using this model of implicit preferences, we now propose an alternative way to sort opportunities in the Opportunity Queue.

4.4. Sorting Opportunities based on Implicit Preferences

RQ2 asks how we can improve the sorting of the Opportunity Queue with our newly acquired aggregation of implicit preferences, T. The algorithm we propose here will link T to opportunities in the Opportunity Queue, and allow us to sort them. Our high-level approach to sorting the Opportunity Queue according on the job seeker's implicit preferences is the following:

1 Award a numeric score to each property of each opportunity in the queue based on the set of implicit preferences T of the job seeker.

2 Reduce the set of scores for an opportunity to a single score.

3 Sort the opportunities based on their numeric score, descending.

Formally defined, the Opportunity Queue for a job seeker consists of a set of opportunities O_q obtained from the Talent Matcher. All opportunities $o_q \in O_q$ have properties $p_o \in P$, where P is the collection of all possible opportunity properties. Using the job seeker's normalised implicit preference scores T, consisting of preferences t_p per property $p \in P$, we award a score s_p to all properties p_o of all o_q with t_p:

$$s_p = t_p \tag{4.4}$$

Therewith we have awarded a numeric score to each property of the opportunities.

Step two of our algorithm is to reduce the set of scores per opportunity to a single score. We have chosen to perform this reduction by taking the maximal score s_p of all properties $p_o \in P_o$. For an opportunity $o_q \in O_q$ and the set of properties P_o for opportunity o_q, the opportunity score s_o is then defined as:

$$s_o = \max_{p \in P_o} (s_p) \tag{4.5}$$

In other words, the score awarded to an opportunity o is the highest normalised view time of all properties of opportunity o.

We apply this scoring algorithm to all opportunities in the Matches Queue for the job seeker, and sort the queue descending on this score. In effect, this means that the opportunities that have the property that the job seeker viewed the longest is shown to them first. This will help us answer RQ2. To see how much our algorithm improves Opportunity Queue performance, we go into sorting algorithm evaluation in the next chapter.

5

Relevance Evaluation

In this thesis, our aim is to improve the ranking algorithms of Magnet.me's Matches Queue. To do so, we must find a way to measure the performance of the ranking algorithms. RQ3 focuses on this: How can we evaluate rank performance, and therewith distinguish between ranking algorithm performance? We look at RQ3a, which is aimed at how we derive relevance from queue interactions, and at RQ3b which asks how we should handle missing feedback.

In this chapter, we first motivate our choice for the rank evaluation metric NDCG. We then introduce 'relevance evaluations', which is how we apply relevance scores to queue interactions. We propose several properties of relevance evaluations that will further help us answer RQ3 with our online experiment. Finally, we combine the relevance evaluation properties to compile a number of relevance evaluations which we will compare in an online rank evaluation experiment.

5.1. Rank Evaluation Metrics

As discussed in Section 2.2, there are many rank evaluation metrics we can use to compare performance of ranking algorithms. For our experiment, we evaluate our rankings using the Normalised Discounted Cumulative Gain (NDCG) introduced in Section 2.2.2; NDCG handles multiple levels of relevance, which means we can consider different queue interactions to have different relevance.

In the rest of this chapter, we will propose varying relevance evaluations that will yield different NDCG scores. To draw conclusions on the different NDCG scores, we also use the Mean Reciprocal Rank (MRR) as a crude baseline.

5.2. Applying Relevance Scores to Job Seeker Match Interactions

NDCG's capability of handling non-binary relevance scores means that we must find a way to derive relevance for job seeker interactions with the queue. We will refer to a such mapping as a 'relevance evaluation': the mapping evaluates the relevance indicated by an interaction and applies a score to it.

The relevance feedback from a job seeker for an opportunity is one of three explicit actions: `like`, `ignore` and `apply`. When no feedback is available for an opportunity, we call this interaction `unknown`. A relevance evaluation is then a set of relevance scores s_i for all interactions i, where i is one of `unknown`, `ignore`, `like` and `apply`. Formally defined, this means a relevance evaluation R is defined as, with $I = \{\texttt{unknown}, \texttt{ignore}, \texttt{like}, \texttt{apply}\}$ and s being a numerical relevance score

$$R = \bigcup_{i \in I} s_i \tag{5.1}$$

We use this model in our online experiment to answer RQ3a; the experiment will show what relevance evaluations are usable. How we handle missing feedback, the subject of RQ3b, is studied with how `unknown` interactions are handled in the relevance evaluations. There are many possible relevance evaluations, which we will introduce in the next section.

5.3. Relevance Evaluation Properties

We now propose several properties of relevance evaluations, i.e. how we score the relevance of types of match interactions of job seekers. We will use these properties to denote categories of relevance evaluations which we will compare in our experiments.

We first introduce different restrictions on the score ranges of the relevance evaluations; we motivate using non-zero lower bounds for DCG, and with these different score ranges, we will test whether we can calculate NDCG with non-zero lower bounds.

Then, we will introduce fairness, which indicates the order of relevance of interactions. We propose different orders of interactions (RQ3a), in which we also handle missing feedback (RQ3b).

Finally, we introduce different numbers of relevance levels. This property is introduced to discern between how many different scores the relevance evaluation uses. We use different numbers of levels to research the influence of adding more levels on NDCG's capability to distinguish between algorithm performance.

5.3.1. Score Ranges

As described in section 5.3.1, we normalise DCG scores allowing for non-zero lower bounds of DCG. The definition of NDCG as given in Section 2.2.2 assumes that the lower bound of DCG equals zero. Here, we motivate using different lower bounds of the DCG in evaluating ranking algorithm performance. To calculate NDCG with these non-zero lower bounds, we must change the way NDCG is calculated slightly. After we have proposed this changed calculation, we propose three different lower bounds, which will be denoted as a relevance evaluation property 'score range'.

Consider the relevance feedback that is received for the Matches Queue, to which we apply a score. Traditionally, NDCG implementations use value 0 for the lowest relevance level. Effectively, this means that the interactions with that level of relevance are considered not to have any gain; they did not satisfy the job seeker's desired interests. However, when we consider the difference between a job seeker actively choosing to `ignore` an opportunity, and a job seeker that does not respond to an opportunity, one can imagine that the `ignored` opportunity might be even *less* relevant than the opportunity for which no feedback was provided. Further even, we could say that `ignore` should have a *negative* gain, and `unknown` a more 'neutral' gain.

DCG with a non-zero lower bound

The definition of NDCG from Section 2.2 does not handle non-zero lower bounds of DCG: the normalisation would fail. If we want to use relevance evaluations with a lowest relevance score that is not equal to zero, we must calculate NDGC differently; we must drop the assumption that the lowest relevance equals zero. The assumption of DCG having the lower bound zero is shown in the normalisation step of the NDCG. This normalisation of the performance of a rank is calculated as $\frac{\text{DCG}}{\text{IDCG}}$, where IDCG is the DCG score for the optimally sorted rank: sort descending on the gain per item (Ideal Discounted Cumulative Gain). This calculation is a simplification of min-max re-scaling: a form of normalisation calculated for a generic variable x as $\frac{x-\min(x)}{\max(x)-\min(x)}$. In the case of the standard NDCG normalisation, $\min(x)$ is assumed to equal zero, which yields the calculation $\frac{x}{\max(x)}$. The normalisation then does not map the score to range $[0,1]$, but to $[\min(\text{DCR}_r),1]$[‡].

The solution to this problem is to calculate the normalisation with the non-simplified $\frac{x-\min(x)}{\max(x)-\min(x)}$. When applied to NDCG, this becomes

$$\text{NDCG} = \frac{\text{DCG} - \text{WDCG}}{\text{IDCG} - \text{WDCG}} \tag{5.2}$$

where we introduce WDCG: The Worst Discounted Cumulative Gain. It is calculated by sorting the items *ascending* on relevance gain, and calculating the DCG for that; the opposite of the IDCG.

We can check whether the implementation of Formula 5.2 behaves like the simpler definition in Section 2.2, by implementing relevance evaluations that have the same order of interactions, but differ in the value of the scores applied to the interactions.

Therefore, we propose the following score ranges

[‡] $\min(\text{DCR}_r)$ is the lower bound of the DCR calculated using rank evaluation r

- non-restricted relevance evaluations, for which all relevance scores $s \in \mathbb{Z}$[‡]
- non-negative relevance evaluations, for which all relevance scores $s \geq 0$
- strictly positive relevance evaluations, for which all relevance scores $s > 0$

It should be noted that the simple definition of NDCG thus requires that the relevance evaluations have a non-negative relevance evaluation. The relevance evaluations we will propose will have different score ranges, which will allow us to test if we could indeed use relevance evaluations with a non-zero lower bound. We expect to see that the NDCG scores for relevance evaluations different score ranges, but other identical fairness and number of relevance levels, yield identical results; after all, nothing has changed in the order of relevance of the interactions.

5.3.2. Fairness

The order of relevance of the different types of interactions defined in Section 5.2 strongly influences the NDCG score; it is therefore important to choose this order with great care. RQ3a and RQ3b research this: how do we derive relevance from feedback, and how should we handle missing feedback?

For explicitly given feedback, the natural order of relevance is fairly clear; an opportunity that the job seeker `likes` is more relevant than an opportunity that is `ignored`. A direct application to an opportunity definitely shows it is relevant, and one could argue that the gain score for an `apply` should be even higher than for a `like`.

The score for the `unknown` interaction, however, is more difficult to assign a value. One could argue that this interaction should not be attributed a positive gain score, because the match cannot be considered relevant. Another argument could be made that, while not relevant, such a match should have a higher gain score than an explicitly `ignored` match, because this interaction specifies a low relevance whereas we may not be able to conclude the same of the `unknown` feedback. Regardless of motivation, the decision is very important; it dictates the ideal sorting of the queue, and that strongly influences the NDCG score.

To discern between the different interaction orders, we introduce the property 'fairness'. Using the definition for a relevance evaluation from Equation 5.1, we consider a relevance evaluation to be 'fair' when $s_{\texttt{ignore}} < s_{\texttt{unknown}} < s_{\texttt{positive}}$. A relevance evaluation is considered to be 'unfair' when $s_{\texttt{unknown}} < s_{\texttt{ignore}} < s_{\texttt{positive}}$. With this property, we can express which we find more relevant: the `unknown` interaction or the `ignore`. Table 5.1 shows examples of fair and unfair relevance evaluations.

Interaction	Fair	Unfair
`like`, `apply`	2	2
`unknown`	1	0
`ignore`	0	1

Table 5.1: An example of a fair- and an unfair relevance evaluation.

We chose the term 'fairness' based on the intuitive appeal to giving a higher gain to an `unknown` interaction than to an explicit `ignore`. Of course, the `unknown` interaction does not have a hierarchical relation to the `ignore`- and positive interactions, so the term serves mostly as a way to discern between the proposed categories, and not as an indicator of actual fairness. Because fair- and unfair evaluations use both possible ways to handle missing feedback, we can use them to answer RQ3b.

5.3.3. Number of Relevance levels

NDCG can handle multiple levels of relevance. To discern between the number of different levels of relevance, i.e. different numbers of unique scores in a relevance evaluation, we propose 'binary', 'ternary' and 'quarternary' relevance. We will discuss their implications on the NDCG score, and motivate what their advantages and disadvantages are.

[‡]One could even use $\mathbb{R}$, but we will stick to integers in this thesis.

Binary Relevance

As the name suggests, binary relevance only has two different score values; it is used to simply discern between relevant and non-relevant matches. Wang *et al.*'s Standard NDCG relevance scores are an example of a binary relevance score.

We propose binary relevance evaluations to compare them to the relevance evaluations with more differentiation in relevance. The low number of relevance levels means that there are less possible permutations for the sorting of a queue; therefore binary relevance evaluations provide more crude insights in the performance of queues. Since the evaluation does not discern between the `ignore` interaction and the `unknown` interaction, we do not label it as fair or unfair.

Ternary Relevance

With three levels of relevance, ternary relevance evaluations allow us to split up the positive- and non-positive interactions from the binary relevance evaluations. We expect splitting up the non-positive interactions, `unknown` and `ignore`, to have significantly more effect on the NDCG score than splitting up the positive interactions `like` and `apply`. Therefore we will only test ternary relevance evaluations that have different values for s_{unknown} and s_{ignore}; all ternary relevance evaluation will therefore apply the same score to `like` and `apply`.

Because different scores can be applied to the `ignore`- and `unknown` interactions, we can create relevance evaluations that are fair or unfair. We also use different score ranges, as discussed in Section 5.3.1. Combining these options, we will propose several ternary relevance evaluations that will combine different score ranges, and fairness.

Quarternary Relevance

Quarternary relevance gives us a fourth layer to discern between even more layers of relevance. We use the fourth layer to not only distinguish negative, unknown and positive relevance; we also distinguish between the positive interactions `like` and `apply`. Specifically, we consider `apply` to suggest higher relevance than `like`, so we assign a higher value to s_{apply} than s_{like}.

With this, we specify that the ideal queue would have all opportunities to which the job seeker directly applied at the start of the queue; then the `liked` opportunities, and the non-positive interactions in the tail. While this 'ideal' queue naively sorts higher relevance at the top, the real queue interactions will rarely show that job seekers `apply` to the first opportunity they see on the Matches Page. Furthermore, when the job seeker applies to a job, they are navigated away from the Matches Page: to actually apply, the job seeker must send a message to the employer. This means that multiple applications in one queue are very rare; the job seeker would have to navigate back to the queue manually and find another opportunity they want to apply to. Because of this, the quarternary relevance revaluations suggest a somewhat unrealistic ideal queue which will never be fully achievable in the real world.

However, we consider applications to be a far stronger indication of positive relevance than a `like`. It would be unwise to ignore this distinction altogether by considering them equal, like for the other numbers of relevance levels. Furthermore, differentiating between like relevance and application relevance will reward sorting algorithms that put the applied-to opportunity in the beginning of the queue over those who fail to do so.

Compared to the less complex binary- and ternary relevance evaluations, the many different levels of relevance in these evaluations may hurt the distinguishing capability of NDCG In other words, we expect that quarternary relevance evaluations may not decidedly point at a winning sorting algorithm.

In conclusion, we expect that added levels of relevance will create more room for error. This may cause a drop in scores for relevance evaluations as more relevance layers are added. Furthermore, the added layers may hurt the distinguishing property of NDCG.

5.3.4. Cutoff points

Described in Section 2.2.2, related work suggests that we should only consider the first k documents in the result set. This cutoff point is used to mitigate the influence of large 'tails' of ranks that have not been evaluated by the job seeker; i.e. there is no relevance feedback available. This score can skew the score unfairly because a relevance gain will be awarded to matches which the job seeker has not considered.

For the Matches Queue, we know job seekers are required to go through the queue from beginning to end, without skipping a match. However, when gathering relevance feedback to calculate NDCG for a queue q_{eval} we also consider feedback given in other queues, because queues generated for a job seeker overlap significantly; this was explained in Section 3.4.5. This means that a queue is typically structured like this: the job seeker explicitly gave feedback on the first opportunities in q_{eval}, and gave feedback on opportunities further in q_{eval}, but in *another* queue. What we see in a typical queue is that the first opportunities have been judged by the job seeker. What follows is a long tail of opportunities for which no feedback is available, i.e. an `unknown` interaction. In this tail, there may be some sparse opportunities for which feedback is available; this feedback was given by the job seeker in another queue that also contained that specific opportunity.

When evaluating the queue's rank, the long tail can skew the NDCG score: If the ideally sorted queue would have all `unknown` interactions at the back of the queue, then this overlaps strongly with the typical queue interactions. This would skew the score of the queue rank based on the `unknown` interactions, while we want to evaluate the sorting of the opportunities that were judged by the job seeker.

Therefore, in line with the related work, we will choose a cutoff point k based on the number of matches that a job seeker actually sees.

5.4. Chosen Relevance Evaluations

The relevance evaluations chosen to evaluate rank performance are based on the properties described in Section 5.3: fairness, number of relevance levels, and score range. In this section, we introduce a set of relevance evaluations with these properties. Per relevance evaluation, we discuss what their properties are and how they influence the expected NDCG scores.

5.4.1. Binary Evaluations

First off, we have chosen three binary relevance evaluations: they are shown in Table 5.2. The first evaluation consists of the non-negative 'Standard NDCG scores': 0 for non-relevant and 1 for relevant interactions. For the non-restricted score range that includes negative numbers, we lower the standard scores by one, giving us the scores -1 and 0; in effect, this score acts as a punishment for non-relevant interactions rather than a gain for relevant interactions. For the positive interactions we increment the standard scores by one, giving us 1 and 2 respectively; both interaction levels are awarded a gain, but the amplitude of the gain is different.

For all binary evaluations, $s_{\text{ignore}} = s_{\text{unknown}}$ and $s_{\text{like}} = s_{\text{apply}}$, and finally $s_{\text{non-relevant}} < s_{\text{relevant}}$. This means that fairness does not apply to them. Another consequence is that the ideally sorted queue will be bi-parted; the queue starts with all positive interactions (`like` and `apply`) in no particular order, and behind that a similar group of the non-positive scores (`unknown` and `ignore`); again in no particular order. For rank evaluation of queues, this means that queues will score high when job seekers interact more positively with matches in the first part of the queue than with matches in the tail. These naive evaluations will give a very rough understanding of the sorting algorithm performance; they do not provide insight into detailed interactions, but are therefore also not affected by intricate differences in relevance levels.

Interaction	Relevance Evaluation		
	Binary	Binary Non-negative	Binary Positive
`ignore`	-1	0	1
`unknown`	-1	0	1
`like`	0	1	2
`apply`	0	1	2

Table 5.2: Binary relevance evaluations

5.4.2. Fair Evaluations

The other relevance evaluations are ternary or quarternary. Permitted by the extra relevance levels, we introduce fairness in these evaluations: we can discern between the relevance of the `unknown`

interaction and the `ignore`. We begin with the four fair relevance evaluations.

Of the four fair relevance evaluations, three are ternary: one for each score range. Table 5.3 shows the fair evaluations. These evaluations are fair because for all $s_{\text{ignore}} < s_{\text{unknown}}$ applies; this means that these evaluations reward sorting algorithms that manage to put explicitly non-relevant opportunities in the tail of the queue. This goes against the nature of the typical queue, which has a tail of `unknown` interactions which the job seeker never reaches. The use of cutoff points remedies the effect of the `unknown` interaction tail, but nevertheless this effect will be visible in the NDCG scores for these evaluations: they will be lower because of it.

The singular quarternary fair evaluation is equal to the unrestricted fair evaluation, except that it also differentiates between `likes` and `applys`: $s_{\text{apply}} > s_{\text{like}}$. This evaluation will judge generated queues like the ternary fair evaluations, except it also expects applications to be put in the very beginning of the queue. Like explained in 5.3.3, this is not a realistic scenario, and we therefore expect few queues to score high in this regard. Despite the lower scores, we do consider the added differentiation between `likes` and `ignores` to provide more detailed insight in the sorting algorithm performance: it emphasises even more on ordering the more relevant jobs in the beginning of the queue. Considering this, and the evaluation's fairness that rewards `ignores` in the tail instead of `unknowns`, and it is trivial that the quarternary fair evaluation will yield low performance scores.

Interaction type	Relevance evaluation			
	Fair	Fair non-negative	Fair positive	Fair quarternary
`ignore`	-1	0	1	-1
`unknown`	0	1	2	0
`like`	1	2	3	1
`apply`	1	2	3	2

Table 5.3: Chosen ternary and quarternary fair relevance evaluations with different score ranges.

5.4.3. Unfair Evaluations

Lastly, we have chosen four unfair relevance evaluations, shown in Table 5.4. Opposite to the fair relevance evaluations, these unfair evaluations reward the typical queue's natural tendency to have `unknowns` in the tail. We therefore expect the scores for these evaluations to be higher than those of the fair evaluations.

The quarternary unfair relevance evaluation is equal to unrestricted unfair relevance evaluation, but again with the exception of differentiating between `likes` and `applys`. Expectations similar to that of the fair quarternary evaluation apply: the effect of differentiating between `likes` and `applys` will likely cause lower performance scores, but gives a more detailed insight of actual performance, because the most relevant jobs are expected at the start of the queue.

Interaction type	Relevance evaluation			
	Unfair	Unfair Non-negative	Unfair Positive	Unfair Quarternary
`ignore`	0	1	2	0
`unknown`	-1	0	1	-1
`like`	1	2	3	1
`apply`	1	2	3	2

Table 5.4: Chosen ternary and quarternary unfair relevance evaluations with different score ranges.

5.5. Chosen Cutoff points

As explained in Section 5.3.4, we should the cutoff point k based on how much opportunities the job seeker judges in their 'attention span' on the Matches Page. We will look at how, and how much job seekers interacted with queues, and derive cutoff points from that. For this analysis, we exclude generated queues for which there are no interactions: these queues were likely generated for a job

seeker visit which was not aimed at the Matches Page, but rather had another intent such as answering messages.

Figure 5.1 shows the number of direct interactions per queue, for 56,778 queues with less than 20 interactions (93.4% of all 64,498 queues) between [redacted] and [redacted] The median of the number of interactions is 5; furthermore we see that the graph has noteworthy plateaus at five, six and ten interactions; these are possible job seeker drop-off points.

Note that the data shown in Figure 5.1 concerns interactions made directly with a queue. When gathering interactions to calculate the NDCG score for a queue, we consider feedback given on the queue's opportunities that may be given by the job seeker while interacting with other queues.

Therefore, we choose two cutoff points $k = 6$ and $k = 10$. We motivate $k = 6$ by its closeness to the median of the number of direct interactions; if we cannot rely on interactions with other queues to provide more relevance feedback, we will not see the distortion of the unknown interactions in NDCG@6 scores. $k = 10$ does allow for inclusion of these interactions from other queues, but it has the downside that it could be affected by unknown interactions if that feedback cannot be retrieved.

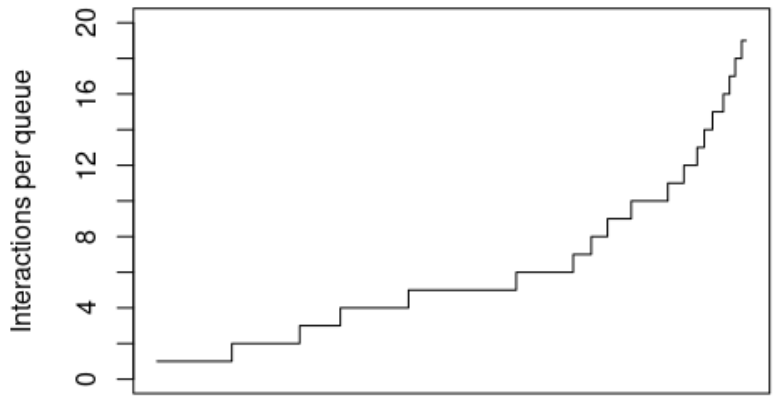

Figure 5.1: Interaction counts per queue, ordered from low to high, for all queues with less than 20 interactions (equivalent to 93.4% of all queues). The graph shows large numbers of queues for five and six interactions, and a noticeable irregularity at ten interactions.

In conclusion, we have introduced relevance evaluations as a set of numeric relevance scores that are awarded to interactions. When calculating NDCG scores for sorting algorithm evaluation, we will use relevance evaluations to translate interactions with the queue to a set of relevance feedback.

We introduced relevance evaluation properties score range, fairness and number of relevance levels, and discussed the expected influence of these properties on NDCG scores. We also motivated our choice of cutoff points for NDCG, based on queue interaction data.

We will use MRR as a baseline evaluation to evaluate the difference in performance between the two algorithms.

In the following chapter, we will perform an online experiment to compare the Opportunity Queue sorting algorithm from Section 4.4. In this experiment, we will compare the chosen relevance evaluations and cutoff points.

6

Online Experiment

To test the performance of the opportunity queue sorting algorithm proposed in Section 4.4, we design an experiment that will run in the production environment of Magnet.me. With this experiment, we answer our research questions: It will show how our sorting algorithm performs compared to the original algorithm, which provides an answer to RQ2. If our algorithm outperforms the original algorithm, we will consider our model of implicit preferences from Section 4.3 to provide a better understanding of job seeker preferences, which is what RQ1 asks. Finally, the experiment will employ different relevance evaluations to calculate NDCG with, which will help us answer RQ3.

The online experiment will run as an A/B-test: job seekers will be split up between a control group and a test group. The control group will be served the original, shuffled Curated Queue, and the test group will be served the Sorted Queue. In Section 6.1, we discuss the experimental setup. Finally the results from the experiment are presented and discussed in Section 6.2.

6.1. Experimental Setup

6.1.1. Participant Selection

The A/B-test experiment requires two separate groups; one control group that gets served the original queue, and one test group that gets served the alternatively sorted queue. Job seekers are divided between groups based on their independently awarded user id to avoid a selection bias in the experiment, per suggestion of related work in Section 2.3. A job seeker is part of the test group when $\texttt{userId} \bmod 10 < t$ for some threshold $0 \leq t < 10$: the selected percentage of job seekers for threshold t is then $t * 10\%$. This experiment ran with $t = 5$, which means that 50% of job seekers was eligible for the test group, and the remaining job seekers form the control group.

6.1.2. Fallback Sorting Algorithm

The sorting algorithm proposed in Section 4.4 depends on the availability of view stats for the job seeker that requests a queue. If there are no view stats available for the user, the queue generator falls back on the shuffle sort for the curated queue.

Unfortunately, a large group of users on Magnet.me creates their account, looks around on the platform once, and then drops off. The first time they open Magnet.me, they land on the Matches Page and see a Curated Queue; there are no view stats to generate a Sorted Queue from. They do not interact with many matches, and do not return to the Matches Page once they have left. As a result, these users will never be served a Sorted Queue, even if they are part of the test group based on their user id.

This causes the division of users between the test- and control group to skew towards the control group. The effect is hard to mitigate because the described behaviour cannot be foreseen, and correcting for it could introduce a bias in the experiment. As a result, the actual division of job seekers between the test group and the control group will not equal the intended 50/50 split.

6.1.3. Experimental Progression

The experiment was launched by setting the threshold t to 1, so 10% of job seekers were eligible for the test group. When the first queue generations with our sorting algorithm and the interactions with it showed no issues, the threshold was incremented to include 50% of users. On [redacted] data recording for the experiment began, and the experiment was closed [redacted] days later on [redacted]

Table 6.1 shows the information on the progression of the experiment: control- and test group size, absolute numbers of queues generated and queue interactions, and duration. In the table, we see the large discrepancy between the number of job seekers eligible for the test group, by id, and the actual number of job seekers that was served a Sorted Queue. This is caused by forced fallbacks to the shuffle algorithm due to a lack of page interaction data for new users; these users received a Curated Queue despite belonging in the test group.

The effect of this shift in groups is shown in the number of queues generated with respectively the shuffle sort and our proposed sort, and the accompanying number of interactions with each queue type. While a threat to the validity, the absolute numbers of sorted queues and interactions with those queues should suffice to draw conclusions on the performance of the sorting algorithm. We will further discuss the threat of validity after the interpretation of the results of this experiment.

Duration of experiment	[redacted]
Start of experiment	[redacted]
End of experiment	[redacted]
Number of job seekers in test group	9,437
Number of job seekers that were served a sorted queue	*3,956*
Number of job seekers in control group	8,812
Total number of job seekers	18,249
Number of curated queues generated	75,070
Number of sorted queues generated	36,804
Total number of queues generated (curated + sorted)	111,874
Number of curated queue interactions	266,101
Number of sorted queue interactions	112,228
Total number of queue interactions (curated + sorted)	378,329

Table 6.1: Information on the experiment size and duration

6.2. Results

We will analyse the scores from different perspectives: First, we will use the experiment to compare different relevance levels, fairness categories and the different cutoffs, and discuss their use in evaluating ranking algorithms for Magnet.me's Matches Queue. Then, we will look at the differences between the original Curated Queue, and the proposed Sorted Queue.

To accommodate the reader, Table 6.2 shows all chosen relevance evaluations from Section 5.4.

Evaluation		Interaction Type Ignore	Unknown	Like	Apply
Binary	Binary	-1	-1	0	0
	Non-negative	0	0	1	1
	Positive	1	1	2	2
Fair	Ternary	-1	0	1	1
	Quarternary	-1	0	1	2
	Non-negative	0	1	2	2
	Positive	1	2	3	3
Unfair	Ternary	0	-1	1	1
	Quarternary	0	-1	1	2
	Non-negative	1	0	2	2
	Positive	2	1	3	3

Table 6.2: All chosen relevance evaluation scores.

6.2.1. Comparing Different Relevance Evaluations

In Section 5.3, we proposed different properties of relevance evaluations: Score Range, Fairness, and Number of Relevance Levels. In this section, we will look at the results of the experiment from the perspective of each property and look what influence the value of the property seems to have on the average scores of the rank evaluation.

The NDCG scores for the online experiment are given in Table 6.3; grouped by evaluation type, and then ordered by score range. The table shows the results for both cutoffs: NDCG@6 and NDCG@10. Finally, the relative increase in performance between the Curated Queue and the Sorted Queue is shown.

Evaluation		NDCG@6 Curated	Sorted	Diff	NDCG@10 Curated	Sorted	Diff
Binary	Binary	0.4722	**0.4939**	+4.58%	0.4409	**0.4657**	+5.62%
	Non-negative	0.4722	**0.4939**	+4.58%	0.4409	**0.4657**	+5.62%
	Positive	0.4722	**0.4939**	+4.58%	0.4409	**0.4657**	+5.62%
Fair	Ternary	0.4073	**0.4452**	+9.30%	0.3786	**0.4200**	+10.96%
	Non-negative	0.4073	**0.4452**	+9.30%	0.3786	**0.4200**	+10.96%
	Positive	0.4073	**0.4452**	+9.30%	0.3786	**0.4200**	+10.96%
	Quarternary	0.4057	**0.4422**	+9.00%	0.3766	**0.4174**	+10.84%
Unfair	Ternary	**0.5785**	0.5747	-0.65%	**0.5831**	0.5721	-1.87%
	Non-negative	**0.5785**	0.5747	-0.65%	**0.5831**	0.5721	-1.87%
	Positive	**0.5785**	0.5747	-0.65%	**0.5831**	0.5721	-1.87%
	Quarternary	**0.5746**	0.5698	-0.84%	**0.5777**	0.5671	-1.82%

Table 6.3: NDCG scores for all chosen relevance evaluations, per queue type, and the increased performance of the Sorted Queue compared to the Curated Queue, for cutoffs 6 and 10. Per relevance evaluation and cutoff, the highest score is printed in bold font.

Score Ranges

In Section 5.3.1, we described different score ranges: unrestricted, non-negative and positive. We tested these different ranges to see if, by modifying the way NDCG is calculated as described in Section 5.3.1, we can use relevance evaluations with a non-zero lower bound to calculate the NDCG score.

Table 6.3 clearly shows that different score ranges do not change the NDCG score, given an unchanged fairness and number of relevance levels: All binary relevance evaluations have identical scores for either cutoff; the same applies to the fair ternary- and unfair ternary levels. Since we only have quarternary relevance evaluations, we cannot show that they have identical scores as well, but we deem it assumable that score ranges do not cause differences in NDCG score here.

In conclusion, we can use arbitrary score ranges for relevance evaluations used in calculating NDCG scores. Punishing non-relevant items with negative feedback, using zero values and using only positive values does not change the output of the relevance evaluation metric, given that the relative relevance between interactions does not change.

Because Score Ranges do not influence the NDCG score, we condense Table 6.3 by leaving out the restricted score ranges; Table 6.4 shows only the unrestricted relevance evaluations for easy readability.

Evaluation		NDCG@6 Curated	NDCG@6 Sorted	NDCG@6 Diff	NDCG@10 Curated	NDCG@10 Sorted	NDCG@10 Diff
	Binary	0.4722	**0.4939**	+4.58%	0.4409	**0.4657**	+5.62%
Fair	Ternary	0.4073	**0.4452**	+9.30%	0.3786	**0.4200**	+10.96%
	Quarternary	0.4057	**0.4422**	+9.00%	0.3766	**0.4174**	+10.84%
Unfair	Ternary	**0.5785**	0.5747	-0.65%	**0.5831**	0.5721	-1.87%
	Quarternary	**0.5746**	0.5698	-0.84%	**0.5777**	0.5671	-1.82%

Table 6.4: NDCG scores for all *unrestricted* relevance evaluations, per queue type, and the increased performance of the Sorted Queue compared to the Curated Queue, for cutoffs 6 and 10. Per relevance evaluation and cutoff, the highest score is printed in bold font.

Fairness

In Section 5.3.2 we describe the relevance evaluation property fairness. Fairness indicates whether the relevance score for an `unknown` interaction is higher or lower than the relevance score for the explicit `ignore`. A score is fair if `ignore` is considered less relevant than `unknown`, and vice versa. When the two scores are equal, the evaluation is neither considered fair nor unfair.

Table 6.4 shows the results of the experiment, grouped by fairness. If we compare the fair and unfair relevance evaluations, we see that the unfair evaluations produce far higher scores than the fair evaluations. This is expected: as described in Section 5.3.2 the unfair relevance evaluations suggest an ideal sorting that has the `unknown` interaction tail, which the typical queue also has. As a result, most queues will be more similar to the ideal sorting according to the unfair relevance evaluations, which yields higher NDCG scores.

Interestingly, we see that all fair evaluations, regardless of the score range and cutoff, suggest that the Sorted Queue outperforms the Curated Queue, and the improvements in performance are relatively similar over all score ranges. The relative increase in performance averaging around 10% shows that the NDCG clearly distinguishes between the two queue types, when using fair evaluations. The unfair evaluations, however, portray a very different result: NDCG@10 reports a performance decrease of less than 2% and NDCG@6 scores even report a decrease of less than 1%. The small decrease of performance does not strongly distinguish the two sorting algorithms.

We conclude that if we find that the `ignore` interaction less relevant than the `unknown` interaction, we should use a fair relevance evaluation; based on NDCG scores, we can then conclude which sorting algorithm works better. In other words; NDCG is capable of distinguishing between the sorting algorithms in terms of performance, when using fair relevance evaluations. Applied in our experiment, we see that the sorting algorithm we proposed outperforms the shuffle algorithm. Alternatively, if we find the `unknown` relation less relevant than the `ignore` relation, then using an unfair relevance evaluation for NDCG does not clearly decide on a winning algorithm.

Number of Relevance Levels

Again using Table 6.4, we will discuss the differences between the different relevance evaluations, this time from the perspective of number of relevance levels. In Section 5.3.3, we describe binary, ternary and quarternary relevance evaluations. They have two, three and four distinct relevance values respectively, which permit more or less differentiation between relevance levels.

Starting with the binary relevance evaluation, we see that the binary evaluation clearly chooses the new sorting algorithm as the winner. Binary evaluations are crude, and somewhat naive relevance evaluations: they only distinguish between 'relevant' and 'non-relevant'. From the results, we conclude that if we do not differentiate between different relevant interactions, or between different non-relevant interactions, then the Sorted Queue outperforms the Curated queue.

Ternary relevance evaluations are more complex than the binary evaluations: they introduce an extra level of relevance. In the chosen relevance evaluations, we use the extra relevance levels to differentiate between `unknown` and `ignore` interactions. This extra layer of complexity causes more room for errors in the ordering, which we see in the fair ternary evaluation: scores for these evaluations are lower than the binary evaluation scores. When we include the quarternary evaluations, with yet another added layer of relevance, we see a continuation of the expected decreasing scores: Fair quarternary evaluations yield lower scores than the fair ternary evaluations, and the same applies to the unfair evaluations respectively.

However, the unfair ternary evaluation yields far higher scores than the binary evaluation. This sudden increase in scores can be attributed to the introduction of fairness, which the binary evaluation does not have: Since binary evaluations are not fair nor unfair, they do not specifically reward the tail of `unknown` interactions in queues, while the unfair evaluation does. This explains the increase in scores when going from the binary evaluation to the unfair ternary evaluation.

Section 5.3.3 added complexity of the fourth level of relevance could hurt the distinguishing property of NDCG. While both fair and unfair evaluations show decreased scores when comparing ternary to quarternary, the scores decrease proportionally. We concluded that NDCG is not capable of distinguishing between the algorithms when using unfair scores, so we cannot conclude if the unfair quarternary evaluation hurt that even further, compared to the ternary equivalent. Contrarily, we can see that NDCG with the fair quarternary evaluation is capable of distinguishing between the two sorting algorithms as well as with the fair ternary evaluation.

In conclusion, added relevance levels do show a slight decrease in yielded NDCG scores, but do not hurt the distinguishing property of NDCG. This means that we do not have to sacrifice specificity of the relevance evaluation to use NDCG to distinguish between sorting algorithms.

6.2.2. Comparing the Curated and the Sorted Queue Sorting Algorithms

After studying the experiment results from all relevance evaluation property perspectives, we now focus on whether our sorting algorithm outperforms the original shuffle algorithm.

First, we look at our naive baseline metric, the Mean Reciprocal Rank: The scores for the respective sorting algorithms are shown in Table 6.5. It shows that our newly proposed sorting algorithm outperforms the shuffle algorithm: in queues generated with our sorting algorithms, job seekers find the first relevant opportunity earlier in the queue.

Evaluation	Curated	Sorted	Difference
MRR	0.4920	**0.5093**	+3.52%

Table 6.5: Mean Reciprocal Rank score for the Curated Queue and the Sorted Queue, and the relative difference.

The classical binary relevance evaluations for NDCG, with scores 0 and 1 for non-relevant and relevant matches respectively, show that the sorting algorithm based on implicit preferences outperforms the original algorithm.

When considering the relevance evaluations with fairness, the NDCG metric using unfair relevance evaluations cannot distinguish between the two algorithms. However, Table 6.3 shows that NDCG using fair evaluations decides on our Sorted Queue sorting algorithm, for all score ranges and cutoff points, and therein agrees with MRR and the binary NDCG evaluation. It considers the relative ordering of missing feedback and negative feedback, which binary NDCG and MRR do not do, and under that consideration fair NDCG reports an even stronger increase in performance than these evaluations.

From these evaluations, we conclude that our sorting algorithm based on implicit job seeker preferences derived from page interactions outperforms the original ordering algorithm. We will draw further conclusions in the next chapter.

7

Discussion and Future Work

7.1. Conclusion

We implemented a new way of monitoring user behaviour on Magnet.me, focused on page interactions. Furthermore, we have shown patterns in opportunity page interaction data that correlates longer duration of interactions with positive affection towards opportunities. From this offline experiment we derive that the data we track is suitable to derive implicit preferences from, answering RQ1a.

Based on the correlation between page interaction duration and affection towards opportunities, we implemented a system that aggregates implicit job seeker preferences regarding opportunities from the page interactions for RQ1b. In turn, these implicit preferences are used in a newly proposed sorting algorithm for Magnet.me's Matches Queue.

With the success of this model of implicit preferences, and the sorting algorithm proposed based on them in our online experiment, we have answered RQ1 and RQ2: We have gained a better understanding of job seeker preferences, beyond their explicitly provided preferences, and we have used the model of implicit preferences to propose a sorting algorithm for the Opportunity Queue.

To evaluate performance of this sorting algorithm, we studied several aspects of ranking evaluation using Normalised Discounted Cumulative Gain: Score Ranges, Fairness, and Number of Relevance Levels. Using these properties, we proposed several relevance evaluations to apply when calculating NDCG scores to evaluate queue performance. We determined that Score Ranges, as expected, do not influence outcomes of evaluations with our modified calculation of NDCG. Looking at Fairness, we see that using fair evaluations with NDCG shows that the proposed sorting algorithm outperforms the original algorithm. However, for unfair relevance evaluations, NDCG does not strongly differentiate performance of the original algorithm and the newly proposed algorithm. Finally, the Number of Relevance Levels does not hurt the distinguishing capability of NDCG. With the online experiment, using the different relevance evaluations, we can answer RQ3a, which asks how we can evaluate recommendation relevance based on job seeker feedback.

To answer RQ3b, which is concerned with how we should handle missing feedback, we compare fair and unfair relevance evaluations. The fair and unfair evaluations differ in how they handle missing feedback through the relative position of the `unknown` interaction to other interactions; therefore, comparing the experimental results of the fair and unfair evaluations suggest how we should handle missing feedback. If we think negative feedback indicates lower relevance than missing feedback, we can determine the better algorithm using NDCG with fair relevance evaluation. If we think negative feedback indicates higher relevance, then NDCG cannot point out a winning algorithm.

We finally conclude that MRR, binary NDCG and fair NDCG evaluation show that the proposed sorting algorithm based on implicit preferences outperforms Magnet.me's original sorting algorithm. Combining our analysis of relevance evaluations, handling missing feedback, and the conclusion that our sorting algorithm outperforms the original algorithm, we answered RQ3: we have shown how we can evaluate rank performance, and we can distinguish between algorithm performance: We have used NDCG with binary and fair relevance evaluations in an online experiment to show that our proposed sorting algorithm, outperforms Magnet.me's current algorithm, which agrees with the baseline

algorithm. Thus, we have shown how we can improve Magnet.me's Job Recommendations algorithm using User Interaction Data, which was the main question of this thesis.

7.2. Threats to Validity

We have shown that our Opportunity Queue sorting algorithm outperforms Magnet.me's original sorting algorithm. However, there are several aspects to our research that could affect the validity of our experiment. In this chapter, we discuss these effects and how much they (may) have affected the experiment, and we suggest ways to mitigate them. We also propose continuations of the work done in this thesis, some of which are already planned for implementation on Magnet.me's roadmap.

7.2.1. Queue Changes

As described in Section 3.4.4 there are several events that can be triggered during a job seeker's interactions with the Matches Queue that change the content of the queue. Specifically, the perceived rank of matches changes when intermediate matches are removed, which affects the validity of the rank evaluation in two way:

Firstly, matches that are removed are still evaluated in the rank evaluation as `unknown` interactions. The removal of the matches should be handled in the rank evaluation by removing them from the evaluated queue too; however, that causes the second effect on the rank evaluation validity:

The matches after a removed match shift in rank in the UX: they are moved one place to the front of the queue. As described in Section 3.4.1, job seekers interact with their matches as a Queue. When a match is removed from the queue, all matches behind it move one rank forward. When evaluating queues with removed matches, these subsequent matches should be discounted less than they are now: because the rank of the subsequent matches is not updated, the rank is too high which makes the discount too strong. In the current situation, the effect of not applying these rank changes impacts the rank evaluation score by lowering it.

7.2.2. Application Interactions

When a job seeker clicks 'apply' on the opportunity modal on the Matches Page, they are sent to Magnet.me's messaging system to send an application message to the employer, or they are redirected to the employer's external application website. There are two considerations to be made because of this UX:

First of all, the redirect to the messaging system does not guarantee that the job seeker actually applies for the job; the completion of the application is not monitored for rank evaluation. While we can assume that the `apply` click strongly suggests the job seeker's intent to apply, the assumption remains that; an assumption. The completion or abortion of the application process could be monitored and fed back to the rank evaluation system; if the process was aborted, this should be handled accordingly when evaluating the queue.

Secondly, the redirect to the messaging system for internal applications makes the job seeker leave the Matches Page. This shortcuts the job seekers interactions with the queue on the page, and affects the total number of interactions they make with that queue. For the queue they were interacting with, this likely means that the typical tail of `unknown` interactions starts earlier than when the job seeker would not have applied. This could mean that the queue receives a lower absolute score because there are more `unknown` interactions which may have been positive interactions; it's even more wry to consider that this lower score would be caused by a job seeker's feedback that the match in the queue was highly relevant. We can conclude from this hypothetical situation that a higher number of interactions per queue might not be an ideal metric to measure queue performance with.

Handling the redirect to the messaging system, and aborted application processes is not trivial; we do not have a ready-to-implement solution. However, the implications on queue evaluation should be taken in consideration.

7.2.3. Talent Matcher Changes

During the process of the online experiment, Magnet.me entered a new market with a different target group: Young professionals. These Young Professionals differ from the typical job seekers because they are more interested in jobs that require some work experience, rather than just a certain education level or field of study.

To facilitate this new target group, the Talent Matcher was updated to match job seekers and employers on the extra opportunity property for required work experience. For our experiment, this means that the opportunities that needed to be sorted by our algorithm changed. The impact of the changes reaches no further than stricter filtering, which may mean that the opportunity queue was smaller on average. It also meant that the categorical opportunity property 'work experience' monitored in the job seeker view stats aggregation got more important.

While the experiment was running, no changes were observed in the intermediate rank evaluation scores, nor in the number of job seekers with- or without view stats. Furthermore, since these changes apply to both the Curated Queue and the Sorted Queue, we consider the effect negligible.

7.2.4. Skewed Selection of Participants

In Section 6.1 we indicate that many job seekers that were selected for the online experiment test group, based on their user id, were actually part of the control group. This was caused by a lack of page interaction data for these job seekers; they simply had not viewed any content yet. Often, these are job seekers that create an account on Magnet.me, look around once (and therefore generate a single curated queue), and then leave, never to be seen again. Such job seekers are prevalent on Magnet.me.

We already discuss that the absolute numbers are still sufficient to make the experiment reliable; after all, the goal of having 50% of job seekers in our test group was to provide us with sufficient job seekers in both the test and control group. However, it suggests that when running experiments like these, the influence of the dependency on page interaction data should be considered with more care. In this case, the percentage of job seekers eligible for the experiment could have been raised to above 50% to correct for the users without page interaction data; by tweaking the threshold one could make sure the groups are of equal size.

Alternatively, one could suggest that for the control group, only job seekers are considered that interact with page interaction data; that way, whether job seekers have interacted with content is invariant between both groups.

7.3. Future Work

While our online experiment has shown promising results, there are plenty of ways the work in this thesis could be improved with. In this section we will first go through a number of items that are directly related to the experiments conducted for this thesis, and the sorting algorithm proposed in it, after which we will propose a number of new aspects that could be employed to improve the performance of the Matches Queue.

In our experiment, we decided not to use the `eventCount` property of page interactions to derive implicit preferences from. However, the offline experiment in Section 4.2.3 suggests that `eventCount` is a significant predictor of positive affection towards opportunities, like the duration is. Including the `eventCount` might further improve the aggregated implicit preferences for job seekers, which could boost performance of the proposed sorting algorithm.

Furthermore, our data selection for the aggregation of job seeker preferences is very conservative, only taking page interactions in consideration where the job seeker already stated their interest in the opportunity on the page. With this conservative selection, we discard more than half of the available opportunity page interactions data. The offline experiment suggests that patterns in these page interactions are suitable for the aggregation of implicit preferences as well. Future work could study the performance of the sorting algorithm when implicit preferences are derived from the page interactions excluded for this experiment; this may very well resolve the lack of job seeker view stats that caused the experiment to fall back on the shuffle algorithm described in Section 6.1.3.

Magnet.me currently tracks user behaviour on an event basis, rather than on a user-centered basis. By that, we mean that it is currently difficult to follow the job seeker through the application and correlate behaviour between different parts of the web platform. To get a proper understanding of how the platform performs, a proper, complete understanding of the job seeker's behaviour is crucial. For example, it would be very interesting to show if the average number of applications for job seekers that were served the Sorted Queue differs from the other job seekers; it would suggest that in a very

early part of their experience on Magnet.me, the Matches Page, the foundations are laid for a more successful path for the job seeker. Magnet.me would greatly profit from such metrics.

In the past, Magnet.me has struggled with feedback on the Matches Queue that remarked the low diversity in jobs. Especially when matched with big corporate employers, job seekers would receive their opportunities back to back in the Matches Page. To mitigate this, the shuffle algorithm was devised.

The underlying need that job seekers expressed in their feedback was diversity, which is a recognised property of recommendations that recommenders aim to suffice in. Instead of the shuffle algorithm, Magnet.me could develop a recommender system that focuses strongly on diversity and possibly serendipity to prevent user fatigue because of monotone content.

One of the constraints for this experiment was that the User Interface and User Experience were not to be changed, except for the order of the matches in the Matches Queue. There are, however, several interesting developments in the field of recommender systems that could contribute to Magnet.me's Matches Page. One of these developments is recommendation explanation: a recommendation is accompanied by an explanation about the selection of this recommendation for the job seeker. Magnet.me has struggled with job seeker's understanding of receiving recommendations in the past, to which they responded by making the explicit preference inputs directly accessible from the Matches Page. To facilitate an even better understanding of job seekers of the recommendations they receive, a short explanation could be very effective.

A

The Talent Matcher

Magnet.me matches job seekers and potential employers based respectively on their qualifications and interests, and requirements and properties. The logic responsible for this matching process is referred to as the Talent Matcher.

The Talent Matcher is a service that recalculates matches on an event basis. These events come from users posting or updating content on Magnet.me: a job seeker that updates their résumé, for example, or a recruiter that posts a new opportunity. A calculation consists of comparing a student profile to all companies and opportunities, or comparing an opportunity to all student profiles.

The Talent Matcher concludes there is a match between a job seeker and an opportunity when the job seeker's résumé meets the requirements of the opportunity. These requirements can consist of:

- an education of a minimal level, and/or in a certain field
- work experience of a minimal duration, and/or in a certain field
- minimal proficiency in one or more languages
- extracurricular activities

Visa versa, the opportunity and the providing company must meet the job seeker's interests:

- job type, e.g. (graduate) internship, job
- job function, e.g. administrative, research, engineering
- employment type, full-time or part-time
- company size, expressed as ranges of number of employees
- location of employment

The service outputs the matches as potential connections between a job seeker and an opportunity, or a company. It is up to the job seeker to actually create the connection. Depending on job seeker profile settings, only then can the recruiter at the other end of the connection get insights in their potential employee.

A.1. Match Score

Given that there is a match between a job seeker and an opportunity, the Talent Matcher also outputs a so called 'match score'. This score was originally intended to indicate the strength of the match: a higher score would indicate a stronger match.

However, the calculation of this match score is severely unbalanced, which caused issues in the user experience on the Matches Page: Consider a job seeker that completed several higher educations, and an opportunity that requires a general higher education without specifying a study field for this higher education. In this case, the Talent Matcher would award a disproportionally high match score to the

match. This problem is exacerbated by a tendency of recruiters to apply such generic requirements to all opportunities of the company. This results in a set of opportunity matches for the job seeker that all have a disproportionally high match score, outweighing the matches of other opportunities with more specific requirements. Furthermore, matches with more specific requirements tend to be better matches for job seekers, as opposed to so called 'catch-all' requirements that are as generic as the one proposed. This mismatch creates an unbalanced user experience for the job seeker.

B

Deterministic Shuffling of the Curated Queue

When a job seeker is served a Curated queue on the Matches Page, it is shuffled deterministically. This shuffling is applied to alleviate a problem in the user experience of the page: long streaks of opportunities from the same companies.

The cause of this problem is bi-partial, and to describe it we must first explain some constraints on the Matches Queue. The queue is retrieved from the server in batches; the 'next' batch is retrieved when there are three matches left in the browser for the job seeker to consider. Between the retrieval of one batch and the next one, the queue may have been recalculated, meaning that the second batch of matches comes from a different queue than the old one. To make sure that the matches in the new batch do not overlap with the left-over three matches, the ordering of the queue must be stable over successive calculations of the queue.

Originally, this was alleviated by using the 'default' legacy ordering, which meant sorting the matches on their Talent Matcher 'match score'. The match score, however, is not suitable as a property to sort on; this is explained in Appendix A.1. It caused the job seeker to see long streaks of opportunities from the same companies, and caused fatigue which caused the user to navigate away from the Matches Page.

To remedy this problem, Magnet.me introduced a 'deterministic shuffle'. The aim of this solution was to introduce randomness in the order of matches, but in a way that is stable over successive calculations of the curated queue. This was achieved by exploiting a pattern in opportunities posted by a single companies: because they would be posted immediately after one another, they would have consecutive ids. One can 'shuffle' numerical values seemingly random by taking a (large) prime number p and use $p \bmod id$ to sort the matches on. Since the $\bmod$ operator is deterministic, this shuffle is stable. For a large enough prime, it can also be assumed that consecutive ids are separated in the shuffled list.

This solution solved the user experience issue, and has remained in effect ever since. Users experience the curated queue as sufficiently mixed, while not being presented with double matches or, in contrary, missing out on opportunities because they 'slipped through the cracks' of the batched retrieval of the queue.

C

Queue Changes

The Curated Queue for a job seeker is subject to change based on events both caused by the interaction of the user with the queue, as from elsewhere. We describe the events that trigger such changes, and describe what the changes are. We also describe how the changes affect the rankings and the rank evaluation process.

- `ignore` **a company.** In this case, the assumption is made that the job seeker is also not interested in the opportunities for this company. These companies are then removed from the queue, so the job seeker does not have to process them. For our experiment, we consider the interaction to be `unknown`; despite the underlying assumption that the job seeker dislikes the opportunities, we do not want to consider the assumption we make to be an explicit action.

- **Match calculation completion.** When a job seeker alters their résumé or preferences, the Talent Matcher (see Section 3.3) recalculates the matches between the job seeker and all companies. If new matches are generated, any existing curated queue for the job seeker is discarded, and a new queue is generated in its place.

- **Opportunity/company deletion from Magnet.me.** When a company, or one of a company's opportunities, is removed from Magnet.me, expires past a preset date, or is unpublished by the employer, all relevant opportunities present in a job seeker's curated queue are removed. The job seeker therefore will not see the relevant matches in their queue, which means that the interaction the user would have made with the matches remains unknown. When calculating the rank evaluation for such a queue, the score for the `unknown` interaction is applied to all affected items.

- **Interaction with a match outside the Matches Queue.** A job seeker is, of course, not guaranteed to be on the Matches Page. Given the situation that a curated queue is generated for a job seeker, it must be kept up to date with any changes in relation between the job seeker and any company or opportunity. When a user navigates away from the queue and interacts with opportunities or companies by connecting to companies or `liking`/`applying` to opportunities, the following cases apply: if a company is `ignored`, the case of ignoring a company as describe above, applies. If the job seeker `likes`, or `applies` to an opportunity, then that opportunity is removed from the queue. The same applies to a company that the job seeker connects with.

  All these interactions are performed outside of the Matches Queue, and because of that, are not tracked in the queue interactions where they can be used as relevance feedback interactions.

The events described above have varying effects on the queue, all of which complicate the analysis of job seeker interactions with the Matches Queue. We can discern between a complete recalculation of the queue and mutations of a queue.

C.1. Queue Recalculation

In the case of a recalculation, an old queue is discarded and a new queue is immediately generated to replace the old one. The events that lead to these recalculations can be rigorous; when a user significantly changes their preferences, they might include or exclude a lot of companies or opportunities in their filter scope. Similarly, when they add an education, they could qualify for more opportunities than before.

Bibliography

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

[2] Z. Siting, H. Wenxing, Z. Ning, and Y. Fan, *Job recommender systems: a survey,* in *2012 7th International Conference on Computer Science & Education (ICCSE)* (IEEE, 2012) pp. 920–924.

[3] P. B. Thorat, R. Goudar, and S. Barve, *Survey on collaborative filtering, content-based filtering and hybrid recommendation system,* International Journal of Computer Applications **110**, 31 (2015).

[4] D. Kluver, M. D. Ekstrand, and J. A. Konstan, *Rating-based collaborative filtering: algorithms and evaluation,* in *Social Information Access* (Springer, 2018) pp. 344–390.

[5] N. Pereira and S. Varma, *Survey on content based recommendation system,* Int. J. Comput. Sci. Inf. Technol **7**, 281 (2016).

[6] R. Rafter and B. Smyth, *Passive profiling from server logs in an online recruitment environment,* in *Proceedings of the IJCAI Workshop on Intelligent Techniques for Web Personalization (ITWP 2001)* (2001) pp. 35–41.

[7] T. Keim, *Extending the applicability of recommender systems: A multilayer framework for matching human resources,* in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)* (IEEE, 2007) pp. 169–169.

[8] J. Malinowski, T. Weitzel, and T. Keim, *Decision support for team staffing: An automated relational recommendation approach,* Decision Support Systems **45**, 429 (2008).

[9] J. Malinowski, T. Keim, O. Wendt, and T. Weitzel, *Matching people and jobs: A bilateral recommendation approach,* in *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, Vol. 6 (IEEE, 2006) pp. 137c–137c.

[10] *Xing.com,* https://xing.com, accessed: 3 May 2019.

[11] W. Xiao, X. Xu, K. Liang, J. Mao, and J. Wang, *Job recommendation with Hawkes process,* in *Proceedings of the Recommender Systems Challenge on - RecSys Challenge '16* (ACM Press, New York, New York, USA, 2016) pp. 1–4.

[12] F. Ricci, L. Rokach, and B. Shapira, *Introduction to recommender systems handbook,* in *Recommender systems handbook* (Springer, 2011) pp. 1–35.

[13] G. Jawaheer, M. Szomszor, and P. Kostkova, *Comparison of implicit and explicit feedback from an online music recommendation service,* in *proceedings of the 1st international workshop on information heterogeneity and fusion in recommender systems* (ACM, 2010) pp. 47–51.

[14] D. W. Oard, J. Kim, *et al.*, *Implicit feedback for recommender systems,* in *Proceedings of the AAAI workshop on recommender systems*, Vol. 83 (WoUongong, 1998).

[15] M. Reusens, W. Lemahieu, B. Baesens, and L. Sels, *A note on explicit versus implicit information for job recommendation,* Decision Support Systems **98**, 26 (2017).

[16] D. Kelly and J. Teevan, *Implicit feedback for inferring user preference: a bibliography,* in *SIGIR forum*, Vol. 37 (2003) pp. 18–28.

[17] M. Nichols David, *Implicit ratings and filtering,* in *Proceedings of the 5th DELOS Workshop on Filtering and Collaborative Filtering. Budapaest: ERCIM*, Vol. 12 (1997).

[18] X. Amatriain and J. Basilico, *Recommender systems in industry: A netflix case study,* in *Recommender systems handbook* (Springer, 2015) pp. 385–419.

[19] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan, *Web usage mining: Discovery and applications of usage patterns from web data,* Acm Sigkdd Explorations Newsletter **1**, 12 (2000).

[20] D. Pierrakos, G. Paliouras, C. Papatheodorou, and C. D. Spyropoulos, *Web usage mining as a tool for personalization: A survey,* User modeling and user-adapted interaction **13**, 311 (2003).

[21] V. Anitha and P. Isakki, *A survey on predicting user behavior based on web server log files in a web usage mining,* in *2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16)* (IEEE, 2016) pp. 1–4.

[22] S. K. Gudla, J. Bose, V. Gajam, and S. Srinivasa, *Relevancy Ranking of User Recommendations of Services Based on Browsing Patterns,* in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)* (IEEE, 2017) pp. 765–768.

[23] M. Morita and Y. Shinoda, *Information filtering based on user behavior analysis and best match text retrieval,* in *SIGIR'94* (Springer, 1994) pp. 272–281.

[24] W. Dimpfel and A. Morys, *Quantitative objective assessment of websites by neurocode-tracking in combination with eye-tracking,* Journal of Behavioral and Brain Science **4**, 384 (2014).

[25] D. Bansal, J. Bose, and A. Kumar, *Eeg based detection of area of interest in a web page,* in *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (IEEE, 2015) pp. 1320–1325.

[26] G. Slanzi, C. Aracena, and J. D. Velásquez, *Eye tracking and eeg features for salient web object identification,* in *International Conference on Brain Informatics and Health* (Springer, 2015) pp. 3–12.

[27] J. Bose, A. Singhai, A. A. Patankar, and A. Kumar, *Attention Sensitive Web Browsing,* in *Proceedings of the Ninth Annual ACM India Conference on - ACM COMPUTE '16* (2016) arXiv:1601.01092 .

[28] L. Peska and P. Vojtas, *Using implicit preference relations to improve recommender systems,* Journal on Data Semantics **6**, 15 (2017).

[29] C. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval,* Natural Language Engineering **16**, 100 (2010).

[30] D. R. Swanson *et al.*, *Historical note: Information retrieval and the future of an illusion,* Journal of the American Society for Information Science **39**, 92 (1988).

[31] C. J. Van Rijsbergen, *Information Retrieval*, 2nd ed. (Butterworth, 1979).

[32] A. Gunawardana and G. Shani, *Evaluating recommender systems,* in *Recommender systems handbook* (Springer, 2015) pp. 265–308.

[33] E. M. Voorhees *et al.*, *The trec-8 question answering track report,* in *Trec*, Vol. 99 (Citeseer, 1999) pp. 77–82.

[34] K. Järvelin and J. Kekäläinen, *Ir evaluation methods for retrieving highly relevant documents,* in *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval* (ACM, 2000) pp. 41–48.

[35] A. Al-Maskari, M. Sanderson, and P. Clough, *The relationship between ir effectiveness measures and user satisfaction,* in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval* (ACM, 2007) pp. 773–774.

[36] E. Kanoulas and J. A. Aslam, *Empirical justification of the gain and discount function for ndcg,* in *Proceedings of the 18th ACM conference on Information and knowledge management* (ACM, 2009) pp. 611–620.

[37] E. M. Voorhees, *Evaluation by highly relevant documents,* in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval* (ACM, 2001) pp. 74–82.

[38] Y. Wang, L. Wang, Y. Li, W. Chen, Y. Wang, L. Wang, Y. Li, D. He, W. Chen, and T.-y. Liu Wang Wang Li He Chen Liu, *A Theoretical Analysis of NDCG Ranking Measures*, Tech. Rep. (2013).

[39] J. S. Breese, D. Heckerman, and C. Kadie, *Empirical analysis of predictive algorithms for collaborative filtering,* in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence* (Morgan Kaufmann Publishers Inc., 1998) pp. 43–52.

[40] M. D. Ekstrand, J. T. Riedl, J. A. Konstan, *et al.*, *Collaborative filtering recommender systems,* Foundations and Trends® in Human–Computer Interaction **4**, 81 (2011).

[41] W. S. Cooper, *Expected search length: A single measure of retrieval effectiveness based on the weak ordering action of retrieval systems,* American documentation **19**, 30 (1968).

[42] V. Raghavan, P. Bollmann, and G. S. Jung, *A critical investigation of recall and precision as measures of retrieval system performance,* ACM Transactions on Information Systems (TOIS) **7**, 205 (1989).

[43] F. McSherry and M. Najork, *Computing information retrieval performance measures efficiently in the presence of tied scores,* in *European conference on information retrieval* (Springer, 2008) pp. 414–421.

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

[45] S. Fazeli, H. Drachsler, M. Bitter-Rijpkema, F. Brouns, W. Van der Vegt, and P. Sloep, *User-centric evaluation of recommender systems in social learning platforms: Accuracy is just the tip of the iceberg,* (2017).

[46] M. Kaminskas and D. Bridge, *Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems,* ACM Trans. Interact. Intell. Syst. **7**, 2:1 (2016).

[47] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, *Controlled experiments on the web: survey and practical guide,* Data mining and knowledge discovery **18**, 140 (2009).

[48] A. Gilotte, C. Calauzènes, T. Nedelec, A. Abraham, and S. Dollé, *Offline a/b testing for recommender systems,* in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining* (ACM, 2018) pp. 198–206.

[49] *Magnet.me press page,* https://magnet.me/press, accessed: 14 May 2019.

[50] *Page visibility api - web apis | mdn,* https://developer.mozilla.org/en-US/docs/Web/API/Page_Visibility_API, accessed: 6 May 2019.