Implementing a Resume Database with Online Learning to Rank

Emil Ahlqvist

August 26, 2015
Master’s Thesis in Computing Science, 30 credits
Supervisor at CS-UmU: Jan-Erik Moström
Supervisor at Knowit Norrland: Andreas Hed
Examiner: Fredrik Georgsson

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN
Abstract

Learning to Rank is a research area within Machine Learning. It is mainly used in Information Retrieval and has been applied to, among other systems, web search engines and in computational advertising. The purpose of the Learning to Rank model is to rank a list of items, placing the most relevant at the top of the list, according to the users’ requirements. Online Learning to Rank is a type of this model, that learns directly from the users’ interactions with the system.

In this thesis a resume database is implemented, where the search engine applies an Online Learning to Rank algorithm, to rank consultant’s resumes, when queries with required skills and competences are issued to the system. The implementation of the Resume Database and the ranking algorithm, as well as an evaluation, is presented in this report. Results from the evaluation indicates that the performance of the search engine, with the Online Learning to Rank algorithm, could be desirable in a production environment.
Acknowledgements

First, I would like to thank Knowit Norrland AB, and all their employees that I have met, for giving me the opportunity to do this project at their office in Umeå. A special thanks goes to Jan-Erik Moström for taking the time to read my report and for giving me feedback, and Andreas Hed that has given me advices and support during the project. Also, a huge thanks to Anne Schuth\(^1\) who has helped me by giving valuable tips and feedback in the Learning to Rank area.

Last, but not least, I would also like to thank my friends, family and especially my girlfriend for support and encouragement during this work.

\(^1\)http://www.anneschuth.nl/
Contents

1 Introduction ................................................. 1
   1.1 Background ............................................. 1
   1.2 Problem statement ..................................... 1
   1.3 Purpose and goals ..................................... 2
   1.4 Related work .......................................... 3
   1.5 Outline ................................................. 4

2 Method .................................................. 5
   2.1 Introduction ........................................... 5
   2.2 Learning to Rank ........................................ 6
      2.2.1 Feature vectors .................................... 7
      2.2.2 Discriminative Training ............................. 8
      2.2.3 Ranking algorithm approaches ..................... 8
   2.3 Online Learning to Rank ............................... 9
      2.3.1 User feedback .................................... 9
      2.3.2 Exploitation versus exploration .................. 11
      2.3.3 Interleaved comparison methods ................ 11
      2.3.4 Dueling Bandit Gradient Descent ............... 13
   2.4 Evaluation measures .................................. 13
   2.5 Ranking algorithm for the Resume Database ........ 15
      2.5.1 Feature engineering ............................... 15
      2.5.2 Evaluation methodology ........................... 16

3 Work process ............................................ 19
   3.1 Preliminaries ......................................... 19
   3.2 How the work was carried out ......................... 19

4 Results .................................................. 21
   4.1 System overview ...................................... 21
      4.1.1 System architecture ................................ 21
      4.1.2 System functionalities ............................ 22
# CONTENTS

4.2 Back-end .................................................. 24
  4.2.1 Database ............................................. 24
  4.2.2 API middleware ..................................... 25
  4.2.3 Search engine ....................................... 26
  4.2.4 Dependencies ....................................... 27
4.3 Front-end ................................................ 27
  4.3.1 PDF resume generation .............................. 28
  4.3.2 Dependencies ....................................... 29
4.4 Ranking algorithm ..................................... 29

5 Evaluation ............................................... 31
  5.1 Method summary ...................................... 31
  5.2 Result .................................................. 32

6 Conclusions ............................................. 37
  6.1 Restrictions ........................................... 37
  6.2 Limitations ........................................... 38
  6.3 Future work .......................................... 38

References ............................................... 41

A Algorithms used in the Learning to Rank implementation 45
  A.1 Dueling Bandit Gradient Descent ..................... 45
  A.2 Team-Draft method ................................... 46

B Screenshots ............................................. 47
List of Figures

2.1 An example of a simple search engine with Learning to Rank. ............... 7
2.2 An example illustration of training data. ..................................... 7
2.3 An illustration of the interactions between the user and ranking algorithm. . 10
2.4 An example illustration of an interleaved ranking. ............................ 12
2.5 An example of how NDCG is calculated for two ranking functions with four
   documents. ........................................................................... 14

4.1 An overview of the Resume Database architecture. .............................. 22
4.2 An example of the detail box on a user’s profile page. .......................... 23
4.3 Auto complete when searching. ....................................................... 23
4.4 A search result list. ...................................................................... 23
4.5 Irrelevant skills can be hidden before PDF generation. .......................... 24
4.6 An illustration of the relationships between the main classes used in the
   system regarding resume generation. ............................................ 25
4.7 A flowchart showing the PDF generation in the Resume Database. ........ 28
4.8 A flowchart showing how Online Learning to Rank is used in the Resume
   Database. ............................................................................... 30

5.1 Graph with results when running the ranking algorithm with exploration and
   exploitation step sizes set to 0. .................................................... 33
5.2 Graph illustrating the learning curves for all three click models when running
   1000 iterations, five times. ....................................................... 34
5.3 A smooth plot over the learning curve when learning the algorithm with the
   realistic click model and running 2000 iterations, 25 times. ................ 34

B.1 A screenshot of a profile page. ........................................................ 47
B.2 A screenshot of the edit pages. ......................................................... 48
B.3 A screenshot of the "edit and preview"-page for a resume, where preview
   mode is enabled. ....................................................................... 49
B.4 A screenshot of the "edit and preview"-page for a resume, where edit mode
   is enabled. ............................................................................... 50
B.5 A screenshot of the "edit and preview"-page for a resume, where edit mode is enabled. ................................................................. 51
List of Tables

2.1 Features that are engineered to be used by the ranking algorithm in the Resume Database. .................................................. 16
2.2 The relevance scale used in the evaluation of resumes for search queries. . . 16
2.3 Overview of the click models used to learn the algorithm for the evaluation. . 18

4.1 All dependencies used in the back-end system. . . . . . . . . . . . . . . . . . 27
4.2 All dependencies used in the front-end application. . . . . . . . . . . . . . . . 29

5.1 The final average NDCG scores and performance improvements for each click model after running an experiment consisting of 1000 iterations, five times. . 33
Chapter 1

Introduction

An introduction to the background, a detailed description of the problem, related work and an outline of the thesis report is presented in this chapter.

1.1 Background

This thesis work was conducted in Umeå at Knowit Norrland AB (hereafter only addressed as Knowit Norrland), a subsidiary to Knowit AB which is a consultancy firm that specializes in IT, Design and Digital Management [9].

In 2013 Knowit Norrland requested a system to manage resumes for their consultants. Up until this point all resumes had been stored and managed manually as documents, which was considered too inefficient and unnecessarily difficult. This resulted in the start of project 1117, where a prototype for a resume database was developed. Project 1117 was completed in June 2013 and the development of the Resume Database was put on hold until the start of this thesis work, in which further development of the system has been made. The preliminary work done in project 1117 has been used as a foundation for this thesis work, which has speeded up the preparation phase.

1.2 Problem statement

Today, Knowit Norrland is still managing their consultants’ resumes by storing them as documents on local computers. This forces the salespersons to manually go through all resumes to find the best match when assigning a consultant to a new project. Hereafter, a user of the Resume Database addresses both a consultant and a salesperson, and a searching user is the same as a salesperson. In addition to this, the resumes must often be edited manually to highlight relevant details and remove irrelevant ones before handing them over to the project owner. This is both time consuming and inefficient, and also why the primary focus of this report is put on the search engine. The importance of having an efficient and precise search engine is easily illustrated with the following example: Imagine if the Resume Database returns a list of dozens, or hundreds of resumes that are all somewhat...
relevant as a response to a query. This is a reasonable and likely event that can occur in reality, but without any ranking of the resumes it is also an inconvenient problem for the user. If this is the case, much work is required by the user to find the best match, since he/she has to manually go through all the resumes. It is easy to conclude that this would hardly be appreciated by the user because of how time consuming the process is. However, it is important to note that if the search engine ranks the resumes - it is vital that the ranking is relevant to the query and the user’s intentions. In other words, this puts a great responsibility on the system - to present satisfying rankings, but makes the process much easier and more efficient for the user.

Two main functionalities are prioritized in this thesis, the search functionality - which matches resumes with projects, and the generate functionality - which generates resumes with only the essential information for specific projects and provides them as documents. For the in-depth study, the following question was stated: how will the system make use of efficient search and resume generation functionalities, that meet the users’ needs?

Learning to Rank is a Machine Learning method used to solve the problem of ranking, without the need to manually design a ranking function, but instead learn this from users of the system. This method will be examined to see whether it can realize the envisioned functionalities for the search engine in the Resume Database. One of the most important parts of this thesis work will therefore be to adapt the algorithm to the specific requirements, that are set on the system. The research questions addressed are:

1. How will the system learn to have the most efficient and precise search engine with the help of Learning to Rank?
2. Which features in the resumes should the algorithm use when ranking?
3. How will the ranking algorithm be evaluated, so that we know that the system is learning and improving?
4. How good is the performance of the algorithm when it has received a specific amount of feedback?
5. Does the Resume Database actually benefit more from the Learning to Rank algorithm than from a simple and static ranking algorithm\(^2\)?

1.3 Purpose and goals

The purpose of the project is to streamline and simplify the management of the consultants’ resumes at Knowit Norrland and evaluate if a Learning to Rank algorithm can work efficiently in the system.

The goal is to design and implement a resume database that will assist the salespersons at Knowit Norrland in their work of selecting the best matching consultant for a project. Because of time constraints, the front-end of the Resume Database will not be completely implemented, but instead serve as a high-fidelity prototype [31]. In the future this prototype can be further developed or serve as a guideline when integrating with the business logic and database of the system. To make sure that the system meets the requirements that are set, the search engine is prioritized and a Learning to Rank algorithm will be implemented. The goal of this implementation is to investigate if Learning to Rank is applicable in the Resume Database and evaluate the performance.

\(^2\)A ranking algorithm that is not using Machine Learning
1.4 Related work

The initial requirements stated for the Resume Database and the general goals that are set for the project are summarized below.

Resume database requirements

– A consultant shall be able to create a profile and add resume details, such as competences and skills.

– A salesperson shall be able to search all consultants’ resumes for specific competences and skills.

– A salesperson shall be able to generate a consultant’s resume as a document, with only the necessary details for a specific project.

General goals

– Implement the back-end system fulfilling all requirements stated above.

– Implement a prototype of the front-end application with most of the requirements stated.

– Implement a Learning to Rank algorithm and evaluate the performance.

1.4 Related work

There is a lot of interesting research happening in the area of Learning to Rank today. It seems like listwise approaches, examples are ListNET[3] and Dueling Bandit Gradient Descent [37], and pairwise approaches, such as SVMRank [14] are the most promising algorithms in Information Retrieval. In this thesis the problem of matching consultants’ resumes with projects is considered very similar to the ones in Information Retrieval. This is the main motivation why such an algorithm is implemented and evaluated in the search engine of the Resume Database.

The company Yelp implemented a pointwise Learning to Rank algorithm for their business matching problem in 2014 [33] with Elasticsearch\(^3\) as the core search engine. Their evaluation results showed that by using Learning to Rank their retrieval system’s matching quality significantly improved and also became more flexible, stable and powerful. The business matching problem is very similar to the one addressed in this thesis.

An e-recruitment system implemented as a web application, the subject of a paper [6], was found similar in many ways to the Resume Database implemented in this thesis. That system extracts information from the applicants LinkedIn accounts as well as their personal blogs. Methods similar to these are discussed as future work in the Resume Database, in section 6.3. The implemented system is stated to use a Learning to Rank process, but the information about this is limited in the paper.

Another related work called Learning to Rank Resumes [25] briefly touches the problem of ranking resumes in a resume search engine. This work features an experiment with the pairwise Learning to Rank algorithm SVMrank [14]. The conclusion of the work was that the problem of ranking resumes was identified and that ranking with SVMrank could be done with good accuracy on approximate models of human relevance judgement.

\(^3\)https://www.elastic.co/
Finally, a very recent related work published in 2015 by Mario Kokkodis, Panagiotis Papadimitriou and Panagiotis G. Ipeirotis showcased three approaches that rank freelancing applicants on their hiring probabilities in an Online Labor Marketplace\textsuperscript{4} [18]. In their paper they argue that Learning to Rank can not be implemented as-is for their particular problem, since they lack multiple ranks. The scenario they are faced with only observes whether or not an applicant got hired and not in terms of which applicant is better than the other. They conclude that the hiring decision problem is very close to the "product search problem", as in [19], and base their work on this conclusion.

The difference in the Resume Database and the related works listed above is that the Learning to Rank algorithm, implemented in this system, will learn from implicit feedback in an online setting. In other words, this thesis focuses on implementing an Online Learning to Rank algorithm in a resume search (or recruitment) system and on performing an evaluation to find out the ranking accuracy of this approach.

1.5 Outline

The rest of the thesis is outlined as follows: Chapter 2 discusses the decision making for the in-depth study and gives a description of Learning to Rank. Chapter 3 explains how the project was planned and how the work was carried out. Chapter 4 presents the results obtained during the thesis work. Both the implementation of the Resume Database and the Learning to Rank algorithm is presented. Chapter 5 presents the results from the evaluation of the ranking algorithm, along with a discussion. Finally, Chapter 6 summarizes the thesis with reflections about the work, the conclusions that has been made and examples of further work.

\textsuperscript{4}The paper gives oDesk.com and Freelancer.com as examples OLM's.
Chapter 2

Method

This chapter starts out with a discussion on the decision for the in-depth study and a description of Learning to Rank. The focus here is on how Learning to Rank can be applied in the search engine of the Resume Database, that is to be implemented. Note that the content given about Learning to Rank in this thesis is not complete\(^1\).

2.1 Introduction

The primary goal of the in-depth study was initially set to answer the question “How will the system make use of efficient skill matching and resume generation functionalities that meet the users’ needs?” It is important to understand that the emphasis is put on the last part of the question, \textit{that meet the users’ needs}. How do you implement the system to find the best suited consultant and generate only necessary details in the resumes, according to the user?

Let us talk about the skill matching in the Resume Database from the programmer’s perspective. If only a boolean model \([7, 36]\) is used to match consultants on the query terms entered by the user, a problem arises. How should the system be able to suggest the best consultant for a specific project if the search query entered to the system match several resumes? If there is more than one document retrieved by the model they are indistinguishable and considered equally relevant to the entered query. A solution for this, is that the system could compare the consultants’ resumes on other details (or features) than the skills entered by the user. If two consultants are matched on their skills, but one is more experienced in terms of, for example, education, certificates, earlier employments or completed projects tied to the query, that consultant should be valued higher. But, how will the system (programmer implicit) know which of these \textit{hidden details} in the resumes that are relevant to the specific entered query? How is this relevance valued? Can this relevance change over time? If this responsibility is put on the programmer, the ranking is likely biased towards the programmer’s own preferences; and even if it is not, how can one know for sure that the users’ needs are met? Does research on the users of the system need to be carried out to achieve this?

With these questions in mind it is evident that a ranking function is needed and that Machine Learning - the sub-field of Artificial Intelligence concerned with programs that learn from experience \([29]\) - is of particular interest. If the Resume Database can learn from

\(^1\) Tie-Yan Liu’s literature \([21]\) contains more elaborate information on Learning to Rank.
the user, how to rank the resumes depending on the query, a solution might be close.

A look-up into the field of Machine Learning and surrounding areas resulted in the discovery of Learning to Rank which seemed to be a good match for this particular problem.

## 2.2 Learning to Rank

Because of the rapid growth of the Web today the efficiency of Information Retrieval on the Web has become more important than ever [21]. Therefore one of the more vital tasks in Web search engines, such as Google or Yahoo!, has become that of ranking. The ranker (ranking function) in Web search engines orders the documents that are retrieved for a given search query that should be presented to the user. This is necessary because of the oftentimes huge result lists that are retrieved from the search engines. Anne Schuth explains it the following way in one of his talks [30]:

“If a user comes to you with their query and you have 5 trillion matching documents you don’t want to put the document the user is looking for on the billionth result page.”

Until recently, the rankers were developed manually, based on expert knowledge. This might work in some applications, but in many cases a good ranking is dependent on the search context such as users’ age, location, and their specific search goals and intents [11]. Addressing each of these settings manually is infeasible and has lead to ranking functions with Machine Learning algorithms that can automatically tune these parameters. This and the combining of predefined features for ranking is what Learning to Rank methods does [21].

An important note is that Learning to Rank is not just used in Web search engines, but can also be applied to several other search tasks. However, when Learning to Rank is addressed in general in this chapter the example of Document Retrieval will be adopted.

The purpose of Learning to Rank is to learn a ranking function that produces satisfying rankings according to the user. This learning can be done with the help of Machine Learning in different ways. But, the Learning to Rank algorithms are often learned in a supervised manner and uses training and testing phases [20]. A supervised algorithm is learned by an explicit training phase to produce similar rankings as those presented by the training data, which are verified with the test data [22]. The training and test data in supervised learning for Document Retrieval are represented as sets of documents and queries, with a grade for each document that represents the relevance to a specific query. These gradings are the basic components used in the training phase that makes the Learning to Rank algorithms able to learn in this setting.

A supervised Machine Learned search engine is illustrated in figure 2.1. When a user query is posed to the system a set of relevant documents are extracted from all of the indexed documents, this is called the Top-k document retrieval. This phase can consist of, for example, a fast and simple boolean model [36]. After this the set of documents are ordered by the ranking model where the most relevant documents are put at the top, before presented to the user. This ranking model is machine-learned with training data that consists of queries and documents. Each query in the training set is associated with a number of documents and a relevance score for each document with respect to the query. An example of this is seen in figure 2.2, where the search query Learning to Rank resumes is identified with id 1 and the other search query Modo hockey arena with id 2. Four documents, two for each query is also illustrated, with their specific relevance score to the query. A similar, but
often smaller, set can also be used as test data to measure the performance of the ranker, and to verify that it produces satisfying rankings, after it has been learned [29].

![Figure 2.1: An example of a simple search engine with Learning to Rank.](image1)

2.2.1 Feature vectors

Tie-Yan Liu [21] summarizes Learning to Rank algorithms by having two properties and defines them as being Feature Based and having Discriminative Training. Feature based means that the documents under investigation are represented by feature vectors. These features can be divided into the three groups listed below [21]:

![Figure 2.2: An example illustration of training data.](image2)
- **Query features** or query level features - only depend on the query.
  Example: type and length of the query or properties of the user.

- **Document features** or query-independent features - only depend on the document.
  Example: length of the document or importance of the document.

- **Query-Document features** or query-dependent (dynamic) features - depend on both the document and query.
  Example: the frequency of the query terms in the document.

Some ranking models that are often used as features in Document Retrieval include the outputs of the BM25\(^2\) model and the PageRank\(^3\) model.

The method of selecting and designing good feature vectors is called feature engineering. In section 2.5.1 the feature engineering for the Resume Database is covered.

### 2.2.2 Discriminative Training

The other property, besides being Feature Based, that Liu [21] mentions, when summarizing Learning to Rank algorithms is Discriminative Training. It means that the learning process can be described by four key components\(^4\): input space, output space, hypothesis space and loss function.

Discriminative training is an automatic learning process based on the training data where the way of combining and weighting the relevance of the features, such that the output of the hypothesis function (mapping function) can predict the scores in the training set, is how the ranking model learns. In other words, Learning to Rank algorithms are trained to model the dependence of unobserved future data, on the training data. To give an example, when learning a linear ranking function a weight vector is used, with a weight for each feature, that is extracted from the documents. These weights represent the importance of each feature and are adjusted, or learned, by the Learning to Rank algorithm. The weight vector is used to compute scores, for the documents under investigation, that are used to rank the documents.

In order to better understand the Learning to Rank algorithms Liu categorizes them into three approaches: the **pointwise approach**, the **pairwise approach** and the **listwise approach**. The discriminative training differs for all of these approaches.

### 2.2.3 Ranking algorithm approaches

The form and semantics of the feature vectors and scores differ between the Learning to Rank approaches. These are divided into pointwise, pairwise and listwise approaches by Liu [21]. In this work a listwise approach is selected as the Online Learning to Rank algorithm, to be implemented, in the Resume Database.

**Pointwise**

Pointwise Learning to Rank takes feature vectors of individual documents as input space and learns a mapping for each relevance degree as output. The ranking problem is transformed into classification, where a binary relevance score is used, or regression, with a continuous

---

\(^3\)[http://en.wikipedia.org/wiki/PageRank]
\(^4\)These are further explained by Liu in his literature [21]
relevance score. To further explain, with classification a document can be predicted to be relevant or not, whilst regression approaches can give a degree of relevance for a document. A disadvantage of both formulations is that they do not correspond well to the Information Retrieval ranking problem. In Learning to Rank for Information Retrieval, the order in which documents are placed is crucial, while an exact prediction of relevance values is not.

Pairwise

Pairwise Learning to Rank approaches operate on pairs of documents, i.e., they take as input pairs of document feature vectors for a given query. The ranking is transformed into pairwise classification or pairwise regression. These pairs are mapped to binary labels, e.g., $y \in \{-1,1\}$. This would indicate whether the two documents under investigation are presented in the correct order as 1, or should be switched as -1. In the extreme case, if all the document pairs are correctly classified, all the documents are correctly ranked.

Listwise

The listwise approach addresses the ranking problem in a more straightforward way. It operates on complete result rankings, i.e. ranked lists. These approaches take as input the $n$-dimensional feature vectors of all $m$ candidate documents for a given query $(x_1,q,...,x_m,q) \in \mathbb{R}^{n \times m}$, and learn to predict either the scores for all candidate documents, or complete permutations of documents. The idea is that a ranking function’s constructed list is compared to the ground truth ranked list and updated accordingly to produce the ideal ranking. *Dueling Bandit Gradient Descent* is a listwise approach algorithm that is used in the Resume Database and is explained more in section 2.3.4.

2.3 Online Learning to Rank

So far in this thesis Learning to Rank has only been described as a supervised learning task (which it is traditionally), where the algorithm is trained in a batch mode. This approach is sometimes called Offline Learning to Rank where the learning and evaluation phases are done in an offline setting. There are some issues with this approach such as that the training data has to be annotated and labelled. This can both be expensive and difficult and may be biased towards the assessors instead of the users [5]. For the Resume Database this issue also applies. Imagine having to label resumes for a training set - it can be a difficult task to decide to label a resume as a perfect match or just as a good match for a particular query. The fact that a supervised Learning to Rank algorithm is only learned once and does not continue to learn, is also seen as an issue. What if the users’ interest change? An Offline Learning to Rank algorithm is not designed to adapt to this. To overcome these problems, weakly supervised approaches can help.

Online Learning to Rank is such an approach where learning is done by using real-time user click feedback. This way the system learns directly from the user and will dynamically adjust the ranker as long as the system is used.

2.3.1 User feedback

Online Learning to Rank algorithms are designed to learn from user feedback. Instead of relying on a traditional training phase where annotators must label the training data the algorithm learns directly from the user of the system. One of the first methods to use
user feedback in a system was introduced by Rocchio [28]. This method was introduced as *relevance feedback* and made the users able to communicate their evaluation to the system after every operation. Relevance feedback is an example of explicit feedback, which as the name suggests is collected from custom interactions in the system. This makes explicit feedback expensive for the users, since it takes both their time and effort. Instead, implicit feedback can be used, which is extracted directly from the users’ natural interactions with the system. An early approach learning from this type of feedback was presented by Joachims [13], which proved that it can be used to improve ranking in search engines. Examples of implicit feedback are clicks, mouse movement and dwell time. Mouse clicks is a good choice of implicit feedback compared to the others, since large quantities can be collected at a low cost [11]. An illustration of the interaction between the user and ranker (ranking algorithm) where mouse clicks are evaluated can be seen in figure 2.3. The user issues a query to the system, which returns a ranked list. Once the user clicks a document in the list, the click is registered and evaluated, which is used to re-learn and update the ranking function.

![Figure 2.3: An illustration of the interactions between the user and ranking algorithm.](image-url)

One important note when using mouse clicks is to interpret them as relative feedback, instead of absolute feedback. Relative feedback can be explained by saying that a clicked document is more relevant to a query than some other (non-clicked) document. Absolute feedback on the other hand, means that a clicked document is, or is not, relevant to a query. Results indicate that the users’ clicking decisions are biased by the order in which the ranked documents are presented [15]. This means that if the ranking is not perfect, the users might not always click the most relevant document because it is not presented at the top of the list. Because of this, absolute feedback is not ideal, since you can not know how relevant the clicked document is by itself to the query. Relative feedback can still be valuable though, because it is easier to know if a clicked document is more or less relevant than a preceding non-clicked document.
The position bias mentioned here can still be a problem for the ranking algorithm, even when relative feedback is used. How this is usually solved is explained in the following section.

2.3.2 Exploitation versus exploration

The position bias is often explained by saying that top results are clicked more often than other results. This is because the user expects more relevant documents to be listed at the top and because people are used to reading pages from top to bottom. Eye-tracking studies have confirmed this and given some interesting results about how the rank influence the attention of a user. Results of a study performed with Google on students at a university in North America [8] showed that the mean time a user looks at link 1 and 2 in a result list is almost equal, but the link ranked first is substantially more often clicked. The results also show that rank becomes much less of an influence for attention when the user has to scroll or change page to study more links. The conclusion of this implies that the position bias is a real and important problem that must be addressed.

Put simply, the ranking algorithm can not always rank based on what it has learned so far (exploit), it also needs to explore other solutions and add different documents to the list [12]. This is what exploitation versus exploration refers to. The ranking algorithm needs to balance what it already has learned with new solutions to continuously learn the best possible ranking in an effective way. If only documents that are expected to satisfy the user is presented, it cannot obtain feedback on other, potentially better documents. However, if only documents that the algorithm can gain a lot of new information from is presented, it risks presenting bad results to the user during learning. This is not unique in Information Retrieval or Learning to Rank. Balancing exploitation and exploration is considered important in Reinforcement Learning as well [17].

It has been proven that balancing exploitation and exploration can significantly improve the performance of Online Learning to Rank. The effect of balancing exploration and exploitation is complex but it is concluded in [12, 11] that more or less exploration, depending on how reliable the feedback to the algorithm is, can improve learning. This type of finer balancing of exploration and exploitation is not implemented in the Resume Database at this time.

2.3.3 Interleaved comparison methods

With an Online Learning to Rank algorithm implemented with a listwise approach a comparison method to evaluate the quality of two rankings is needed. It is obvious that both rankings can not be presented to the user side-by-side to evaluate the best ranking [26]. Instead an interleaved comparison method is often used, which first has the task of interleaving the ranked lists into one list that is presented to the user, and later the task of evaluating the clicks made by the user. As an example, the interleaving of two ranked lists when searching on "Modo Hockey fans"\textsuperscript{5}, with two Web search engines, is illustrated in figure 2.4. There have been several methods proposed, such as Balanced Interleave, Team-Draft, Document Constraints and Probabilistic Interleave. Balanced Interleave and Team-Draft, two methods that have been shown to work reliably and efficient in practice [4], are described below.

\textsuperscript{5}Modo Hockey is a swedish ice hockey team from Örnsköldsvik - my home town.
Balanced Interleave

The Balanced Interleave (or balanced interleaving) method was proposed [13] to interleave two rankings into one in a balanced way. First, one of the lists is randomly selected to start to contribute its top-ranked document that is not yet part of the interleaved list. Then, the other list does the same by contributing its highest ranked document that is not yet in the interleaved list. This continues until the lists are empty or the interleaved list is fully constructed. When this is done the interleaved list is presented to the user and clicks are recorded. A click is counted towards the original list where the clicked document is ranked the highest. Ties are broken randomly. The original list that gets the most clicks among its top-ranked documents is declared the winner and the learning algorithm is updated accordingly.

Unfortunately the Balanced Interleave method can potentially lead to biased results in some cases. This is made obvious by Radlinski, et al. [26] who proposed the Team-Draft method as a substitute.

Team-Draft

To correct the bias problem in Balanced Interleave a similar comparison method was proposed called Team-Draft [26]. This method follows the analogy of selecting teams for a friendly team-sports match, hence the name Team-Draft. The difference compared to the Balanced Interleave method is minor, but significant. With the Team-Draft method, it is not just at the start that a list is randomly selected to contribute its highest ranked document, but at every new round. This method also remembers which list that each document is contributed from. This is done with an assignment, which later is used during the evaluation - instead of then identifying which list that has the clicked document ranked the highest. To compare the two lists, the clicks are counted towards the list that contributed those documents. This ensures that each list has an equal chance of being assigned clicks. The Team-Draft algorithm implemented in the Resume Database is summarized in algorithm A.2 in the appendices.
2.3.4 Dueling Bandit Gradient Descent

Dueling Bandit Gradient Descent [37] is a listwise algorithm that has been specifically developed for Learning to Rank in an online setting. It is designed to compare the quality of two document lists from implicit feedback. To summarize, DBGD optimizes a weight vector which represents the importance of each feature that is used in the score calculation. The algorithm maintains a candidate \( w_t \) as source weight vector and compares it with a different weight vector \( w'_t \) along a random direction \( u_t \). If \( w'_t \) wins the comparison, the source weight vector \( w_t \) is updated along \( u_t \). Two parameters are required, the exploration and exploitation step sizes, which impact how much the algorithm explores each step and how much the exploitative weight vector is updated.

An iteration in the DBGD algorithm, i.e. each time a search query is handled by the system, can be described as follows:

1. (The first time the algorithm is run, an initial weight vector, \( w_0 \), is set as the exploitative source weight vector \( w_t \))
2. A search query is received and an exploratory weight vector, \( w'_t \) is constructed with a uniformly sampled unit vector and an exploration step size parameter.
3. Two lists are constructed; one exploitative list with \( w_t \), called \( l_1 \) and one exploratory list with \( w'_t \), \( l_2 \).
4. A new interleaved list \( L \) is constructed by the Team-Draft method with \( l_1 \) and \( l_2 \).
5. The top-ranked results in \( L \) is presented to the user.
6. If items assigned to the explorative list, \( l_2 \), was clicked the most, \( w_t \) is updated in the direction of \( w'_t \) with an exploitative step size.
7. The new best weight vector \( w_t \) is persisted and used for the next iteration.

The DBGD algorithm implemented in the Resume Database is presented in the appendices as algorithm A.1.

2.4 Evaluation measures

Several evaluation measures are used in Information Retrieval and to evaluate Learning to Rank algorithms. One of these are Normalized Discounted Cumulative Gain [16] which will be covered in this section and later used in the evaluation of the Learning to Rank algorithm in the Resume Database. To fully understand NDCG one must first understand what Discounted Cumulative Gain is.

Assume that documents are ranked based on relevance scores and highly relevant documents are more valuable than marginally relevant documents, for the user. This implies that if the relevance scores, for all documents in a result list, are summed, the total relevance of the documents in the result list is evaluated, but not the ranking. Hence, two result lists containing the same documents, but ranked differently, will get the same summarized score. To overcome this problem and actually evaluate the ranking of a result list, a discount factor is used for each rank, hence the name Discounted Cumulative Gain.

Using a graded relevance scale of documents in a search engine result set, DCG measures the usefulness, or gain, of a document based on its position in the result list. The gain is accumulated from the top of the result list to the bottom with the gain of each result
discounted at lower ranks. In other words, a smaller share of the document score is added to the cumulated gain for greater ranks, than for lower ranks [16]. A simple way of discounting the document score, as its rank increases, is to divide the document score by the log of its rank. This produces a smooth reduction, in comparison with dividing with just the rank. For example $2^{log\ 2} = 1$ and $2^{log\ 1024} = 10$, thus a document at position 1024 would still get one tenth of its face value.

The equations in this section are not presented as they were originally [16], but instead as in [32]. DCG at a particular rank position $p$ can be expressed in formula 2.1 (the logarithmic with base 2 is used).

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{log_2(i)}$$ (2.1)

Where $rel_i$ is the graded relevance of the result at position $i$ and the document score at the highest ranked position does not need to be discounted, hence $rel_1$ is not divided with the logarithmic. However DCG can not be used alone to compare a search engine’s performance from one query to the next since the gain at each position is not normalized across queries. This is why NDCG is used instead, which can be calculated as in formula 2.2.

$$NDCG_p = \frac{DCG_p}{IDCG_p}$$ (2.2)

Where $IDCG_p$ refers to the idealized $DCG_p$.

This means that in a perfect ranking algorithm, the $DCG_p$ will be the same as the $IDCG_p$, producing a NDCG score of 1. All NDCG values are on the interval 0.0 to 1.0 and are therefore cross-query comparable. A simple example with NDCG, that illustrates how it is calculated can be seen in figure 2.5.

**Figure 2.5:** An example of how NDCG is calculated for two ranking functions with four documents.
2.5 Ranking algorithm for the Resume Database

As already concluded, the benefits of not having to label training data but instead learn directly from the users clicks in the system is a major advantage for the Resume Database. This argument as well as the personal interest in realising Machine Learning in a system constitutes the foundation for choosing to implement an Online Learning to Rank algorithm in the Resume Database. This also answers Research Question 1 stated in section 1.2, how the system will learn to have the most efficient and precise search engine with Learning to Rank.

If we look back on the questions posed at the beginning of this chapter, it is certain that Online Learning to Rank can be a good solution that answers the majority of them. With this algorithm implemented it is not up to the system or programmer to know which details that are most relevant to use in a ranking. Manual adjustments to the system should not be necessary when taking into account that the relevance for these details can change over time. The user is not forced to manually go through the complete search result to decide the best or most relevant resume each time searching, but can instead rely on the ranking by the algorithm that is an extension of the user’s previously selected preferences. An Online Learning to Rank algorithm does all this, and most importantly it does satisfy the users’ needs. The notion that the ranking is continuously learning from the users via their natural interactions with the system is an ideal solution in theory.

A Dueling Bandit Gradient Descent algorithm with a Team-Draft method is implemented in the system. Only a few adjustments to the main functionality and data used for the system are needed to integrate the algorithm efficiently. The Team-Draft interleaved comparison method is chosen because it does not suffer from the bias problem found in Balanced Interleave and is easy to implement. The user click that is made when generating a resume as a document is used as the implicit feedback to learn the ranking algorithm. This click is considered to be made by a user that has decided that the selected resume is the best choice and therefore highly relevant to the issued search query. No other clicks in the system are registered or used as feedback to the algorithm.

The computation of scores for the resumes is simple and done linearly, meaning that all extracted features will be multiplied with its weight and summed into one total score for each resume.

In section 4.4 the result of the implementation of the ranking algorithm is presented.

2.5.1 Feature engineering

Feature engineering is the process of selecting and determining which features that the algorithm should calculate the ranking on. Research Question 2 in section 1.2 is therefore addressed here. Some examples of features that can be used by the ranking algorithm in the Resume Database are listed in table 2.1. The features are described as large-grained and generic here, but can be implemented more fine-grained in the system. Note that not all of the features listed are used by the Resume Database and some of them that are used have been disassembled into several sub-features. An example is the skill match feature, which is split in the system into smaller features depending on if the skill is tied to a project, education or certificate - making it possible to weigh the importance of these differently.
Feature Description

**Document features**
- Importance: How important the consultant is to the company.
- Experience: How experienced the consultant is based on earlier projects.
- Completeness: How complete the consultant’s resume is.
- Popularity: How often the consultant is chosen (irrespective of the query).
- Salary: How expensive the consultant is.

**Query-document features**
- Skill match: How far the consultant’s skills/competences match the query.
- Text match: How far the textual content match the query (Apache Lucene TF-IDF).
- Popularity: How often the consultant is chosen with respect to the query.
- Up-to-date: How up-to-date the consultant is regarding skills/competences.
- Availability: Is the consultant available for the project?

Table 2.1: Features that are engineered to be used by the ranking algorithm in the Resume Database.

The features described in table 2.1 are mimicked on so-called *document features* and *query-document features*. *Query-features*, the last feature type described in section 2.2.1, are not used in the Resume Database. This is because the ranking model is linear and these features are the same for all resumes under a query and therefore do not impact the scores.

### 2.5.2 Evaluation methodology

The evaluation of the ranking algorithm in the Resume Database is using Normal Discounted Cumulative Gain. This evaluation metric has been proven to be reliable with Learning to Rank [2]. Research Question 3 in section 1.2, concerning how the ranking algorithm should be evaluated to know if it is learning and improving, is therefore addressed in this section.

**Preliminaries**

If a Web search engine is implemented with a Learning to Rank algorithm, the evaluation can make use of several publicly available sets of graded documents and queries as training and test data. This is however not an option for this thesis work, since no pre-existing, publicly available data has been found that can be used for the Resume Database, prior to the evaluation. The training and test data for this evaluation is instead generated manually and graded on a relevance scale between 1-5, with the help of future expert users of the Resume Database. These gradings can then be used to create rankings of the resumes that is seen as truth tables for each query. In table 2.2 the relevance scale, used in the evaluation, is presented.

<table>
<thead>
<tr>
<th>Relevance</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Perfect match</td>
</tr>
<tr>
<td>4</td>
<td>Good match</td>
</tr>
<tr>
<td>3</td>
<td>Fairly relevant</td>
</tr>
<tr>
<td>2</td>
<td>Minimally relevant</td>
</tr>
<tr>
<td>1</td>
<td>Completely irrelevant</td>
</tr>
</tbody>
</table>

Table 2.2: The relevance scale used in the evaluation of resumes for search queries.

---


[9]
The resumes and search queries are generated by selecting random features from sets of skills, professions, projects and earlier employments. It is important that these resumes and search queries only contain data that the expert users can produce honest rankings on. Only features that the ranking algorithm use are presented in the resumes, which implies, for example, leaving out personal details such as name, sex and age. We do not want to present other features, that can affect the user’s gradings, so that their rankings can not be imitated by the ranking algorithm.

When the expert users have graded all resumes for each search query, rankings, based on these gradings, can be constructed and used as truth tables, or IDCG rankings (see section 2.4, for the evaluation.

**Evaluation phase**

When the preliminary work is done for the evaluation and the IDCG values are computed, the ranking algorithm with an initial weight vector can be evaluated. This initial weight vector will be set to zero which is considered equivalent to an unlearned algorithm\(^7\). The evaluation then proceeds by loading the Resume Database with the same set of resumes that the expert users graded earlier and for each search query that the expert users had available, calculate the NDCG score for the rankings produced by the algorithm. All NDCG scores are averaged into one score, which can be used as a measure on how good the algorithm is ranking at this particular time.

After this the algorithm can be learned, both by a click model and by real users, on a training set of resumes and search queries that also are graded to know their real relevance grades. A click model can simulate users’ click behaviour in a system and is further explained below. Why a click model is used is because the learning phase, when the algorithm is completely untrained, is not well-suited for real users. Having real user’s interact with the system with an unlearned algorithm would most likely be a very bothersome and time-consuming task. During and after the learning phase, the algorithm’s ranking accuracy is evaluated. This is done by computing NDCG scores for the rankings generated by the algorithm with the test data. These scores can then be inspected, to find out if the implemented algorithm actually is learning a better ranking or not.

**Click model**

A click model is used to simulate real users’ click behaviour in a system. It can be used to systematically simulate clicks to learn a ranking algorithm for evaluation purposes. The click model explained in [11], which is based on the Dependent Click Model [10], is used in the evaluation of the Resume Database. This model is taking advantage of that users start examining at the top of a result list and for each item examined, they determine if it seems promising enough to click on it and if the clicked item is enough satisfactory to stop examining further results. Therefore, both click, \(P(C|R)\), and stop, \(P(S|R)\), probabilities usually need to be defined based on the relevance of the items. However for the evaluation in this thesis only the click probability is used, since the stop probability is always 1 for all click models. This is because the Resume Database does not use other feedback than the click when a consultant’s resume is selected to be generated as a PDF, i.e. only one click is registered and used as feedback. See section 4.4 for further explanation.

As explained in [11] several instantiations of click models can be used to simulate different types of user behaviours, ranging from very reliable to very noisy click behaviour. In this

\(^7\)Later in production the initial weight vector can be set with values that by experiments are known to produce good and satisfying rankings from start.
thesis a perfect, realistic and almost random click model is used for the evaluation and these are defined with the relevance grades (1-5) used by the expert users during the initial evaluation phase, when grading training and test sets. An overview of the resulting click models can be seen in table 2.3.

<table>
<thead>
<tr>
<th>relevance grade</th>
<th>click probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>perfect</td>
<td>0.0 0.2 0.4 0.8 1.0</td>
</tr>
<tr>
<td>realistic</td>
<td>0.1 0.3 0.5 0.65 0.85</td>
</tr>
<tr>
<td>almost random</td>
<td>0.4 0.45 0.5 0.55 0.6</td>
</tr>
</tbody>
</table>

Table 2.3: Overview of the click models used to learn the algorithm for the evaluation.

The perfect click model can be seen as an upper bound on the performance (a "perfect clicking user"), which always clicks on perfect matching resumes and never clicks on completely irrelevant resumes. On the other side, the almost random click model is seen as a lower bound on the performance and has a very small linear decay in the click probabilities for the different relevance grades. The realistic click model is constructed to approximately or roughly simulate the clicking behaviour of a real user in the Resume Database, i.e a salesperson at Knowit Norrland.
Chapter 3

Work process

In this chapter, the preliminary work is described, along with an explanation of how the project was carried out. The work has been divided into two parts, one for the implementation of the ranking algorithm and one for the implementation of the rest of the system.

3.1 Preliminaries

At the start of the project, focus was put on evaluating frameworks, practical models, libraries and languages to be used in the implementation of the Resume Database. This was possible because a major part of the requirements on the system was clear from the beginning, since the preparatory work from Project 1117 was available. It was early decided that a REST API middleware would be implemented to decouple the system’s front-end and back-end, as well as that the implementation of the back-end would be prioritized and that the front-end would only be implemented as a prototype. After this preliminary evaluation, some time was spent on getting acquainted with the chosen frameworks and libraries.

The in-depth study, which surrounded Online Learning to Rank, was also initially focused upon, during the beginning of the project. At the end of this period a first draft for the ranking algorithm was designed, in addition to some work on the feature engineering. Plans for the evaluation phase were also established during this time.

After gathering this background information the design and implementation stage was initiated.

3.2 How the work was carried out

The plan during the implementation stage of the Resume Database was to adopt the agile development method, Scrum\(^1\). This would have included iterative sprints, daily meetings, Kanban boards, and using a product backlog. However, during the start of this phase, this plan was changed and ended in a custom, but simple, agile method with just weekly meetings and demonstrations with direct feedback from Urban Holmgren and Andreas Hed at Knowit Norrland, who acted as product owners. Reasons for this decision were that the time available with the product owners did not seem sufficient enough, and the gain of using Scrum was not considered more beneficial than the time needed to work with the method. The implementation stage followed an iterative process where the main focus was

\(^1\)Read more here: [http://scrummethodology.com/](http://scrummethodology.com/) (last visited 2015-03-16)
on implementing only the most necessary features according to the product owners and future users.

At the beginning of the implementation some ground work was done to set up the base for both the back-end and front-end, and to have a minimal but working prototype of the system. After this the system could be further developed in small steps, continually adding functionalities based on feedback from the product owners, i.e. the future users of the system. When an initial demonstration of the system had been shown, the implementation continued by adding small building blocks to the prototype and continually demonstrating these to the product owners to get direct feedback.

The back-end system was prioritized during the implementation, but developed along-side with the front-end application. During the first weeks of the implementation stage the development of the ranking algorithm was put aside. It was only during the last weeks that the ranking algorithm was implemented and integrated into the system. Before the end of the project the ranking algorithm was evaluated with the help of Andreas Hed and Urban Holmgren as system experts.
Chapter 4

Results

In this chapter the results of the implementation of the Resume Database and Learning to Rank algorithm is presented.

4.1 System overview

Here a general overview of the Resume Database is presented. First the system architecture is described and second the main functionalities implemented in the system with companying screenshots.

4.1.1 System architecture

The resume database is developed from scratch with a back-end system written in Java, a MySQL database and a front-end application implemented as a website with AngularJS and Bootstrap. In figure 4.1 a system illustration over the Resume Database is presented. All dependencies, such as programming languages, libraries, frameworks and softwares, used for the back-end are listed in section 4.2.4 and for the front-end in section 4.3.2. The business logic and user interface are decoupled by communicating via a REST API. This decision was made to ease the implementation of a new front-end application and is discussed further in section 4.2.2.

The ranking algorithm is implemented as a proof of concept and is therefore also decoupled from the back-end system in its own sub-module. This module can be disabled at start-up of the back-end server. The decision was made because the goal is to use the system in a real setting and the ranking algorithm, as it is implemented today, is very unlikely to be suited for this purpose.
4.1.2 System functionalities

The resume database is using Knowit’s AD service for authentication, which implies that no explicit registration is needed by Knowit’s employees before starting to use the system. This is because each employee at Knowit Norrland is given their own AD account when hired. The first time an employee logs in to the Resume Database, with his AD credentials, a new user will be created, with the entered AD username.

Each user have their own profile page, which presents their merits and experiences, see image 4.2. This page is one of two central parts of the Resume Database. If a user visits a consultant’s profile page, there is an option to download that consultant’s resume as a PDF document and also to open an edition of that consultant’s resume, that has been generated earlier. These two functionalities are described further below. If the visited profile page is the user’s own there is also an option to edit personal and resume details.
4.1. System overview

The second major part of the system is the search engine. Here a user can search for resumes with skills, professions and custom keywords, which for example can be company names of earlier projects. The skills and professions are constants that can be added manually by the consultants when editing their resumes. This enables the use of auto-complete which is used on the search page to assist the searching user. An example of this is shown in image 4.3. The custom keywords can be entered as anything, since those are searched for in the text fields of projects such as the description field and name field. In the future these keywords could be searched for in several other text fields as well. Each search term, entered in the search field, is illustrated in a coloured block (or tag). In the advanced settings tab these tags can be marked as required or optional. A required tag must exist in a consultant’s resume while an optional tag must not, but is seen as a qualifying keyword and will only be used in the ranking of matching resumes.

When a search query is issued to the system the result will be a list of relevant consultants that have the searched, required keywords in their resumes. If the ranking module is enabled in the back-end system the returned list will be ranked, placing the most relevant consultants at the top of the list.

Each row in the result list will represent a consultant, where some general information will be presented, such as the relevant skills matching the search query and all professions that the consultant has (image 4.4). Information about which Knowit office they belong to, as well as their name and profile picture are also displayed. There are two alternatives represented with two buttons for each consultant in the result list. One button is used to open a modal with detailed information from the consultant’s resume, that can be used to quickly inspect a consultant’s relevance to the search query. The other button is used to get directly to the "edit and preview"-page of the consultant’s resume, where the user can choose to hide or show details.
in the resume and also generate and download it as a PDF. Each returned resume will have all relevant skills and professions, that matches the search query, highlighted and irrelevant ones hidden. This is implemented because experienced consultants may have large number of skills, projects, professions and other details added in the system, but only those that are searched for are relevant in a generated resume.

When a user chooses to download a resume as a PDF document, the selected consultant’s resume will first be opened in the "edit and preview"-page. On this page the resume can be edited, by manually hiding and editing resume details before generation, see image 4.5. The appearance of the resume at the preview page will be the same as the resulting PDF document. However, one important note is that all changes made to the resume at this page are only temporary and will not update the consultant’s original details. They will only be reflected on the generated PDF document. All generated PDF documents are stored on the server with a unique name and are therefore available for download at a later time. The resume edition created on the "edit and preview"-page will also be persisted with the unique name, which can be re-opened and modified as new editions later. Saved resume editions can be opened from the profile page of a consultant.

See the appendix, chapter B for more and larger screenshots of the system.

4.2 Back-end

The back-end system is implemented in Java, with a REST API that handles the communication, and a database to persist the data. All dependencies used in the implementation of the back-end system are listed in section 4.2.4.

The main components used in the Resume Database are the representations of a user and a resume, the relationship between these is one-to-one, meaning that a user has exactly one resume, and vice versa. However, when a resume is generated as a PDF document, a resume edition will be created and used in the generation process. These resume editions are modified versions of a user’s resume and have a many-to-one relationship to the original resume. In figure 4.6 these classes and their relationships are illustrated.

The resume generation is making use of the software wkhtmltopdf, which is capable of converting HTML files to PDF documents. An explanation of how the resume generation works in the system can be read in section 4.3.1.

4.2.1 Database

Different databases were evaluated before a MySQL database was chosen for the implementation of the Resume Database. These included the NoSQL and graph databases OrientDB\(^1\) and Neo4j\(^2\). A graph database was considered to fit the application good, especially in the sense of performance and simplicity for matching consultants in the search engine. However, the time limit for the thesis work as well as the choice of focusing on the ranking problem

\(^1\)http://orientDB.org  
\(^2\)http://neo4j.org
4.2. Back-end

ended in the decision of using a relational database instead. In section 6.1 this reasoning is further discussed. A potential transition to a graph database is discussed in section 6.3.

The communication between the back-end and database is done with the object-relational mapping framework Hibernate. All tables for the Resume Database can be automatically created by setting the `hibernate.hbm2ddl.auto` parameter to `create or update` in the Hibernate XML configuration. This setting will create the tables based on the Hibernate annotations set in the source code. At this time the parameter is set to `validate`, which will output a warning if the database structure is invalid.

In figure 4.1 the database API is illustrated, which uses Hibernate and is built with a custom pattern similar to a repository pattern [23] or data access object pattern [24]. This API is implemented to ease further development and a possible transition to a new database, if needed in the future. There is an abstract class called `BaseRepository` that makes use of generics and handles all standard CRUD methods, such as Create, Read, Update and Delete. This class uses Hibernate operations to communicate with the database, and should be extended with a custom repository, if further development of the system takes place. An example of this is the `User` object, used in the system, which makes use of its own `UserRepository`, which extends the `BaseRepository`.

4.2.2 API middleware

The API middleware is built with Restlet, a lightweight Web API framework for Java. With Restlet you can easily customize which server you want the API to be hosted on. The resume database uses a `Simple` HTTP server connector as the internal Web server to host the API. Changing the desired server connector is as easy as replacing the jar file included in the Maven project with another supported server connector [35].

![Diagram](image1.png)

Figure 4.6: An illustration of the relationships between the main classes used in the system regarding resume generation.
Authentication

The authentication is session-based by using HttpOnly cookies, which are non-standard but widely supported as explained in the CookieSetting documentation [27]. When authentication is made against /authenticate a cookie is created by the server and sent to the client. This cookie contains an encryption of the username of the authenticated user concatenated with an expiration time, and must be sent by the client with all subsequent calls to the API. The expiration time makes the cookie unusable after a period of time, which forces the client to re-authenticate. To destroy the cookie a call can be made against /logout which sets the expiration time of the cookie to 0 and triggers the cookie to be instantly removed at the client-side (this is often handled by modern browsers).

The authentication is making use of Knowit’s AD service that is used for other internal systems by all employees already. This decision was made both because it eases the account managing for the users of the Resume Database and the credential confidentiality for the back-end system. The passwords are stored in Knowit’s AD service while only the usernames are duplicated in the back-end of the Resume Database.

Along with this authentication method is also an alternative method implemented, which uses an API key, that can be passed as a parameter in the URL to the front-end. In section 4.3.1 this authentication method is explained more.

Securing the API with SSL is a requirement if the system is to be available outside of Knowit’s local network, since user credentials are sent via unencrypted POST messages to the Resume Database API. However, the communication between the Resume Database API and Knowit’s AD service is secured with SSL already. The alternative authentication method using the API key, on the other hand, is recommended to be reworked, if the back-end system and front-end application is located on different networks outside of Knowit’s local network. SSL encryption is not enough here, since the key is sent in plain-text as a parameter in the URL. But, as long as the system only is located and open for machines connected to Knowit’s private network, which is considered secure, and not put online, this is not seen as a significant vulnerability.

Authorization is not a feature in the system at this moment, but can easily be set up as suggested by the Restlet team [34]. This would enable the use of role authorization which could for example be used to handle administrators or setting different permissions for consultants and salespersons.

4.2.3 Search engine

The search engine in the Resume Database is using Hibernate Search, which integrates the full text indexing and searching library Apache Lucene for use with Hibernate. With Hibernate Search, all skills, professions, project names and project descriptions in the Resume Database are indexed, which offers a fast and reliable search for resumes. Experiments with 10000 consultants added in the system showed that search queries took less than a second (with the ranking module enabled), which is more than enough performance-wise in a practical setting for the Resume Database.

A search query that is handled by the back-end system can be composed by both required and optional keywords. These are combined in the back-end to construct a Lucene query that represents a boolean junction which can be looked-up against the set of indexes created by Hibernate Search.

3https://www.owasp.org/index.php/HttpOnly
4The attribute accessRestricted specifies the CookieSetting as an HttpOnly cookie.
5https://lucene.apache.org/core/
4.3 Front-end

Hibernate Search is using Apache Lucene for an internal scoring algorithm which makes use of a Vector Space Model to score documents and rank them by relevance. This scoring algorithm can be customized\(^6\), but in its default form, simply put, makes use of a variant of the TF-IDF formula \([1]\). In general this model scores based on how many times a query term appears in a document relative to the number of times the term appears in all the documents in the collection. Because the returned result list when searching with Hibernate Search is ranked with the Apache Lucene scoring algorithm, the set of resumes is always shuffled, if the ranking module is enabled, before being ranked. The scoring algorithm’s relevance score for each resume is instead leveraged and used as a feature by the implemented ranking algorithm, see section 4.4. If the ranking module is disabled in the back-end the set of resumes is never shuffled. Instead, the returned search result will be ranked by the TF-IDF scores.

### 4.2.4 Dependencies

The libraries, frameworks and relevant softwares used in the implementation of the back-end system are listed in table 4.1.

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>1.8</td>
</tr>
<tr>
<td>Restlet</td>
<td>2.3.1</td>
</tr>
<tr>
<td>MySQL</td>
<td>5.5.41</td>
</tr>
<tr>
<td>MySQL Connector/J</td>
<td>5.1.34</td>
</tr>
<tr>
<td>Hibernate</td>
<td>4.3.8</td>
</tr>
<tr>
<td>Hibernate Search</td>
<td>5.1.1</td>
</tr>
<tr>
<td>imgscalr</td>
<td>4.2</td>
</tr>
<tr>
<td>args4j</td>
<td>6.1.0 (beta 1)</td>
</tr>
<tr>
<td>docx4j</td>
<td>3.2.1</td>
</tr>
<tr>
<td>wkhtmltopdf</td>
<td>0.12.2.1</td>
</tr>
</tbody>
</table>

Table 4.1: All dependencies used in the back-end system.

### 4.3 Front-end

The front-end is a website implemented with AngularJS and Twitter Bootstrap. As stated earlier the front-end is only implemented as a prototype and therefore not providing all functionalities that are required by Knowit Norrland, at this time. All dependencies used in the implementation of the front-end application is listed in section 4.3.2.

At the writing of this report, the website is hosted on an Apache/2.4.7 HTTP server. This Web server is configured to rewrite all URLs that are not pointing at a static file to the `index.html` file. The configuration is made because `html5Mode`\(^7\) is enabled in AngularJS and the website will not work properly if a page reload is made without it.

---

\(^6\)The scoring algorithm can be overridden and customized for specific ranking scenarios. This is however classed as an expert user task in the documentation for Apache Lucene and is therefore not addressed in this thesis. Read chapter 6 for a discussion regarding this.

\(^7\)The reason why html5Mode is used in the system is described by Chris Sevilleja at https://scotch.io/quick-tips/pretty-urls-in-angularjs-removing-the-hashtag
4.3.1 PDF resume generation

In this section the resume generation as PDF with the software wkhtmltopdf is described. Even though the description is put under this section dedicated for the front-end, it describes the process in the back-end as well.

Wkhtmltopdf is an open source command line tool capable of generating a PDF document from an HTML page with just the URL to the HTML page. This software is installed on the same server as the back-end system for the Resume Database and is executed and handled by a special PDF generation module in the back-end.

In the front-end application it is possible to edit and preview a resume before downloading it as a document. This "edit and preview"-page has a unique url for each resume that is to be generated, and is ideal to be used by wkhtmltopdf to generate the PDF documents. However, since the website requires authentication and cookie sessions are used for this, an alternative authentication method is used when generating PDF documents.

There is an unique API key that can be passed in the URL when visiting the "edit and preview"-page, that authenticates the caller. This key should not be known by anyone except the PDF generation module and the authentication module in the back-end (where this key is also stored). The front-end does not need to know the correct key, but simply redirects the API key with all PDF generation calls to the back-end and lets this system do the authentication. If the call is not authenticated the front-end will redirect the caller to the login page. Note that some elements and some of the design on the "edit and preview"-page in the front-end application is hidden and changed when the API key is passed, in the URL, to generate a clean PDF. The process flow in the system when generating a PDF can be seen in figure 4.7.

As stated in section 4.2.2, this authentication method is not intended to be used, if the Resume Database is available outside of Knowit’s local network. This applies especially if the back-end system and front-end application are located on separate networks, where communication is done outside of Knowit’s private network. If this is the case, it is recommended to implement a different authentication method for the PDF generation module.

![Figure 4.7: A flowchart showing the PDF generation in the Resume Database.](image-url)
4.3.2 Dependencies

The libraries, frameworks and relevant softwares used in the implementation of the front-end application are listed in table 4.2.

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>JQuery</td>
<td>2.1.3</td>
</tr>
<tr>
<td>AngularJS</td>
<td>1.3.0</td>
</tr>
<tr>
<td>Twitter Bootstrap</td>
<td>3.3.4</td>
</tr>
<tr>
<td>UI Bootstrap</td>
<td>0.12.1</td>
</tr>
<tr>
<td>X-editable</td>
<td>1.5.1</td>
</tr>
<tr>
<td>ngTagsInput</td>
<td>2.3.0</td>
</tr>
<tr>
<td>Flow.js ng-flow</td>
<td>2.6.1</td>
</tr>
<tr>
<td>angular-unsavedChanges</td>
<td>0.2.3</td>
</tr>
<tr>
<td>Apache HTTP server</td>
<td>2.4.7</td>
</tr>
</tbody>
</table>

Table 4.2: All dependencies used in the front-end application.

4.4 Ranking algorithm

A Dueling Bandit Gradient Descent algorithm, with a Team-Draft comparison method, is implemented in the Resume Database. The algorithm is developed in Java, built without any external Machine Learning libraries or frameworks. However, the algorithms and models implemented, in the ranking module, are well-known in the Online Learning to Rank area. Initial weights and values for the exploratory and exploitative step size parameters can be set in the configuration file `ranking-settings.cfg.xml`. The exploitative weight vector is persisted and backed up in the file `ranking-weight-vector-backup` as binary data. This is done every hour as a precaution. By removing the ranking-weight-vector-backup file, the ranking algorithm is reset and will start running with the initial weight vector set in `ranking-settings.cfg.xml`.

In figure 4.8, one complete iteration of how the ranking algorithm is integrated and used, in the system, is illustrated. Note that clicked consultants, which resume’s are not selected for generation, are not registered to learn the algorithm. Usually all clicks made in the interleaved result list is registered as feedback to the algorithm, but this is not the case in the Resume Database. Only the chosen resumes for PDF generation are used as implicit feedback, to learn the algorithm, at this point.

The ranking algorithm is implemented with the goal to be as smoothly integrated with the rest of the system as possible. All data handled by the algorithm is stored in the backend, and hidden as much as possible from the front-end. There is no extra communication or data needed in the messages between the front-end and back-end, except the user id of the sending client, when a search query is issued, and the user id of the clicked consultant, when a resume is selected to be generated as PDF. The user id of the searching user is used as a key when storing data during the ranking phase, and when loading the correct data, when the ranking algorithm is later updated. This data is information about which resumes that were assigned to, and contributed from, the exploratory list, and the sampled unit vector used when generating the exploratory weight vector.

Not all features listed in section 2.5.1 are implemented in the ranking algorithm of the Resume Database. This is mainly because the data indexed by Hibernate Search is the only
Figure 4.8: A flowchart showing how Online Learning to Rank is used in the Resume Database.

data regarding the resumes and consultants that is available for the ranking algorithm. The reason why the indexed data is the only data used in the ranking algorithm, is because it significantly improves the performance of the system. Initial experiments showed that sets of resumes larger than 200 could take several seconds to rank, when data, to be used by the ranking algorithm, was fetched from the database. This is further discussed in section 6.1.

The results of the evaluation of the ranking algorithm is presented in chapter 5.
Chapter 5

Evaluation

This chapter presents the results from the evaluation of the Online Learning to Rank algorithm.

5.1 Method summary

Below follows a summary of the evaluation process of the ranking algorithm. In section 2.5.2 the evaluation methodology is described more in-depth.

Andreas Hed and Urban Holmgren at Knowit Norrland, who represents expert users of the Resume Database, assisted in the evaluation of the ranking algorithm by ranking a set of resumes for a set of search queries. These resumes were generated by selecting random features from sets of skills, professions, projects and earlier employments. Personal details such as name, sex and age were ignored in the resumes, since these are not part of the features used in the implemented ranking algorithm. The search queries were created with varying number of query terms, consisting of skills, professions, company names and other keywords. Both the resumes and search queries for the evaluation were generated to capture a large range of realistic scenarios. The expert users task for the evaluation were to label all resumes, for each search query, based on a relevance scale. The relevance scale consisted of grades between 1-5, where 1 represented a completely irrelevant resume and 5 a perfect matching resume. Permutations of the resumes based on the grades labelled by the expert users were constructed for each query, and these rankings were later considered the ground truth tables, during the rest of the evaluation.

It was concluded in an early stage of the evaluation phase, that the results could be greatly affected by the time allocated to gather and construct training and test data. This was because the time proved to be too short and therefore it resulted in very limited data for the evaluation. The problem has however, been taken into account during the discussion and presentation of the results later in this section. One reason why the time was not sufficient, was that the learning phase in the evaluation was beforehand decided to only be performed by real users - hence no graded training data would have been needed. Unfortunately this plan had to change, since the time that the intended users could spare, to learn the algorithm for the evaluation, seemed insufficient, to guarantee that the learning would turn out adequate. The algorithm was instead decided to solely be learned by click models, which on the other hand needs graded training data to work efficiently, but this problem was considered easier to get around.

The algorithm was learned by three different click models; one that simulates a perfect
clicking user, one that simulates a realistic clicking user and one that simulates an unreliable clicking user. These are further explained in section 2.5.2. For each click model an experiment consisting of 1000 clicks were performed, and during the experiment an evaluation with the testing data was done every 10th click. This was repeated 5 times and an average was computed for each click model. During the experiments the exploration and exploitation step size parameters were varied to find out the best performing values on a validation set. The experiments showed that an exploration step size of 0.1 and exploitation step size of 0.01 performed well. Joachims and Yue [37] confirms this, when they show that 1 and 0.01 are the best performing parameters, but suggest that smaller values are better when sampling fewer queries, as the case in this evaluation.

5.2 Result

The results presented here are not advised to be taken completely for granted. Do not place too much importance on the exact figures and numbers presented, instead see the result as indications and pointers of the performance of the implemented ranking algorithm. The initial results presented here might indicate that the algorithm has been fallen victim to overfitting, the case where the model has learned only how to predict the training data and not unseen data (the test data), which is the actual goal. If not made clear before, the training and test data that was used in the evaluation are different from each other, and does not contain the same resumes or search queries, which is seen as a necessary requirement. To confirm that only minimal overfitting might have taken place, experiments were made where it was shown that the performance of the test set closely followed the training set performance. This was followed up by gathering additional data for a second test set which yielded similar performance as the first test data, that is used in the figures and tables below. These arguments can however, not be used to completely conclude that overfitting has not been a factor during the learning of the ranking algorithm, for this evaluation. The point here is simply to show that this problem has been taken into account during the evaluation. It is strongly recommended to perform new evaluations, with larger training and test sets, before considering to use the implemented algorithm in the Resume Database in a real setting.

In figure 5.1, the result when running the algorithm five times with 1000 iterations without any learning is presented. This result is presented to demonstrate the initial performance scores of the algorithm. The experiment shows that an unlearned algorithm can produce rankings with varying NDCG scores. This is because the list of resumes is shuffled at each iteration, and therefore it can produce even perfect rankings initially, without any learning. However, as can be seen in the graph, the average rankings in this experiment varied in performance roughly between 0.38 and 0.67.

In table 5.1 the resulting NDCG scores, of the ranking on the top 4 resumes (NDCG@4), for the algorithm, learned with a perfect, realistic and almost random click model, can be seen. As expected the almost random click model performs the worst rankings and the perfect click model performs the best rankings. Slightly surprising is the fact that the realistic click model’s final performance is almost on par with the perfect model with only a small difference in final scores. The performance improvements presented in table 5.1 answers Research Question 4 in section 1.2, how good the performance of the ranking algorithm is after being learned. Even with the almost random click model the algorithm is improving over time with a 40% improvement. This is however, as expected, far from

\[1\] A separate set, of graded resumes and queries, that is not part of the training or test data.
5.2. Result

The improvements when learned with the remaining click models, which landed at a 72% improvement for the realistic and 79% improvement for the perfect click model.

<table>
<thead>
<tr>
<th>Click model</th>
<th>Final score</th>
<th>Performance improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect</td>
<td>0.88</td>
<td>79%</td>
</tr>
<tr>
<td>Realistic</td>
<td>0.85</td>
<td>72%</td>
</tr>
<tr>
<td>Almost random</td>
<td>0.72</td>
<td>40%</td>
</tr>
</tbody>
</table>

Table 5.1: The final average NDCG scores and performance improvements for each click model after running an experiment consisting of 1000 iterations, five times.

In figure 5.2 the learning curves for the experiments where the performance was calculated with NDCG@4, are plotted. The illustration clearly shows that the ranking algorithm is learning a better ranking via the implicit feedback over time. It is concluded that the perfect click model is the fastest at learning the algorithm, while the almost random click model takes the longest time to learn the algorithm. Both the realistic and almost random click models are naturally more unreliable, because of the increased noise in the feedback, than the perfect click model, which is visible in the figure. Especially the almost random model is learning the algorithm in a very unstable way with a lot of noise that results in a widely varying curve between iterations, which makes it jump between high and low performance scores, during the entire learning phase. This is very much expected because of the added noise and the small decay in click probabilities for this model. However, even with this significant noise it is still obvious that the algorithm does learn, when the feedback is unreliable.

A final experiment was performed with the realistic click model learning the algorithm during 2000 iterations and 25 runs, to plot a smooth realistic\(^2\) learning curve for the implemented ranking algorithm. In this experiment the NDCG@8 was calculated to find out the

\(^2\)The small size of the training and test sets must be carefully taken into account here, especially since the realistic click model is not proven to actually simulate a realistic human behaviour, but is only reasonably constructed on known facts about real users of the system.
Figure 5.2: Graph illustrating the learning curves for all three click models when running 1000 iterations, five times.

ranking performance of the top 8 resumes in the ranked list. The result of the experiment is presented in figure 5.3. Here the learning is shown to steadily improve up until iteration 300-350. At iteration 2000 the performance score averaged to 0.83. After the 300th iteration the performance is pending between a score of 0.80 and 0.84. This result suggests that an upper limit is met, and it seems like the algorithm can not learn a better ranking than this, with the settings and data used in the experiment.

Figure 5.3: A smooth plot over the learning curve when learning the algorithm with the realistic click model and running 2000 iterations, 25 times.

The last Research Question (5) posed in section 1.2 is not trivial to answer. It is quite safe to say that a static ranking algorithm would probably be able to construct more reliable and better rankings, if adjusted good enough, than the algorithm implemented here. However, it is the complexity of knowing or determining these adjustments that is the key challenge.
If you already know how to produce good rankings for a scenario and know that this will not change over time, then a static ranking algorithm should preferably be a better choice. But, if this ranking is unknown, Machine Learning can contribute with a major advantage to this ranking problem, which also reflects the preferences of the users of the system.

To summarize this discussion: If the Resume Database would operate in a real setting at Knowit Norrland today even a simple static ranking function would probably be efficient enough for this purpose. However, if this system would be used in a larger setting, let's say for the whole Knowit concern with almost 2000 employees, then the answer might be different. If the Online Learning to Rank algorithm implemented in the Resume Database would prove, in a larger evaluation, to be as good as the results in this evaluation suggest, it sure do stand as a serious contender, at least.
Chapter 6

Conclusions

The goal of this thesis has been to implement a resume database with an Online Learning to Rank algorithm and evaluate the performance. Results from the implementation and evaluation have been presented and discussed in this paper, hence all general goals addressed in section 1.3 are therefore considered reached. The requirements that were initially set on the Resume Database together with Urban Holmgren and Andreas Hed at Knowit Norrland are also considered met by the involved parties.

The result from the evaluation of the ranking algorithm indicated that performance in a simulated environment proved to be sufficient, and that the algorithm could be desirable in a real setting for the Resume Database in the future. It is however concluded that the algorithm is not suitable to be used, in practice, in its current state. More experiments and a larger evaluation is necessary before this can be suggested.

6.1 Restrictions

Originally the evaluation of the ranking algorithm was planned to utilize the expert users of the Resume Database more than how it turned out. It has been discussed already in this paper, but can not be addressed enough, that a larger evaluation with more training and test data would have been the highest priority if more time was available for this thesis work. An evaluation where actual users learn the algorithm, instead of simulations with click models, would have been interesting both from a personal and scientific point of view, since no earlier work on this has been found during this thesis.

Regarding the implementation of the Resume Database, the biggest regret was to not apply a test-driven development. This was discussed initially and even suggested by people at Knowit Norrland, but was opted out due to the time limit set for the thesis work. The system is however implemented with this in mind, and adding testability at this time should not need to be a major task. Another regret, regarding the implementation of the Resume Database, is the choice of using a relational database. The decision behind this is discussed in section 4.2.1, and the reasons still holds, but it would have been more interesting to use and evaluate a graph database for this system.
6.2 Limitations

The decision on using only the indexed data by Hibernate Search in the ranking module, resulted in a limited set of features implemented for the computations in the Online Learning to Rank algorithm. To use all features listed in section 2.5.1, more data about the resumes and consultants, than what can be extracted from the indexes, is needed.

If a user issues a search query twice, directly followed by each other, to the system today, the search result will most likely differ. This is because the Online Learning to Rank algorithm injects exploratory resumes with the Team-Draft method, and the exploratory resumes are selected by a randomly generated unit vector each time. To overcome this problem, which mainly is considered to affect the user experience, the search results are suggested to be cached.

Knowit Norrland requested a functionality to display a relevance score for each resume in the search results. This was looked into, but unfortunately no solution was found during the thesis work. The scores that are computed for each resume and used internally by the ranking algorithm might seem like an obvious candidate at first. However, since a result list is always interleaved by two rankings, one exploitative and one exploratory, the scores can not be used together.

6.3 Future work

The improvements possible for the Resume Database and, in particular, the implemented ranking algorithm are many. However, the improvements needed to use the Resume Database, without the Online Learning to Rank algorithm, in a production environment, are not considered major.

One important improvement is to handle errors in the back-end system and, more importantly, in the front-end application better than it is done today. Restlet automatically handles most common errors and returns these as HTTP error messages (such as code 500, 404, 401, etc.) to the client. There are also a few customized error messages created and returned when specific errors regarding the Resume Database occurs in the back-end. The error management in the front-end is however, almost completely disregarded, except major errors, such as authentication or server errors.

Except the authentication vulnerabilities already discussed in section 4.2.2 and 4.3.1, the expiration time set on the authentication cookie should be pro-longed as long as a client is active. Today, the expiration time is static and will not update for an already created cookie. This forces the client to re-authenticate after the expiration time has run out, even if the client has been active during the whole session. If the expiration time instead is pro-longed, as long as the client is active, the client will only be needed to re-authenticate if being inactive for the specified duration.

Other identified improvements for both the front-end application and back-end system are:

- Even if the website is implemented with a responsive design, the "edit and preview"-page and search page need improvements, if mobile usage is a priority.

- If a user returns to the search page, where a search has been issued before, by using the navigational buttons, the search result should still be available. Adding the search query as a parameter in the URL is a suggestion to implement this behaviour.
6.3. Future work

- Some styling is hard-coded on the website. This code should be refactored into classes and put in the CSS stylesheet.

- Pagination on the search page. This is partly implemented in the back-end system, when fetching all users in the system, but the search engine would need a similar implementation as well.

- The resumes could be a lot more detailed. As examples, the project description could be separated into smaller components and more information, such as a consultant’s cost could be added.

- A backup interval should be set-up for the database. Today no backups are done automatically.

A transition to a graph database, such as Neo4j, from the MySQL database currently in-use in the Resume Database, is recommended as a further development. The MySQL database used today with Hibernate Search is more than enough for the intended usage of the system. However, a graph database is considered a much better fit, especially in the sense of performance, for the matching of consultants than a relational database. Unfortunately, since Hibernate has not been found to support NoSQL-like databases such as Neo4j, the database API, used in the Resume Database, can not be used for this purpose. This implies that if a transition to a graph database is done, the database API must either be modified or a new separate API must be implemented. If more time would have been available for this thesis work, the possible usage of both a graph database and a relational database would also be looked into. This way, the graph database can hold all data about resumes that is needed for the matching in the search engine, and the relational database all other data used in the system.

A further development that could be beneficial for the Resume Database, but is far from a requirement, would be to integrate LinkedIn’s API, to fetch information from a user’s personal profile. This idea is influenced by the work done in [6] and would probably be of great assistance to the users of the Resume Database, since a LinkedIn profile often is updated with resume information.

The Apache Lucene functionality in Hibernate Search could be utilized far more, than it is in the Resume Database today. An interesting area to investigate would be the possible integration of the Online Learning to Rank algorithm in the Lucene scoring algorithm. This is certainly something that should be looked into if the Resume Database is further developed. With more time this unexplored area could prove to be a much more efficient way to implement the Learning to Rank algorithm than it is now.
References


Appendix A

Algorithms used in the Learning to Rank implementation

The algorithms that are implemented in the Resume Database which builds off the Dueling Bandit Gradient Descent algorithm with a Team-Draft method are presented here. Note that both algorithms are modified for hopefully easier understandings but are similar summaries of how the algorithms originally were illustrated.

A.1 Dueling Bandit Gradient Descent

Algorithm 1 The DBGD algorithm implemented in the Resume Database

1: **Input:** $\gamma, \delta, w_0$
2: $w_t \leftarrow w_0$ //Set initial weights.
3: **for** query $q_t (t = 1..T)$ **do**
4: $R \leftarrow$ All resumes relevant to $q_t$.
5: Sample unit vector $u_t$ uniformly.
6: $w'_t \leftarrow w_t + \delta u_t$ //Generate exploratory weight vector
7: **if** $TD(R, w_t, w'_t)$ **then** //Team-Draft interleave compare method
8: $w_{t+1} \leftarrow w_t + \gamma u_t$
9: **else**
10: $w_{t+1} \leftarrow w_t$

**Clarification:**

- $\gamma$ = Exploitation step size.
- $\delta$ = Exploration step size.
A.2 Team-Draft method

Algorithm 2 The Team Draft interleave comparison method algorithm : $TD$

1: **Input:** $R, w_1, w_2$
2: Generate ranked list $l_1$ from $R$ with $w_1$.
3: Generate ranked list $l_2$ from $R$ with $w_2$.
4: **for** rank $r (1..r_{max})$ **do**
5: \hspace{1em} $k \leftarrow$ Random double between 0 and 1.
6: \hspace{1em} **if** count($l_1$)$<$count($l_2$) $\lor$ k$>$0.5 **then**
7: \hspace{2em} $L[r] \leftarrow$ Top-ranked resume from $l_1$ not already in $L$.
8: \hspace{1em} **else**
9: \hspace{2em} $L[r] \leftarrow$ Top-ranked resume from $l_2$ not already in $L$.
10: **Display** $L$ and observe clicked resume $c$.
11: **if** $c \in l_1$ **then**
12: \hspace{1em} **return** true
13: **else**
14: \hspace{1em} **return** false

Clarification:
- $R =$ All relevant resumes based on the search query.
Appendix B

Screenshots

Figure B.1: A screenshot of a profile page.
Figure B.2: A screenshot of the edit pages.
Figure B.3: A screenshot of the "edit and preview"-page for a resume, where *preview mode* is enabled.
Figure B.4: A screenshot of the “edit and preview”-page for a resume, where edit mode is enabled.
Figure B.5: A screenshot of the “edit and preview”-page for a resume, where edit mode is enabled.