LEARNING TO RANK CANDIDATES FOR JOB OFFERS USING FIELD RELEVANCE MODELS

Author: Min Fang

Supervisor: Malvina Nissim

Co-Supervisor: Dietrich Klakow

Master of Arts
Department of Linguistics
Faculty of Arts
University of Groningen

Master of Science
Department of Comp. Linguistics
Faculty of Arts
Saarland University

August 2015

ABSTRACT

We leverage a dataset of Dutch job postings, applicants' CVs and the corresponding hiring decisions made by real recruiters to improve the ranking of candidates with respect to search queries representative of real-world job offers. For this purpose we first propose a field relevance model of CVs, which can represent implicit domain knowledge available in a large collection of CVs without or with minimal supervision. We show in a query expansion experiment that such a model alone can improve the recall of the retrieval of candidates. In a second step we learn a (re-)ranking model which operates on the initial ranking of a search engine. We demonstrate that this model can be obtained through a learning-to-rank framework, for which we define a number of features, including features which can be computed based on the field relevance model. Hiring decisions, i. e. whether a candidate was hired, interviewed or rejected, can be used as relevance labels in the learning scheme. In our experiments we show that this supervised training procedure is able to produce a reranking model that improves significantly over the search ranking on common information retrieval performance metrics.

DECLARATION

I hereby confirm that the thesis presented here is my own work, with all assistance acknowledged.

Groningen, August 2015

Min Fang

ACKNOWLEDGMENTS

I wish to thank my supervisors Malvina Nissim and Dietrich Klakow for their guidance and the interest they have taken in this small piece of research. I am especially grateful to Malvina, who was willing to carry out remote supervision and was able to pep-talk me whenever I was in doubt.

This work would not have been possible without the people at Textkernel B.V., who gave me a great chance to work with real-world data on a real-world problem. I would like to thank Remko for the unique opportunity and Agnes and Henning for all the support and fruitful discussions. I will always remember the Friday borrels.

Last but not least, my everlasting gratitude go out to my parents, who have always supported and encouraged me to follow my own path.

CONTENTS

1 INTRODUCTION

Finding the right person for the right job has never been an easy feat for companies, whose value is very often to a large degree derived from their manpower. With the increased mobility of job seekers in recent years, more and more jobs are seeing rapidly growing pools of potential candidates, requiring respective recruiters to wade through hundreds if not thousands of CVs to find the perfect match.

Set against this backdrop, recruitment software is aimed at automating repetitive routine tasks in the screening process while leaving the recruiter to make the important decisions (e. g. which candidates should be invited to an interview). When there is a large number of candidates, the automation can also include a more challenging task: scoring and ranking candidates' CVs according to their match to a job posting (represented concretely as a query). This task is usually implicitly performed by a search engine's retrieval model, which computes ranking scores based on the text overlap of the CVs with the query keywords. While text similarity might reflect good matches for some jobs, it is easy to find examples where this naive approach fails (e. g. a *warehouse manager* should not appear on top of a list for *assistant warehouse manager*). Improvement could be obtained through data-driven approaches, however, in the domain of recruitment it is very costly to compile and annotate enough data so that supervised learning would be possible.

This thesis takes on a data-driven approach but avoids the hand-labelling of training data by using a different kind of information: We use a dataset with recruiters' hiring decisions to learn a ranking model, which can improve on the initial search ranking. For this purpose we first introduce *field relevance models* for CVs, which are generally unsupervised models that can take advantage of implicit domain knowledge in a large collection of CVs. We demonstrate in an experiment that such a model has the potential to improve the recall of retrieval when applied in a simple query expansion set-up.

To improve the ranking of CVs we take on a feature-oriented, learning-to-rank approach (LTR), i. e. we propose a number of features for our problem domain, forming the basis for a machine learning algorithm whose resulting model is a ranking model. In particular, we also propose features which can be computed based on the field relevance models. The relevance labels necessary for this supervised learning scheme are supplied by real-world hiring decisions, i. e. information about which candidates were hired/interviewed/rejected for a collection of jobs. We show experimentally that this LTR set-up can indeed lead to a ranking model that improves over the search baseline in terms of common information retrieval performance metrics. Our evaluation on a hand-labelled set of vacancies with associated CV pools shows that the learned ranking model has some generalisation ability, while we also point out ideas for potential future improvements.

The main contributions of this thesis can be summarised as follows:

- We propose an unsupervised model that can take advantage of implicit knowledge in the domain of recruitment and show how it can improve the recall in a retrieval setting.
- We adopt a feature-oriented view of retrieval and describe a number of features that can be used in a learning-to-rank framework for the ranking of CVs. In particular, we show how the unsupervised models can be used to create meaningful features.

- We conduct a number LTR experiments with hiring decisions as relevance labels and show that the ranking of candidates can be significantly improved compared to the baseline, i. e. the original search ranking.

2 BACKGROUND & MOTIVATION

2.1 RECRUITMENT TECHNOLOGY

Hiring the right person for the right job is a common challenge faced by all companies. Especially for positions with a large number of applicants the search for the right candidate(s) can feel like looking for a needle in the haystack. In these situations traditional methods of recruitment can be too expensive and time-consuming to be a viable option. Hence, not surprisingly, recruitment technology that can facilitate this process are in high demand. E. g. using a (searchable) database of candidates and a search engine a recruiter can preselect a small number of suitable candidates from a much larger pool so as to assess them in further recruitment procedure. It should be noted that the goal of such software *is not* to replace the "human" in human resources but to make the decision process smoother for the recruiter.

For this purpose an increasing number of software packages provide means for executing recurring tasks automatically. Both CVs and job postings can be automatically parsed and relevant information is extracted and stored in databases. Easy-to-use interfaces are provided for maintaining the quality of extracted information (e. g. for manual corrections) and for keeping track of typical HR processes involving vacancies, candidates and interviews (so-called applicant tracking systems or ATS). With the growing importance of social media, more and more companies nowadays also offer "social recruiting" capabilities, which can tap into an even larger, more global pool of qualified candidates through social media platforms such as LinkedIn and

Xing. Thus, in order to take full advantage of the bigger candidate pools, it is crucial to apply smart search and ranking strategies such that good candidates are indeed placed on top and do not disappear in the crowd.

2.2 CURRENT SYSTEM AND IMPROVEMENT IDEA

The goal of this thesis is to extend the search and ranking component of an existing commercial recruitment software package. In particular, we target the ranking component of a CV search engine, which is responsible for scoring and ranking candidates' CVs for queries (either issued by users or automatically translated from job postings).

This existing software offers some basic functionalities, which form the foundation of many common recruitment processes:

- The automatic CV parsing extracts relevant information such as name, address, skills and previous work experience from original CVs (e. g. given as PDF or DOC documents) and transforms them into a searchable semi-structured format.
- The search engine indexes parsed CVs and enables searching with semi-structured queries as well as through search facets and tag clouds. CVs are assigned a relevance score w.r.t. the query by the search engine and are ranked accordingly.
- Automatic vacancy parsing extracts relevant information from vacancies such as the title of the advertised position, skill requirements and other job-opening-related keywords.
- The query generation component automatically generates semi-structured search queries for finding matching candidates in the document collection.

The CV and vacancy parsing models are machine-learned models, which are trained to detect relevant phrases and sections in CVs and vacancies and can infer what kind of information the given phrase represents. Knowing the "meaning" of relevant parts of a CV allows more sophisticated search and filtering options, e.g. by searching only in the skills section or filtering candidates by their years of experience.

The workflow that we are mainly interested in involves the query generation component, which uses the information obtained from the vacancy parsing model and generates a query according to a predefined template. This kind of query is generally longer than user-defined queries and contains a lot more information. An example query is given in Listing 1, which contains both terms that should match in specific fields and terms which can match anywhere in the CV (so-called *fulltext* terms).

Listing 1: An example query generated based on a real Dutch vacancy.

```
%jobtitlesonlyrecent:[medewerker financiele administratie] %
    jobcodesonlyrecent:"2028"  %jobclassidsonlyrecent:"1"  %
    jobgroupidsonlyrecent:"1"  %city:"Schiedam"+50   %educationlevel:(2
    3) %langskills:NL  %compskills:"Word"   %compskills:"Excel"  %
    compskills:"Outlook"  %compskills:"Powerpoint"      %experienceyears
    :(1..2 3..5)  %"Word/Excel" %"Klantgerichtheid" %"Financiele
    Administratie" %"crediteurenadministratie"   %jfsector:"5"
```

Thus, these queries provide a good basis for finding candidates that match the original job posting well with very little human effort. Based on the search terms in the generated query, the search engine in our current system computes a score for each candidate's CV, according to which a final ranking is created.

The focus of our work is to extend this current ranking system by learning a model that can re-rank an already ranked list of candidates according to some notion of suitability for a given query. Concretely, the initial ranking is performed by the search engine's TFIDF-based retrieval model. Our "re-ranking" model should manipulate the ranking of this preselected list to ensure that the best candidates are placed on top, as sketched in Figure 1. Furthermore, we want to gain some understanding of the aspects that play a role in the learning and the ranking procedure and how they relate to possible notions of suitability/relevance.

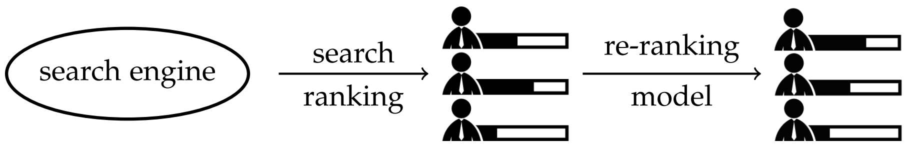

Figure 1: Re-ranking of the search ranking list.

This task is challenging because we face a logical (i. e. given the already existing set-up) but highly competitive baseline provided by the search ranking. Previous user feedback suggests that the retrieval model used by the search engine already captures some notion of relevance that has a correspondence to the suitability of candidates.

The approach that this work follows to tackle this challenge is one of machine learning, i. e. we want to learn a model from data without having to craft a ranking function or ranking rules explicitly. This approach requires a suitable dataset from which a ranking model can be learned. The details of the datasets that are used in our project are given in the next section.

2.3 LEARNING FROM DATA

The practical motivation for our learning approach is the availability of a relatively large dataset which contains real-world job ads, original CVs of the applicants for these jobs as well as information about which candidates were hired in the end of the recruitment process. We will refer to this dataset as *the hiring decisions*. Another, much smaller dataset contains human relevance judgements for a number of job ads and CVs of people who did not necessarily apply for the given job (in this thesis referred to as *the relevance assessments*). A short overview table of the two datasets is given at the end of this section in Table 1. Using the bigger dataset of the two, i. e. the hiring decisions, we will apply a learning-to-rank approach (in short: LTR) to learn a re-ranking model, where the actual hiring decisions made by recruiters serve as relevance labels. Different learning-to-rank strategies are discussed in Section 3.2.

2.3.1 *The Hiring Decisions*

The hiring decision set originates from a Dutch recruitment agency and contains overall approximately 9K Dutch vacancies, 300K CVs of applicants and information about which candidates were hired for the corresponding vacancy. In order to use this raw dataset in our setting we had run the data through a few steps of preprocessing (illustrated in Figure 2):

- vacancies
  - parse the vacancies with Dutch vacancy parsing model
  - automatically generate semi-structured search queries based on the output of parsing
- CVs
  - parse the CVs with Dutch CV parsing model
  - index the parsed CVs in the search engine and associate them with their corresponding vacancy (query)

The result of the preprocessing is a set with approximately 9K queries as proxy for the corresponding vacancies, 300K searchable

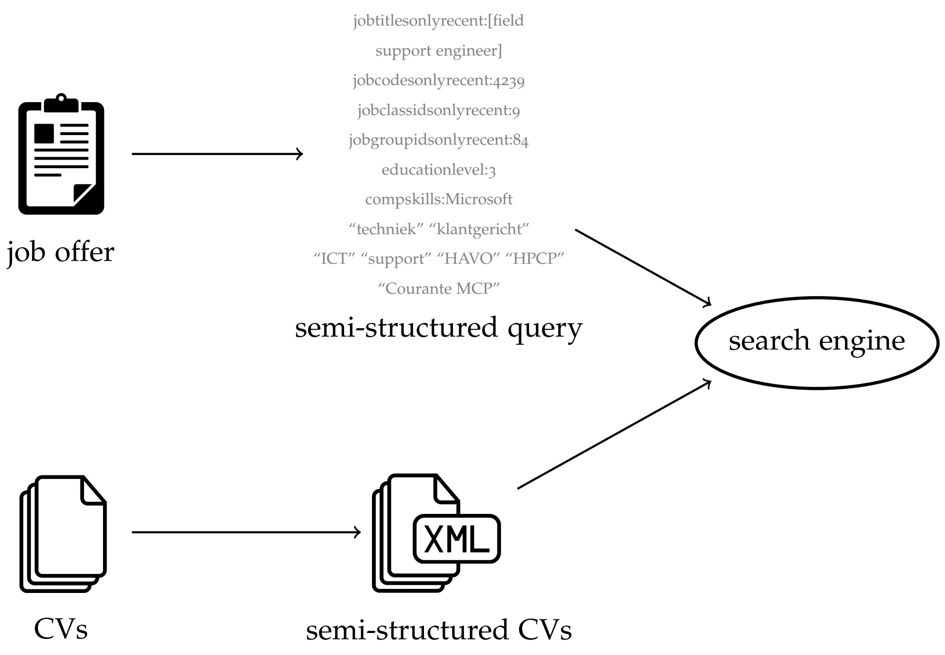

Figure 2: A simplified illustration of the preprocessing steps.

CVs whose predefined fields are indexed in the search engine. This set-up will allow us to issue a query (including its unique id) to the search engine, which will then retrieve the CVs of those candidates who applied for the corresponding vacancy according to the original raw dataset. The ranking of the retrieved CVs is computed based on the search engine's internal ranking model,[1] which provides us with a natural starting point for the re-ranking.

The actual hiring decisions, i. e. the information whether a candidate was hired or not, are processed from a large number of Excel speadsheets and transformed into a simple *hired* vs. *rejected* label. However, the spreadsheets also contain information about meetings with the candidates (e. g. dates of meetings), which we interpret as interviews with the candidates. Hence, there is a number of candidates who did not get hired according to the hiring information, yet they seem to have been invited to an interview instead of getting rejected

1 In our case the model used is the default model in ElasticSearch, which is based on the TFIDF and Vector Space Model. Details are given under https://www.elastic.co/guide/en/elasticsearch/guide/current/practical-scoring-function.html.

straightaway. Even though this information is less explicit in the raw data, we consider it an interesting signal that is worth experimenting with. Details about how we use this optional label to generate different versions of the dataset are given in Section 5.1.

Despite the size of the dataset, which makes it predetermined for learning, it has to be noted that it is a considerably heterogeneous set. The heterogeneity is to a certain degree inherent to the problem domain since both vacancies and CVs are produced by different people with different templates/strategies in their minds, given that there are no universal rules of how to compile a CV or a job offer concretely. Fitting the CVs and the vacancies into our structural template (i. e. with our predefined set of fields such as job title, skills, industry etc.) is bound to result in missing values and values that cannot be unequivocally assigned to a specific field. However, this kind of normalisation is necessary to make different CVs and different vacancies at least somewhat comparable and it allows more focused search queries via semi-structured search.

At the same time the dataset is also heterogeneous in the sense that it contains data from different companies with different, possibly unknown, demands for their candidates and policies for their recruiters. Hence, it is possible that for two jobs with very similar requirements, two different companies might hire two candidates with a very dissimilar profile. Conversely, we may have two candidates with very similar skills and experiences applying for similar jobs; however, one gets hired and one get rejected due to some hidden variables (e. g. contextual information not available to the dataset). Hence, we have to expect the learning from such a hiring decision set to be noisy and more difficult than e. g. learning from an equivalent set with relevance judgements given by one person.

Another property of the preprocessed dataset is the fact that the preprocessing itself might introduce noise and missing values: Both parsing models are based on machine learning techniques and are naturally not 100% accurate. Information that is in the original raw data and (only) human-readable may get lost or classified wrongly when the parsing models fail to extract it properly. Similarly, the automatic query generation process is based on predefined, heuristic rules and may, thus, miss crucial information for some vacancies. However, the parsing errors are unavoidable if the raw data is to be made machine-readable and searchable. Optimising the query generation process (e. g. deciding what information should be included in the query, which weight to assign to the terms etc.) is possible but reserved for future research.

In summary, the hiring decision set has several flaws that will make learning from it more difficult. However, because of its considerable size we expect it to overcome those flaws at least partially.

2.3.2 *The Relevance Assessments*

In order to model the notion of suitability on a broader level we compiled a dataset of 99 vacancies (basically randomly selected from the vacancies in the hiring decision set[2]) and associated each of them with a set of at least 100 CVs that are gathered through pooling.[3] For this set we collected exhaustive relevance judgements given by three students, who were asked to assess the CVs according to their match to the given vacancy with one of the following labels: *not-relevant*, *somewhat-relevant*, *would-interview*, *would-hire*, *overqualified*. These students were required to participate in a "tutorial session" led by a pro-

2 We tried to ensure the breadth of companies included in the set.

3 We used the top-ranking documents from five different systems to fill the pools: the hiring decisions (where documents are sorted according to the labels *hired* > *interviewed* > *rejected*), automatically generated queries issued to a search engine based on Indri and based on ElasticSearch, manually created queries issued to the Indri engine and to the ElasticSearch engine.

fessional recruiter, who gave them some insight into the recruitment procedure.

It is evident that this dataset only provides a rough approximation of candidate suitability and because of its modest size it cannot (straightforwardly) be used as a training set for learning. However, it still has some advantages over the hiring decisions: As opposed to the vacancies in the hiring decisions, all vacancies in this set have a pool depth of at least 100 and also contain CVs from people who did not apply for the given vacancy. The used pooling technique ensures broader coverage of relevant CVs and, thus, can lead to a more meaningful evaluation set for certain settings.[4]

Our usage of this dataset is twofold:

- We use this dataset as an evaluation set for our recall experiment (Section 4.4) because of its bigger and broader pool of relevant documents.
- We evaluate our model trained on the hiring decisions also on this set in order to gain some insight in the correspondence (or discrepancy) of the notion of relevance in these two different datasets (Section 5.3).

2.4 MODELLING RELEVANCE

Deciding what relevance means in the domain of recruitment is a non-trivial task. Similar to document retrieval the notion of relevance in a recruitment system is strongly tied to the user intent, in particular, to the recruiter's assessment of the job ad and what kind of candidates she deems suitable for the ad. However, different from

4 Testing on this set may be especially interesting for the use case where a recruiter wants to search through an existing pool of internal employees to find suitable candidates within the company/database before considering external applications.

Table 1: An overview of the two available datasets.

hiring decisions	**relevance assessments**
9K vacancies & queries	99 vacancies
300K parsed CVs	9900 parsed CVs
hired, rejected, (interviewed)	not-relevant, somewhat-relevant, would-interview, would-hire, overqualified
heterogeneous, missing values, noisy	deeper pools

document retrieval, in recruitment it also matters whether the candidate associated with a retrieved CV would consider the advertised job suitable as their next career step, making the system what is sometimes referred to as a *match-making system* (Diaz et al., 2010). In other words, it is not only the preferences of the recruiter that matters but also the preferences of the candidate. In Mehta et al. (2013) these aspects are explicitly modelled as separate dimensions, for instance, they take into account the probability of the candidate accepting the job offered to them as well as the probability of the candidate remaining with the organisation for long term, for both of which they have explicit historical data to learn from.

Another important aspect of relevance is its subjectivity: Whether or not a document is considered relevant to a user intent may vary from user to user. This fundamental problem also exists in the recruitment domain: For the same job offer the same candidate might be deemed appropriate by one recruiter but not by another. These judgements of relevance are highly dependent on the company and their hiring policies as well as the preferences of the individual recruiters. Given the subjectivity in the hiring process, some real-word applications aimed at facilitating this process propose methods based on personalised search (e. g. Malinowski et al. (2006)).

Using hiring decisions as a proxy for relevance necessarily neglects certain aspects of relevance: As a simple label (*hired* vs. *not-hired*) it can mask processes that could have taken place between the initial inspection of CVs and the final hiring decision such as interviews (this is for instance different from online labour markets, cf. Kokkodis et al. (2015)). E. g., two candidates with similar qualifications and work experience may be barely distinguishable based on their CV, however, only one of them may excel at an one-on-one interview and is then hired thereafter. Does this fact make the other candidate non-relevant for our purposes? No, from the ranking system's perspective both candidates are equally suitable and relevant. Thus, the process of hiring is usually extremely selective compared to the number of applicants and usually artificially reduces the number of relevant candidates to a very small number. In other words, we could consider pure hiring decisions as very high-precision labels, yet with low recall in the pool of relevant candidates. We will try to lessen the impact of this flaw by introducing an heuristic *interviewed* label,[5] which also makes those candidates relevant who were not hired but at least got to the stage of a personal meeting with the recruiter.

In our dataset being hired also means that the candidate accepted the job, hence, indicating also some information about the attractiveness of the job for the given candidate as a career move. However, what we cannot expect from these labels are explicit information about subjectivity since we do not have the information about individual recruiters. We expect this aspect to be interesting research for the

5 Heuristic because the dataset does not contain this information explicitly as is the case with the hiring information, but it is only represented e. g. sometimes as the reason for rejection or there is a date for a meeting etc.

future, which might involve personalised search using for instance clickthrough log data of users and online learning-to-rank strategies.

2.5 EVALUATION OF THE RANKING

Since we cast our matching/ranking problem as an information retrieval task we can use common IR metrics such as NDCG and MAP as a measure of performance (cf. e. g. Manning et al. (2008)). For this purpose a collection of test topics (representable as queries) is needed as well as labelled documents associated with the topics. In many retrieval tasks the documents are collected through the process of pooling (i. e. a combination of the top-ranked documents from a number of different retrieval systems) and the labels are human relevance judgements that are given w.r.t. the topics (*not* the queries) and can be considered a gold standard (sometimes called the *ground truth*). The labels can be binary (*relevant* vs. *non-relevant*) or more fine-grained depending on the context. The collection of our assessment set follows this procedure as described in Section 2.3.2 as its purpose is to serve as an evaluation set.

In the domain of recruitment it may not be immediately clear how the notion of ground truth can be applied and what it means for a document to be relevant for a given topic, i. e. for a CV to be relevant for a given job posting. As discussed in Section 2.4 using hiring decisions as relevance labels has its flaws and can only be considered one aspect of relevance/suitability. Nevertheless, we will evaluate our models on a subset of vacancies taken from the hiring decision set in order to measure whether and how well our trained model is able to put those candidates on top who were indeed hired (or at least interviewed). This evaluation is likely to give us a lower bound of our model's performance as we can assume that the majority of candidates who got hired (or interviewed) can indeed be considered relevant. Conversely, however, not everybody who was not hired (or interviewed) has to be non-relevant.

Additionally, we will also take advantage of our relevance assessment set as a secondary evaluation set. This set offers more fine-grained relevance labels, some of which correspond roughly to the hiring decisions in meaning: The labels *would-interview* and *would-hire* could be mapped to *interviewed* and *hired*, respectively, while the remaining labels *not-relevant*, *somewhat-relevant* and *overqualified* could be considered *rejected*. As we do not explicitly train our models for this set (too small for training), it is likely to perform worse on this set than on the hiring decisions. However, since the meaning of the labels in these two sets are very much related, optimising the performance on one set should also help the task in the other set. We will present the results of these two evaluation option in Section 5.3.

The metrics we use for our evaluations in the thesis are typical in IR tasks. The simpler set retrieval metrics recall and precision operate on the binary distinction between relevant vs. non-relevant documents. Metrics that take the concrete ranking into consideration are NDCG, MAP and P@k. In particular, NDCG (*normalised discounted cumulative gain*) is suitable for graded relevance labels (as in our case with *rejected*, *interviewed* and *hired*, which denote an increasing degree of relevance expressible as grades 0, 1 and 2, respectively). MAP (*mean average precision*) also operates on binary labels, hence, we would map our *rejected* label to grade 0, and both *interviewed* and *hired* to grade 1. The concrete definition of these metrics can be found in any IR textbook, e. g. in Manning et al. (2008).

2.6 THE CHALLENGES

Given the properties of our dataset for learning and the general setting of the task we face several challenges (despite the relatively restricted setting):

UNBALANCED DATA Similar as in other IR tasks the ratio of relevance vs. non-relevant documents is heavily skewed towards the non-relevant documents. This property of the data is especially problematic if a simple classification algorithm (corresponding to point-wise LTR, cf. Section 3.2.1) is used to make a two-class prediction since it will tend to classify (almost) everything as non-relevant, hence, not learning a meaningful model. Pair-wise and list-wise learning approaches will be more suitable.

SEMI-STRUCTURED DOMAIN Different from Web search, which is the more common domain of application of LTR, our documents are formed according to a predefined structure. Intuitively, this structure adds information to the documents that we would like to encode into the learning process, e. g. as features.

SPARSITY The difference between semi-structured and fully structured documents is that in our case despite the structural skeleton in the documents, which are given as fields, most of the field values are still unrestricted natural language text created by people. Hence, if we were to use this language data directly we will have to anticipate sparsity issues.

MISSING INFORMATION/NOISE As discussed in Section 2.3.1 we have to do the learning based on a dataset that is sure to contain a number of missing values and an unpredictable amount of noise. A strict training-testing procedure has to be put into place in order to avoid fitting to the noise.

FEATURES As far as we know there is only little work on feature-based retrieval/ranking in the recruitment domain (cf. Section 3.3.1), and often times even if there is some insight obtained for a particular system, it cannot be straightforwardly applied to a different system because of non-availablility or non-applicability.[6] Our goal is to explore a set of features that are tailored to our data and investigate how they influence the learned ranking. This will help us perform a kind of feature selection in order to train a model based on the most effective features.

6 E. g. Kokkodis et al. (2015) study online labour markets where they have access to user ratings of freelancers, which can be used as a feature denoting the reputation of a candidate. In any non-online labour market such information is usually not directly available.

3 RELATED WORK

3.1 LANGUAGE MODELLING FOR INFORMATION RETRIEVAL

There is an extensive body of research in the domain of probabilistic models for information retrieval, especially the work on language modelling approaches for IR. The following section reviews some classic work in document retrieval within the LM frameworks as well as research focusing on retrieval of semi-structured documents. Some of the models mentioned here will be picked up in later sections when we describe our own models and features derived from them.

3.1.1 *Models Derived from the LM Approach*

Due to its empirical success and the flexibility of its statistical formulation the language modelling approach has enjoyed considerable popularity in IR research and applications, and many variations and extensions of the original model described in the pioneering work of Ponte and Croft (1998) have been developed. In this section we will survey a number of theoretical models, some of which have served as inspiration for the current thesis. A more comprehensive and detailed review including examples of application can be found in Zhai (2008).

3.1.1.1 *Query Likelihood Models*

The main contribution of Ponte and Croft is to introduce a new way to score documents with respect to a query: For each document we first estimate a language model and then we rank the documents according to the likelihood of the given query being generated from these estimated models (hence, later denoted by query likelihood scoring or model). Hence, we rank documents higher if the query is a probable sample from the language models associated with the documents. Formally, the scoring function can be formulated as follows:

$$\mathrm{score}(Q, D) = p(Q|\theta_D), \tag{1}$$

where Q denotes the query, D denotes a document and θ_D denotes the language model estimated based on document D.

Depending on how we define and estimate θ_D we get different realisations of the query likelihood scoring. In Ponte and Croft's (1998) original paper the authors define what can be considered a multiple Bernoulli model for θ_D, i. e. they define binary variables X_i which represent the presence or the absence of words w_i in the query, $\theta_D = \{p(X_i = 1|D)\}_{i \in [1,|V|]}$. Thus, their model can be specified in full as follows:

$$P(Q|\theta_D) = \prod_{w_i \in Q} p(X_i = 1|D) \prod_{w_i \notin Q} p(X_i = 0|D). \tag{2}$$

Another possibility is to define a multinomial model, also commonly called a unigram language model: $\theta_D = \{p(w_i|D)\}_{i \in [1,|V|]}$. Such a model can take the counts of terms (so not just the presence or absence) directly into account. The query likelihood in this model can then be defined as follows:

$$P(Q|\theta_D) = \prod_{i=1}^{m} p(q_i|D) \tag{3}$$

Intuitively, this model captures the TF retrieval heuristic since the query likelihood is high for words that appear often in the document.

The remaining question is how to estimate the word probabilities in the corresponding models. This is usually done with the maximum likelihood estimator using the words in the document, with the underlying assumption that the document is a representative sample of θ_D. For instance, the unigram model can be estimated as follows:

$$\hat{p}(w_i|D) = \frac{c(w_i, D)}{|D|} \tag{4}$$

However, there is a problem with this estimator: Unseen words in the document will be assigned zero probability, which in turn will make the whole query likelihood zero, independent from the other terms in the query. One way to deal with this clearly undesirable characteristic is to apply a smoothing method to reserve some small probability mass for unseen words (cf. e. g. Zhai and Lafferty (2004)).

Interestingly, the original scoring function associated with the language modelling approach can be generalised to ranking by the conditional probability (Zhai, 2008; Manning et al., 2008). By applying the Bayes formula we can also introduce a document prior into the ranking function:

$$\text{score}(Q, D) = p(Q|D) = \frac{p(Q|D)p(D)}{p(Q)} \propto p(Q|D)p(D) \tag{5}$$

In web search the document prior can for instance incorporate some static ranking function such as the PageRank score, which is only dependent on the document but not the query. This general formulation also allows for different interpretations for $p(Q|D)$, thus opening up new modelling possibilities for the query likelihood.

3.1.1.2 *Document Likelihood Models*

Just as we rank documents by the likelihood of them generating the query, it is also possible to view the problem from the opposite direction: We could rank the documents according to how likely they are generated by some query model. Thus,

$$\mathtt{score}(Q, D) = p(D|\theta_Q). \tag{6}$$

The difficulty lies in defining and estimating θ_Q, since queries are usually a lot shorter than documents and any language model estimated solely based on the query will have to undergo thorough smoothing to be usable.

However, this formulation of the query model has the advantage that is it very intuitive to incorporate relevance feedback (Manning et al., 2008): The query model can easily be updated with higher probabilities for words that occur in relevant documents. This feat is less theoretically justifiable in the query likelihood model where the query is treated as a sample, i. e. a sequence of terms, from the document language model (Zhai, 2008). In Section 3.1.2 we will more systematically review some techniques for incorporating relevance feedback in language modelling approaches, some of which are more ad-hoc heuristics, while others are more theoretically grounded.

An empirically successful instantiation of this kind of model is the relevance model as defined by Lavrenko and Croft (2001), which also incorporates pseudo-relevance feedback into the estimation of the query model. In particular, Lavrenko and Croft estimate the query model based on the top-ranked documents and thus also assign high probabilities to words that occur frequently in documents which match the query terms well. The authors suggest two concrete estimation methods for θ_Q, of which we will only reproduce the first:

$$\begin{aligned}\hat{p}(w|\theta_q) &= \frac{p(w, Q)}{p(Q)}\\ &\propto p(w, Q)\\ &\approx \sum_{\theta_D \in \Theta} p(\theta_D)p(w|\theta_D)\prod_{i=1}^{m} p(q_i|\theta_D),\end{aligned} \tag{7}$$

where Θ is the set of smoothed document language models based on the top-ranked documents. This formulation of the query model can also be seen as a step towards bridging the potential vocabulary gap between queries and documents and directly model the information need of the user underlying a concrete query.

3.1.1.3 *Divergence Retrieval Models*

While query models represent the user's information need, document models can be interpreted to represent the topic or content of a document. Given these interpretations it seems natural to compare the correspondence of these models and rank the documents according to the document model's similarity/divergence to the query model. Lafferty and Zhai (2001) formulate the scoring function in this manner by using the Kullback-Leibler divergence:

$$\begin{aligned} \mathrm{score}(Q, D) &= -D(\theta_Q || \theta_D) \\ &= -\sum_{w \in V} p(w|\theta_Q) \log \frac{p(w|\theta_Q)}{p(w|\theta_D)} \\ &= \sum_{w \in V} p(w|\theta_Q) \log p(w|\theta_D) - \sum_{w \in V} p(w|\theta_Q) \log p(w|\theta_Q) \end{aligned} \tag{8}$$

$$\mathrm{score}(Q, D) = -H(\theta_Q, \theta_D) + H(\theta_Q) \tag{9}$$

Since the KL divergence[1] can be decomposed into the negative cross-entropy and the entropy of the query model, which is constant across all documents for a single query, this scoring function results in the same ranking as a function ranking based on the negative cross-entropy of the query model and the document model alone. Lafferty and Zhai (2001) have shown in experiments that this divergence-based ranking

1 Note that the KL divergence is an asymmetric measure. Hence, one could theoretically also formulate $\mathrm{score}(Q, D) = -D(\theta_D || \theta_Q)$ as a ranking function. This function would also take the entropy of the document models into account for ranking.

function is superior to models solely based on either document likelihood or query likelihood.

3.1.2 *(Pseudo-)Relevance Feedback*

While in some models relevance feedback can be naturally and directly incorporated into the model, other models might require some more heuristic methods. As alluded to earlier, it is not entirely clear how to include relevance feedback in a principled way in the query likelihood framework where the query is simply a sequence of terms.

However, a simple ad-hoc method that immediately comes to mind is to expand the query with additional query terms that have high probabilities in the relevant documents (but e. g. low probabilities in the collection). Even though this approach does not have a direct probabilistic interpretation within the query likelihood model, it has shown to been empirically effective in Ponte (1998). Because of this heuristic's simplicity and how it can be basically applied in any retrieval framework as an ad-hoc method, we conduct a small query (term) expansion experiment to investigate its effect on improving retrieval recall (details in Section 4.4).

The models that allow the definition and the estimation of a separate query model, on the other hand, make the expansion with relevant terms also interpretable in a probabilistic context. The basic idea in all of the proposed work is to re-estimate the query model based on the documents that are known to be relevant and as a consequence perform a massive query expansion. Zhai and Lafferty (2001), for instance, propose to interpolate an existing query model with a model estimated on based on the relevant feedback documents. Thus,

$$p(w|\theta_Q) = (1-\alpha)p(w|\theta_Q) + \alpha p(w|\theta_R), \qquad (10)$$

where θ_R is the model based on the relevance feedback documents (see the original paper for the proposed estimation methods).

Similarly, the already introduced relevance model (Section 3.1.1.2) estimates its query model based the top-ranked documents, effectively incorporating pseudo-relevance feedback so that terms absent in the query can still be assigned high probabilities if they are indicative of relevant documents.

3.1.3 *Semi-Structured Documents*

The previously introduced models all derive from the need to retrieve documents that have no or negligible structure given some keywords as a query. However, when the documents of interest do contain meaningful structure we could benefit from retrieval models which explicitly take the structure of the model into account instead of treating the documents as free text. Some traditional retrieval models have been extended to cater for this need by modelling a semi-structured document as a set of fields, i. e. $D = \{F_1, F_2, ..., F_{|D|}\}$, so that a query term match in one field could contribute more significantly to the ranking score than a match in another field. Examples of such extensions are the BM25F (Robertson et al., 2004) and the mixture of field language models introduced in Ogilvie and Callan (2003).

In this section we will focus one model in particular: Kim and Croft (2012) proposed the so-called field relevance model, a query likelihood model which aims at linking field weights to the notion of relevance in such a way that relevance feedback can be incorporated in a principled manner. Their scoring function is defined as follows:

$$\text{score}(Q, D) = \prod_{i=1}^{m} \sum_{F_j \in D} p(F_j|q_i, R)p(q_i|\theta_{F_j}) \tag{11}$$

The term $p(F_j|q_i, R)$ models the relevance of a query term distributed among the fields of the document, also called field relevance. Hence, query likelihood is calculated per field, where for each field a language model θ_{F_j} is estimated, and is weighted with the corresponding field relevance. Intuitively, this model captures the situation where the match of a query term in some field is more significant or meaningful than in others.

If we have a set of feedback documents D_R which are judged as relevant, they can be incorporated into $p(F_j|q_i, R)$ by estimating the field relevances based on them:

$$p(F_j|q_i, R) \propto \frac{p(q_i|F_j, D_R)}{\sum_{F_k \in D} p(q_i|F_k, D_R)} \tag{12}$$

$$= \frac{p(q_i|\theta_{R,F_j})}{\sum_{\theta_{F_k} \in \Theta_F} p(q_i|\theta_{F_k})} \tag{13}$$

where Θ_F denotes the set of smoothed language models estimated for different fields based on the set of relevant documents. However, since in practice relevance judgements are hardly ever available, the authors also propose several other sources of estimation for the field relevances and define the final estimator to be a linear combination of the various sources.

We subscribe to Kim and Croft's idea that some query terms are more strongly associated with certain fields than others. However, while they estimate it based on collection statistics and some other heuristics because no other information is available (in particular, the query is unstructured in their case), we want to encode certain dependencies known in our domain and in our queries directly into our model. We do this by creating features that capture the dependency of certain fields and what would be field relevances are automatically learned within a learning-to-rank framework. We describe the learning-to-rank approach to information retrieval in the following section and the details of our model proposal are given in Chapter 4.

3.2 LEARNING-TO-RANK

Different from traditional IR models feature-based retrieval models can combine a number of signals encoded as so-called features directly into the ranking model. How the features are combined into a ranking function can be learned with a machine learning algorithm that optimises a desired objective function. This learning task is referred to as *learning-to-rank* (LTR), which is briefly introduced in the following section. More detailed explanations and examples can be found e. g. in Liu (2009) and Li (2014).

3.2.1 *Definition*

LTR is an inherently supervised task, i.e. we need a training set that has appropriate relevance labels associated with the records.[2] The training data is made up of queries and documents, where each query has a number of documents associated with it. Each query-document pair has a relevance label associated with it, which denotes the document's level of relevance with respect to the query. Formally,

$$S = \{(Q_i, \mathbf{D}_i), \mathbf{y}_i\}_{i=1}^{m} \tag{14}$$

where Q_i denotes the i-th query in the set of m queries, $\mathbf{D}_i = \{D_{i,1}, \dots, D_{i,N_i}\}$ denotes the corresponding documents and $\mathbf{y}_i = \{y_{i,1}, \dots, y_{i,N_i}\}$ the corresponding relevance labels.

A feature vector is created from feature functions, which map a query-document pair to a vector in a high-dimensional feature space, i. e. the training data can be concretely formulated as

$$S' = \{(\mathbf{X}_i, \mathbf{y}_i)\}_{i=1}^{m} \tag{15}$$

2 Though, there are ways to automatically generate labels that can also function as relevance labels e.g. from search log data.

where $\mathbf{X}_i$ is a set of feature vectors computed based on query-document pairs made of query Q_i and its corresponding documents $\mathbf{D}_i$, with $\mathbf{y}_i$ as the corresponding labels.

The goal of LTR is to learn a ranking function, which, given an unseen query and a set of associated documents as represented by a list of feature vectors $\mathbf{X}$, can assign a score to each of the documents, i. e.,

$$\mathbf{score}(Q, \mathbf{D}) := F(\mathbf{X}) \tag{16}$$

Hence, during testing or application phase for each new query and a set of documents that should be ranked, we create a set of corresponding feature vectors and apply the trained model to the vectors to obtain a set of scores. These scores can be used to rank the unseen documents w.r.t. the given query.

3.2.2 *General Training Approaches*

Depending on how the learning objective is formulated LTR gives rise to three main training approaches: point-wise, pair-wise and list-wise learning.

In the *point-wise* approach the ranking problem is in fact transformed into a classification or regression problem, where the list structure of the original problem is neglected. I. e., each feature vector derived from query-document pairs is assumed to be an independent data point and the objective function (e. g. minimise some loss function based on misclassification) is computed based on costs/losses of individual data points. With this reformulated training data any already existent classification, regression or ordinal regression algorithm can theoretically be applied and a ranking can be devised based on the scores or grades returned by the model.

The *pair-wise* learning approach also does not take the list structure of the ranking problem into consideration, however, different from the point-wise approach, it uses the ordering of document pairs and creates new feature instances as preference pairs of feature vectors: For instance, for a given query Q_i, if $D_{i,j}$ has a higher relevance label than $D_{j,k}$, a preference pair $\mathbf{x}_{i,j} \succ \mathbf{x}_{i,k}$ is created from their respective feature vectors. These preference pairs can be considered positive instances in a new classification problem (a negative instance can be created from the reverse), for which existing algorithms can be employed. The loss function is then defined in terms of the document/vector pairs. A notable example is Herbrich et al. (1999), in which a linear SVM is employed and preference pairs are formulated as the difference of feature vectors, i. e. $\mathbf{x}_{i,j} - \mathbf{x}_{i,k}$. Other pair-wise algorithms include RankNet (Burges et al., 2005), which uses Neural Network as ranking model and cross-entropy as loss function, and RankBoost (Freund et al., 2003) based on the technique of boosting.

The *list-wise* approaches model the ranking problem in a more natural way in the sense that it incorporates the list structure into both the learning and the prediction procedure. Furthermore, classical IR metrics such as NDCG can be directly optimised in the loss function, making for instance relevant documents on top weigh more than relevant documents at a lower rank (which is not the case in the pair-wise optimisation scheme). Concretely, a training instance in a list-wise algorithm is a ranking list, i. e. all the feature vectors associated with one query, rather than a vector derived from a query-document pair as in the previous approaches. In this formulation of the problem, however, conventional machine learning techniques cannot be directly applied.

Some advanced list-wise algorithms include AdaRank (Xu and Li, 2007), ListNet (Cao et al., 2007) and LambdaMART (Wu et al., 2010).[3]

3.3 MATCHING/RANKING APPLICATIONS

3.3.1 *Recruitment*

Yi et al. (2007) experiment with a problem task similar to ours: Matching a large collection of semi-structured CVs to real-world job postings.[4] Their approach adapts relevance models (cf. Section 3.1.1 and Lavrenko and Croft (2001)) to a structured version by estimating relevance models for each field based on labelled data (relevance judgements in their case). Even though the authors are able to improve their baselines with the proposed method by a small percentage, they acknowledge that the matching task in this domain is very difficult.

Singh et al. (2010) describe PROSPECT, a full e-recruitment system that is similar to our current system (without the re-ranking module): Important information such as work experience and skills are mined from candidates CVs with a dedicated information extraction module and the values are then indexed in a search engine, which supports full text search. Additionally, recruiters can use search facets to further filter the list of candidates by specifying certain criteria for specific fields. The authors report that Lucene's out-of-the-box ranking model with a boost on skills performs best in their ranking experiments.[5] Note that this traditional kind of retrieval model does not involve any machine learning or supervision.

3 LambdaMART is in fact difficult in classify in this scheme as it directly optimises a list-wise IR measure but still uses pairs as input samples in the implementation. Therefore it is sometimes also classified as a pair-wise algorithm.

4 Though instead of generating dedicated queries from those posting they simply use the whole posting as a query.

5 In fact, ElasticSearch, our search engine, is also built based on the Lucene engine.

Mehta et al. (2013) decompose the ranking model into several independent rankers denoting different dimensions of suitability of the candidate other than just the technical match (i. e. how well their skills match the job offer): the quality of the candidate as suggested e. g. by the university or last employer, onboard probability (how likely is the candidate to accept the offer?) and attrition probability (how likely is the candidate to stay with the company?). For each of these dimensions the authors train separate classifiers based on labelled training data (historical records in their case) and finally aggregate the individual ranker's scores as a linear combination to produce a final ranking. The authors argue that in this formulation of the aggregation companies can determine the importance of the different dimensions by themselves simply by selecting their own weights for each dimension.

The most recent work that we know of that displays a feature-oriented view of the matching problem in the recruitment domain is Kokkodis et al. (2015). Their focus is on online labour markets (OLM) where they extract features based on the freelancers' profiles, the employers' profiles and the job description. Their final ranking (in their best model) is based on the hiring probability score of a candidate w.r.t. an job description by a certain employer, estimated by means of a hand-crafted Bayesian Network model built with their features.

Note that our work is different from all of the approaches above in the sense that we take on a feature-oriented view of ranking and use learning-to-rank methods to learn a ranking model based on hiring decision labels. While Kokkodis et al. (2015) also use hiring decisions as labels, they consider them unsuitable for LTR for their purposes. Mehta et al. (2013) take advantage of supervised machine learning methods, however, their labelled training data are much more diverse than ours. Yi et al. (2007) and Singh et al. (2010) do without machine learning completely and only rely on traditional IR retrieval models.

3.3.2 *Other Domains*

One notable work in a different yet similar domain is Diaz et al. (2010)'s work on online dating. The domain of online-dating is in many ways similar to the domain of recruitment as it is another instance of so-called *match-making* systems. As in our work, the authors formulate the matching/ranking problem as an IR problem and take on a feature-oriented view by extracting a number of features from both the structured and unstructured portions of users' profiles and queries.[6] Similar to our domain, the definition of relevance in online dating is also non-trivial. The authors resolve to using hand-crafted, heuristic rules based on post-presentation user interactions (e. g. exchange of phone numbers vs. unreplied message) to generate their own relevance labels for their data, which they use as their gold standard labels. These labels are admittedly noisy, but, as the authors argue, they should still be more accurate than human relevance judgements.

6 In their set-up the retrieval of matching profiles for a certain user involves more components than in our domain: the searcher's profile, the searcher's query, candidates' profiles and the candidates' queries.

4 FIELD RELEVANCE MODELS FOR CVS

As in many ML learning undertakings, acquiring a sizeable dataset that is suitable for learning is often the most difficult task in the domain of recruitment. Real CVs of ordinary job seekers are sensitive and often subject to privacy concerns. However, what is even more rarely available are data that can be used as labels in supervised learning. Collecting high-quality relevance judgements by human annotators is expensive and time-consuming, as a large amount of data has to be assessed by experts. Even hiring decisions, which is what we will use to approximate relevance, are hard to obtain.

This is the main motivation for our *heuristic field relevance models*, which essentially aim to take advantage of unsupervised data (most often a collection of CVs) to approximate some notion of relevance. We propose to derive models from the internal structure of CVs and use them in combination with a smaller set of relevance labels to benefit retrieval tasks. In the following we will first illustrate the proposed model with a concrete example (Section 4.1), which should facilitate the understanding of the general idea (Section 4.2 and Section 4.3). We report and analyse the results of a small, self-contained experiment in Section 4.4, which uses the proposed example model to perform query expansion.

4.1 AN EXAMPLE: MODEL OF JOB TRANSITIONS

In this example we were interested in modelling typical career advancements (a similar idea is pursued in Mimno and McCallum (2007)), which can be seen as a proxy for candidates' preferred career choices. In other words, if many candidates move from job A to job B, the transition from job A to B should be a typical and presumably attractive career step from the candidates' point of view given their current position.

Since such job transition information is usually readily available in CVs (e. g. in the *work experience* section), we can build a model of typical job transitions in a completely unsupervised manner without requiring any labelled data (so without any relevance judgements or hiring decisions w. r. t. specific vacancies). Hence, because of what we know about the conventions of structuring a CV, we in principle get historical hiring decisions *for free*.[1]

The obvious drawback of this information is that we only have access to reduced information, i. e. in most cases we cannot rely on any additional vacancy information apart from a job title. On the other hand, the big advantage of this approach is that CVs are usually much more readily and in a larger number available than any kind of explicit hiring decisions. The main goal of this approach is to take advantage of the large number of CVs and possibly combine models derived from it with a smaller number of hiring decisions to obtain a better ranking result.

4.1.1 *The Taxonomy of Jobs*

In our parsing model every job title is automatically normalised and if possible mapped to a *job code* (an integer value). A set of related job codes are grouped together to have one *job group id*, and a set of group ids comprise a *job class*, hence, making up a job hierarchy as

1 However, since CVs are written by people seeking to present themselves in the best possible way we have to expect some embellishments that may not reflect reality (e. g. by stating having stayed with a company for longer than in reality).

illustrated in Figure 3. The existing software maintains 4368 job codes that represent the universe of job titles in a relatively fine-grained manner, yet less fine-grained and sparse than some of the original linguistic material (some examples are given in Table 2). There are 292 job group ids and 25 job classes.

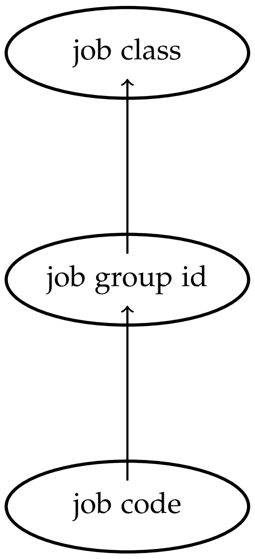

Figure 3: The structure of the internal job taxonomy.

Since the *job code* field can be found in each experience item in the experience section (if the normalisation and code mapping was successful) and provides us with less sparse representations of a job, we will use this field (instead of the original *job title* field) for the model of job transitions.[2] So more concretely, it is a model of transitions from *job code* to *job code*.

2 Theoretically we could also use the *job group id* and *job class* fields to build models. However, our experiments have shown that they are both too coarse-grained to be useful.

Table 2: This table illustrates the job taxonomy with a few example jobs (English translations of the Dutch original).

job class	**job group**	**job code**
engineering	business administration and engineering experts	product engineer
engineering	engineering managers	lead engineer
healthcare	specialists and surgeons	psychiatrist
healthcare	medical assistants	phlebotomist
ICT	programmers	Javascript programmer
ICT	system and application administrators	system administrator

4.1.2 *Predecessor and Successor Jobs*

Using a language modelling approach and "job bigrams"[3] we can estimate a model based on "term" frequencies, which predicts the probability of a job occurring after another job:

$$\hat{P}_{succ}(job_t|job_{t-1}) \overset{MLE}{=} \frac{c(job_t, job_{t-1})}{c(job_{t-1})} \quad (17)$$

where $c(.)$ denotes a function that counts "term occurrences" in the collection of CVs (more specifically, the collection of job sequences). As always when using MLE some kind of smoothing is required (more details about our smoothing approach is given in Section 4.3.2).

3 We will use a slight variation of simple bigrams by also allowing 1-skip-bigrams, cf. Section 4.4.

Conversely, it is also possible to go back in time and predict the probability of predecessor jobs:

$$\hat{P}_{pred}(job_{t-1}|job_t) \overset{MLE}{=} \frac{c(job_{t-1}, job_t)}{c(job_t)} \tag{18}$$

These models are interpretable in the sense that they give us insights about what typical career paths look like according to our data. In addition, because of the language modelling approach these models can straightforwardly be used to compute features for the LTR task as explained in Section 5.2.

4.2 UNSUPERVISED MODELS

The previous model built on *job codes* in the experience section can be generalised to other fields and several variations are possible by tweaking certain parameters. In the example above we only used one field to estimate the language model of job transitions and we used bigrams because of the semantics of job transitions. However, it is also possible to take into account field dependencies (e. g. by conditioning the values of one field on the values of another field), or to use arbitrary *n*-grams to build the model (provided the data does not get too sparse).

Modelling field dependencies can be useful in those cases where we intuitively assume that there must be some kind of dependency, e. g. between the candidate's education and the candidate's skills, or the candidates most recent job title and their listed skills. This kind of two-field dependency can for instance be formulated as in the following (for the bigram case), where f_i, f_j denote concrete values from some dependent fields F_i and F_j.

$$\hat{P}_{M_{F_i,F_j}}(f_i|f_j) \overset{MLE}{=} \frac{c(f_i, f_j)}{c(f_j)} \tag{19}$$

Note that the value we condition on, f_j, is a value in field F_j, while the value predicted, f_i, comes from a different field, F_i.

The model can also be sensibly formulated in terms of unigrams:

$$\hat{P}_{M_{F_i,f_j}}(f_i) \overset{MLE}{=} \frac{c_{f_j \in F_j}(f_i)}{N_{f_j \in F_j}},$$

where $c_{f_j \in F_j}$ denotes a counting function that only counts the specified term in documents where $f_j \in F_j$, and $N_{f_j \in F_j}$ denotes the number of documents s. t. $f_j \in F_j$.

4.3 DISCUSSION OF VARIATIONS

4.3.1 *Supervised Models*

The field model we proposed above relies on the availability of a large amount of unlabelled data, in particular, CVs. However, it is possible to imagine a supervised variation of dependent field models where we take into account e.g. hiring decisions by only considering vacancy-CV pairs where the CV belongs to a hired (relevant) candidate.

For instance, we could build a model based on the job title in the vacancy and the skills of hired candidates, which would give us good predictions about which skills as they are listed in CVs are highly associated with which jobs. This kind of model could be useful in cases where the vacancy lists a set of skills that do not match the skills in CVs entirely because of the vocabulary gap of vacancies and CVs.

There is, however, an obvious drawback: Because hiring decisions or any kind of labelled data are much more scarce than unlabelled data we will most likely run into a sparsity problem with language data. Hence, this supervised variation can practically only be applied to highly structured fields (e. g. the language field where there is usually only a limited number of pre-defined values given a user base).

4.3.2 *"Relevance Feedback" for Smoothing*

In the introduction of the field models above we have wilfully omitted any details about smoothing, which is, however, inevitable in any kind of approach involving language modelling since we can never have so much data as to cover all possibilities of language.

There is a number of smoothing techniques (Chen and Goodman, 1999; Zhai and Lafferty, 2004) to choose from and usually applications determine experimentally which technique and which parameters are most suitable to their task with some held-out dataset. We take the same approach in the experiments described in this thesis, however, we want to propose a small variation given our domain and task.

The models we build might be unsupervised, yet given that we have a small number of labelled data we could use this small set to construct a held-out set in the same format as the original unlabelled set to estimate smoothing parameters from. As our models are estimated based on n-gram counts of field values, we could create the same n-grams based on the labelled data and feed it back into the original models (reminiscent of *relevance feedback* in traditional IR) by means of choosing appropriate smoothing parameters. Depending on the task different optimisation objectives can be chosen for the held-out dataset (e. g. we optimised for perplexity for our language models but for IR tasks any IR measure could also be used).

4.4 TERM-BASED RECALL EXPERIMENT

To demonstrate the field relevance model proposed in this section we conduct a simple experiment with the model of job transitions as described in Section 4.1. In this experiment we will expand some of our original queries with high-likelihood predecessor jobs given the advertised job in the original query. I. e., given a job code job_i in the query, we will add additional job codes job_j to the query according to the model $\hat{P}_{\mathrm{pred}}$ if $\hat{P}_{\mathrm{pred}}(\mathrm{job}_j|\mathrm{job}_i)$ is high enough.[4] Adding additional query terms will allow us to retrieve candidates who would have not otherwise been retrieved.

4.4.1 *Set-up and Parameters*

We adapt the model $\hat{P}_{\mathrm{pred}}$ in Section 4.1, a model of *job code* transitions that gives predictions about predecessor jobs, with a slight variation: Instead of just bigrams we also allow 1-skip-bigrams (Guthrie et al., 2006), i. e. we allow skips of 1 to construct the bigrams based on which the language model is estimated. An illustration is given in Table 3.

The reasoning behind this is that careers are assumed to be somewhat flexible and it should be possible to sometimes skip one step in the ladder to get to a higher position. Furthermore, the skipgrams can model the situation where a job in a person's career might diverge from its "normal" course (given a previous or a successor job as a reference point). If that particular job is indeed unusual as a career choice, it will have a lower count compared to jobs in line with the given career.

4 There are of course more sophisticated methods of query expansion, which are, however, out of the scope of our project. For our purposes this simple set-up will be sufficient.

Table 3: An example illustrating how 1-skip-bigrams are constructed compared to simple bigrams.

$job_{t-4} \rightarrow job_{t-3} \rightarrow job_{t-2} \rightarrow job_{t-1} \rightarrow job_t$		
bigrams	(job_{t-4}, job_{t-3}),	(job_{t-3}, job_{t-2}),
	(job_{t-2}, job_{t-1}), (job_{t-1}, job_t)	
1-skip-bigrams	(job_{t-4}, job_{t-3}),	(job_{t-4}, job_{t-2}),
	(job_{t-3}, job_{t-2}),	(job_{t-3}, job_{t-1}),
	(job_{t-2}, job_{t-1}),	(job_{t-2}, job_t),
	(job_{t-1}, job_t)	

We estimated the model based on approximately 400CVs and only considered experience items that contain a date and that start after the candidates highest education (to avoid including low-level student jobs that are less relevant for one's career path). To smooth the model we applied absolute discounting with linear interpolation (Chen and Goodman, 1999) and estimated the smoothing parameters based on a held-out set constructed from a small set of hiring decisions (200 queries) as described in Section 4.3.2.

We automatically generated semi-structured queries for the set of 99 vacancies that were used for collecting relevance judgements. However, only a subset contained a job code and of those we only expanded 32 queries. The reason for the rather small number of expanded queries is because we applied some rather strict rules for query expansion, which were determined experimentally on a small set of queries: For each job code in the query, we only consider the top-10 ranked predictions and only include them as an expansion term if they are *not* more likely to be predictions of 20 other jobs or more. In other words, we only expand with jobs that are very likely to be predecessors of the posted jobs and less likely to be predecessors of most other jobs. We also exclude expansion of queries containing job codes for which have low evidence (seen less than 20 times in the data). We employ this cautious strategy because we assume that for certain queries (and jobs) expansion simply does not make sense (e. g. lower-level jobs for which no typical career path exists) or the most probable predecessor job is in fact the job itself.

Both the queries and the expanded queries are issued to a search engine containing a collection of roughly 90K indexed documents.[5] The retrieved documents are compared to the relevance labels as collected for the relevance assessment set (cf. Section 2.3.2) based on which IR metrics can be computed. For this purpose the labels *not-relevant*, *somewhat-relevant* and *overqualified* are all mapped to *not-relevant*, thus, only *would-interview* and *would-hire* are considered relevant candidates.

We also conducted the same experiment with the hiring decision set by expanding approximately 200 queries (with the same restrictions as described above) and evaluating recall with the labels in the hiring decisions. However, as explained in Section 2.3.1 this set is less suitable for recall-oriented experiments as for many retrieved documents there is simply no relevant label associated (since the search engine also retrieves non-applicants as potentially relevant candidates). Nevertheless, we report the numbers here for the sake of completeness.

4.4.2 *Results and Discussion*

We present and discuss the results of the experimental set-up described above. However, as with every recall-oriented IR experiment

5 The collection does not contain the full 300K documents because at the time of pooling the documents for collecting the relevance judgements the search engine also only contained 90K documents. Hence, all experiments based on the dataset of relevance judgements have to use the same configuration to be valid.

where the relevance judgements have been collected prior to the execution of the experiment through pooling, we are likely to encounter retrieved documents without any judgement, which are automatically considered non-relevant in the computation of IR metrics. We should be aware of this fact when interpreting the numbers.

4.4.2.1 *The Relevance Assessments*

Table 4: Recall and MAP of the issued queries evaluated with the relevance assessment set.

	#ret. rel. docs	**recall**	**MAP**
original queries	423	0.45	**0.1874**
expanded queries	**459**	**0.48**	0.1773

The set of expanded queries results in a slightly higher recall compared to the original, non-expanded queries when evaluated with the relevance assessment dataset, as shown in Table 4. However, what is also immediately observable and not that surprising is that with the higher recall, the precision is slightly decreased, as manifested in the lower MAP.

This result suggests that while we do manage to retrieve some more relevant candidates with the expanded queries overall, some of the less relevant documents now receive a higher rank than before. This kind of tradeoff between higher recall and lower precision is a common problem and it will be implicitly addressed in the LRT experiments when we optimise for precision-oriented objectives (cf. Chapter 5).

Since the set of evaluation queries is rather small in this experiment we also inspected some individual queries to understand the effects of the query expansion and of the model of job transitions better. One important observation is that despite the overall increase in recall, expanding with jobs as predicted by the model only benefits certain queries or certain jobs.

Table 5 shows some example job codes which occurred in the queries where the expansion resulted in higher recall. Jobs in this category are usually highly specialised jobs for which the job transition model predicts semantically very related jobs.

Table 5: Job code examples where the query expansion results in higher recall.

posted job	**expanded with**	**#ret. rel. docs**
e-business consultant	business adviser	$12 \rightarrow 13$
nursing assistant	carer	$18 \rightarrow 21$
programmer C, C++, C#	software engineer, programmer (other)	$32 \rightarrow 38$

Another category of jobs is shown in Table 6, where the expansion did not improve the recall. These are usually jobs that are less specialised in their title and do not really have a typical career path associated with them. For this kind of jobs a lot of contextual information is required to determine what the job is concretely and a model based on just the job code is simply not expressive enough to improve the retrieval task. Expanding the queries with predictions anyways leads to the retrieval of candidates who are semantically only remotely related to the posted job and, thus, did not occur in our initial pooling process. These additional candidates are automatically considered non-relevant (as there is no relevance judgement associated with them) and for these very general jobs this treatment intuitively makes sense.

However, there is a third category of jobs (Table 7) which also has lower recall. Here we argue that the results have to be taken with a

Table 6: Job code examples where the query expansion in fact results in lower recall.

posted job	**expanded with**	**#ret. rel. docs**
administrative assistant finance	accounts payable assistant	$30 \rightarrow 24$
administrative assistant	administrative assistant finance	$29 \rightarrow 25$
coordinator	project leader (other), administrative assistant	$2 \rightarrow 1$

grain of salt, as the expanded queries retrieve a number of documents for which there are simply no relevance judgements in the set (thus, automatically assumed to be non-relevant), similar to the previous category. Yet, given the semantics of the expansion job codes the real number of retrieved relevant documents is likely to be higher.

Table 7: Job code examples where the query expansion in fact results in lower recall, yet whose real recall is likely to be higher.

posted job	**expanded with**	**#ret. rel. docs**
personnel and labour experts	human resources officer	$10 \rightarrow 9$
functional administrator	application manager	$16 \rightarrow 15$
head of legal affairs	jurist	$3 \rightarrow 2$

4.4.2.2 *The Hiring Decisions*

Similar to the relevance assessment set we are able to observe a small increase in retrieved relevant documents as shown in Table 8. However, as explained before the overall recall is much smaller and as a result also the MAP.

Table 8: Recall and MAP of the issued queries evaluated with the hiring decisions.

	#ret. rel. docs	**recall**	**MAP**
original queries	95	0.10	0.0144
expanded queries	**101**	**0.11**	**0.0146**

4.4.3 *Conclusion*

The field relevance models as proposed in this section can provide us with insightful information about our domain, i. e. about people's typical career paths and typical associations of certain properties such as job title, education and skills. However, as we have seen in the experiments reported above their degree of usefulness in the retrieval task can depend on how and where they are applied. The simple query expansion experiment shows that a model built solely on the work experience section of CVs has the potential to benefit recall of the retrieval task for a number of queries, yet some refinement is in need if we do not want to trade off precision. This issue is implicitly addressed in the next section where we propose features built based on the field models and at the same time introduce supervision with hiring decision labels in order to train ranking models that also optimise for precision-oriented measures.

5

LEARNING-TO-RANK EXPERIMENTS

In order to improve the retrieval results we adopt a machine learning approach, in particular, a learning-to-rank approach (cf. Section 3.2), which takes the initial search ranking as an input and recomputes ranking scores for the returned documents based on a pre-trained ranking model. Thus, the matching/ranking of candidates for job postings is cast as a common IR problem where more relevant documents (candidates) should be placed on higher than less relevant documents (candidates).

In the following we will present the details of our experiments, how we processed the our data (Section 5.1), which features we extracted (Section 5.2), the results we obtained on a test set (Section 5.3) as well as analyses of our results from different perspectives (Section 5.4).

5.1 PREPROCESSING AND FILTERING OF THE DATA

Our initial raw data is of considerable size but only a subset of the original data is truly useful once we have filtered out vacancies that either do not have any relevant candidates or too many applicants to be useful for learning.[1] However, since it is not entirely clear which filtering criteria make most sense, we devised four sets of data, all

1 We set the cut-off point at 100 applicants. Our reason for discarding vacancies with a large number of applicants is the high biasedness of the these vacancies. In such cases the number of candidates that are marked as relevant, i. e. are hired or interviewed, is still very low simply because a company normally does not have the capacity to interview all of the relevant candidates. Hence, we assume some arbitrariness in these vacancies, which makes it harder to learn from them.

created by different degrees of "filtering strength". An overview of the datasets and the criteria according to which they are created is given in Table 9. As explained in Section 2.4 we processed the raw hiring decisions to generate three labels: *hired*, *rejected* and the tentative *interviewed* label, where the latter is heuristically gathered, e. g. from information about personal meetings and rejection reasons and, hence, is considered an optional label.

Table 9: An overview of the different datasets that are created according to different filtering criteria (number in parentheses refers to the number of vacancies in the set). The abbreviations *h*, *i* and *r* stand for *hired*, *interviewed* and *rejected*, respectively, where *hired* and *interviewed* are considered relevant.

dataset	**labels**	**#rel.**	**#non-rel.**	$\frac{\textbf{\#rel.}}{\textbf{\#non-rel.}}$	**#CVs**
HR (3145)	h, r	$\geqslant 1$	$\geqslant 1$	$\geqslant \frac{1}{50}, \leqslant \frac{1}{2}$	$\geqslant 5, \leqslant 100$
HIR1 (4876)	h, i, r	$\geqslant 1$	$\geqslant 1$	$\geqslant \frac{1}{50}, \leqslant \frac{1}{2}$	$\geqslant 5, \leqslant 100$
HIR2 (5938)	h, i, r	$\geqslant 1$	$\geqslant 1$	$\geqslant \frac{1}{50}, \leqslant 1$	$\geqslant 5, \leqslant 100$
HIR3 (6971)	h, i, r	$\geqslant 1$	$\geqslant 1$	-	$\geqslant 5, \leqslant 100$

For the queries and documents belonging to these four datasets we computed the corresponding feature vectors. Table 10 shows the number of feature vectors associated with each dataset.

Table 10: Number of queries and feature vectors per dataset.

dataset	**#queries**	**#feature vectors**	**#pos. samples**
dataset HR	3145	70281	4939 (7%)
dataset HIR1	4876	144391	19337 (13%)
dataset HIR2	5938	164144	26235 (16%)
dataset HIR3	6971	186171	35782 (19%)

5.2 FEATURES

We take on a feature-oriented approach for obtaining a ranking model, i. e. we extract features from our data and learn through machine learning algorithms how these features can be combined in order to achieve a better ranking result.

5.2.1 *Basic Features*

Our basic set of features are based on the fields of the indexed documents and the queries: Most of these features can be described as "match" features, i. e. they denote whether the value(s) of a field given in the query matched the value(s) of the same field in the document. This information is usually returned by the search engine in the form of a score for each indexed field, which can be directly used as a feature.

In the following two sections, Section 5.2.1.1 and Section 5.2.1.2, we describe some details of our basic features. Section 5.2.1.3 reports a small experiment that indicates the informativeness of search scores for LTR.

5.2.1.1 *Match Features*

Match features are crucial query-document features, i. e. they depend both on the query and the document, hence, giving an indication about how well a document matches a specific query. Table 11 shows the fields for which we have defined match features. For the fields that can have multiple values, e. g. *compskills*, we concretely defined several match features: 1) the match score averaged over all values given in the query, 2) the maximum match score (only relevant for the float score variant, cf. Section 5.2.1.3), and 3) the sum of all match scores.

In addition to query terms associated with certain fields most of our queries also contain fulltext search terms, i. e. terms that can match in any field. Hence, we also defined match features for these terms.

Table 11: Fields for which "match" features are extracted if available.

field	**meaning in CV**	**meaning in query**
jobtitle-onlyrecent	recent ($<$ 5 years) job title(s)	posted job title
jobcodes-onlyrecent	recent job code(s)	posted job code
jobgroupids-onlyrecent	recent job group id(s)	posted job group id
jobclass-onlyrecent	recent job class(es)	posted job class
jfsector	last industry	posted industry
compskills	computer skills	required computer skills
location	city of residence	posted location
edulevel	highest education level	education requirement(s)
langskills	known languages	required languages
(*fulltext*)	(full text search terms)	

5.2.1.2 *Query- and Document-Level Features*

While the match features are clearly dependent both on the query and the document, we have also defined a number of features that are either only dependent on the query or on the document. An overview is given in Table 12.

Table 12: Basic document-dependent and query-dependent features.

field	**document-dependent**	**query-dependent**
compskills	#values	#values
jobcodesonlyrecent	#values, #unique values	
jobcodes	#values, #unique values	
edulevel	level code (1-5)	
experience	years of experience	
fulltext		#values

5.2.1.3 *Binary vs. Float ES Score*

Most search engines return the match information in terms of a binary score (1/0 for *matched/not matched*) for each value. However, their underlying ranking model in fact uses a more informative score (usually a float number $\geqslant 0$, depends on for instance how often a term matched) to compute the ranking. This score is usually not returned by default since getting it is an expensive operation that is normally not justified. However, we expect it to make a difference in learning and therefore we conducted an experiment with both versions of the scores to find out more about its impact on retrieval.

For this experiment we used our biggest dataset *HIR3* (cf. Section 5.1) and trained the same ranking algorithm on the two versions of search scores. For this experiment we used the 26 basic features, of which 13 are based on search scores. As a ranking algorithm we chose LambdaMART optimising for NDCG@10 and trained with the following parameters: #trees = 1000, #leaves = 10, minimum leaf support = 1, learning rate = 0.1, early stop = 100. The two rankers were trained on the same training set and were tested on the same test set.

Even though a small improvement can be achieved with also the binary scores, Table 13 shows that the float scores do contain more

Table 13: Comparison of binary and float search scores on the test set.

	NDCG	MAP	P@5	P@10
ES baseline	0.392	0.3376	0.207	0.1883
binary search scores	0.4089	0.3525	0.2193	0.1958
float search scores	**0.4302**	**0.3732**	**0.2312**	**0.1992**

information such that a even better ranker can be trained. Extracting the original float scores from the search engine is an additional effort, however, it should pay off in terms of better retrieval results on all crucial metrics.

5.2.2 *Word-Based Features*

In addition to the basic, "dense" features described above we also include a number of bag-of-words-based, sparse features. These features correspond to the occurrence of certain words in the field *jobtitlesonlyrecent*.

To determine the vocabulary we first collected the most frequently occurring job title words from both the CVs and the queries (vacancies) and stemmed them using an English and Dutch stemmer. After removing some morphological redundancies manually we finally end up with 326 word stems obtained from CVs and 157 stems obtained from vacancies. Some examples are given in Table 14.

In the case of the query stems, the derived features simply denote whether the stem occurs in the queries job title or not. The document-dependent features, on the other hand, describe the number of times the CV stems occur in the multi-valued *jobtitlesonlyrecent* field in the document.

Table 14: Some examples of the word-based features derived from frequent job titles.

job title words	**examples**
in CVs	admin, assist, bank, beheerd, comput, directeur, docent, financ, horeca, ict, juridisch, klanten, logist, onderhoud, projectmanag, schoon, teamleid, verkoop
in queries	adviseur, assist, beveil, commercie, directeur, gezondheid, horeca, informati, monteur, projectleid, recruit, secretar, vrijwillig

5.2.3 *Model-Based Features*

In this section we describe how we used our models of job transitions as described in Section 4.1 to build further query-document features. Evidently, the idea of these features are very general and, thus, can be applied to any field relevance model.

5.2.3.1 *Likelihood-Based Score*

In line with traditional IR models that compute a score for each document by calculating the query likelihood based on an estimated document model or the document likelihood by an estimated query model (cf. Section 3.1.1), we can also compute query and document likelihood based on our model of job transitions.

Given a job code in the query job_Q and some job codes $\text{job}_{D,i}$ in the document's field *jobcodesonlyrecent*, we can compute the query likelihood features as in the following:

$$\begin{aligned} f_{QL_{max}}(Q, D) &= \max_i \hat{P}_{succ}(\text{job}_Q|\text{job}_{D,i}) \\ f_{QL_{avg}}(Q, D) &= \operatorname*{avg}_i \hat{P}_{succ}(\text{job}_Q|\text{job}_{D,i}) \end{aligned} \tag{20}$$

Similarly, the document likelihood features can be specified as follows:

$$\begin{aligned} f_{DL_{max}}(Q, D) &= \max_i \hat{P}_{pred}(job_{D,i}|job_Q) \\ f_{DL_{avg}}(Q, D) &= \operatorname*{avg}_i \hat{P}_{pred}(job_{D,i}|job_Q) \end{aligned} \tag{21}$$

Intuitively, these features model how similar the career path of a given candidate would be to a typical career path involving the given jobs. Depending on which job is advertised in the job posting, a more or less clear career path is associated with it. Hence, these likelihood features should be combined with with query-level features.

5.2.3.2 *Divergence-Based Score*

The divergence-based scoring takes advantage of having both a document model and a query model and computes the difference between these models. The intuition behind this scoring method is that for a good match of a document and a query the two corresponding models should be similar to each other. We can adapt this idea to our model of job transitions in a couple of ways.

Given a number of job codes in the *jobcodesonlyrecent* field in a document we could estimate a unigram LM on the fly (e. g. with Dirichlet smoothing) and check its correspondence to the query model as given by the job code in the query. Formally,

$$\hat{P}^{(D)}_{emp}(job_D) = \frac{c(job_D)}{N_D} \tag{22}$$

$$f_{KL_{emp}}(Q, D) = KL(\hat{P}_{pred}(.|job_Q)||\hat{P}^{(D)}_{emp}), \tag{23}$$

where $\hat{P}^{(D)}_{emp}$ denotes the estimated unigram LM for document D and N_D the number of job codes in document D. $KL(.||.)$ is the Kullback-Leibler divergence (the definition can be found in Section 3.1.1.3).

The idea of this feature is a document is ranked higher if its LM of job codes corresponds more closely to the distribution of typical predecessor jobs than other document's LMs.

Another feature variant is to calculate the divergence based on the pre-estimated document models and query model as in the following:

$$\begin{aligned} f_{KL_{min}}(Q,D) &= \min_i KL(\hat{P}_{pred}(.|job_Q)||\hat{P}_{succ}(.|job_{D,i})) \\ f_{KL_{avg}}(Q,D) &= \operatorname*{avg}_i KL(\hat{P}_{pred}(.|job_Q)||\hat{P}_{succ}(.|job_{D,i})) \end{aligned} \tag{24}$$

A low KL score means that there is a high overlap between the distribution of predecessor jobs given the query job code and the distribution of successor jobs given the document job code(s), possibly indicating that there are some "fuzzy" paths leading from the document jobs to the queried job (maybe through more than just one transition).

As a variation we can also substitute the asymmetric KL divergence with symmetric variants of the measure. Two versions that we used are given below as KL_{sym} and JS, where JS is called the *Jensen-Shannon divergence* (Lin, 1991).

$$KL_{sym} = KL(P||Q) + KL(Q||P) \tag{25}$$

$$\begin{aligned} M &= 0.5(P+Q) \\ JS &= 0.5KL(P||M) + 0.5KL(Q||M) \end{aligned} \tag{26}$$

5.2.4 *Overview*

Table 15 presents an overview of the feature sets described in the sections above and how many features are in each set. We will refer to these sets in our analysis of the impact of feature sets in Section 5.4.2.

5.3 RESULTS

After experimenting with several LTR algorithms such as SVMRank, RankNet, ListNet and LambdaMART, we decided to use LambdaMART

Table 15: An overview of our used feature sets and the number of features in each set.

feature sets	**#features**
basic features (Section 5.2.1)	26
doc job title words (Section 5.2.2)	326
query job title words (Section 5.2.2)	157
model-based features (Section 5.2.3)	9
all features	518

(Wu et al., 2010) for training our final ranking models as it consistently outperformed the other algorithms. We optimise NDCG@10 for all of our experiments as the optimisation of this one metric returns models that generally perform better on all other relevant metrics. We performed grid search to find the best performing parameters on a validation set and found that several combinations performed equally well. We finally settled on the following parameter combination: #trees = 1000, #leaves = 10, minimum leaf support = 1, learning rate = 0.1, early stop = 100. Our best performing model is trained on dataset *HIR1*. The results of the evaluation on the test set in the hiring decisions (628 queries) is given in Table 16. The trained model performs significantly better than the baseline on all metrics. The highest relative increase of 21% is achieved with P@5 (randomisation test, p-value < 0.01).

Even though our model was not in any way trained on the relevance assessments, we will also evaluate it on this set. The performance on this test set could give us an idea about how well the labels in relevance assessments are mirrored by real-word hiring decisions. Table 17 reports how our best model performs on the 99 vacancies from the assessment set compared to the search baseline performance.

Table 16: Comparison of the ElasticSearch baseline and our best model when evaluated on the test set from the hiring decisions. ** denotes significance with the randomisation test at $\alpha = 0.01$.

	NDCG@10	**MAP**	**P@5**	**P@10**
ES baseline	0.392	0.3376	0.207	0.1883
HIR1	**0.4498****	**0.4005****	**0.2502****	**0.2181****
	(+13%)	**(+14%)**	**(+21%)**	**(+16%)**

Because of the pooling process used to form the assessment set the baseline is less strong than on the hiring decisions.

Table 17: Comparison of the ElasticSearch baseline and our best model when evaluated on the relevance assessment set. * denotes significance at $\alpha = 0.05$).

	NDCG@10	**MAP**	**P@5**	**P@10**
ES baseline	0.212	0.2386	0.1737	0.1848
HIR1	**0.2644***	**0.273**	**0.2162**	**0.2101**
	(+25%)	**(+15%)**	**(+24%)**	**(+14%)**

While the NDCG@10 can also be significantly improved on the relevance assessment set with our model trained on the hiring decisions, the improvements on the other metrics are not statistically significant (at $\alpha = 0.05$ with randomisation tests). Looking at the performance of the model on individual queries in Table 18, we can see that while the re-ranking improves the precision of many queries, it also decreases the precision of some other queries. These results suggest that learning a re-ranking model based on hiring decisions is more effective on some queries, where the hiring decision labels might more accurately reflect the notion of suitability/relevance. Future research should focus on attempting to tease apart different "query types" such that

Table 18: The change of MAP and P@5 on the assessment set queries after re-ranking with the learned model.

MAP ↓	**stable MAP**	**MAP ↑**	**P@5 ↓**	**stable P@5**	**P@5 ↑**
36	9	54	21	41	37

re-ranking is only applied to queries for which the performance is more likely to increase.

5.4 ANALYSIS

In the following we report some additional results from our conducted experiments on the hiring decisions, which will give us an idea of how certain aspects of the data and the features affect retrieval performance.

5.4.1 *Datasets*

We are interested in how different datasets can affect the performance of the trained models. In particular, we want to assess the impact of assigning different labels, i. e. whether including the *interviewed* label leads to a more performant model (i. e. higher increase w.r.t. the baseline than in the *HR* set, which does not contain the *interviewed* label). Furthermore, the dataset *HIR1*, *HIR2* and *HIR3* contain increasingly more training data. However, in this case more data also means potentially more heterogenous data. Thus, it is not a priori clear whether the biggest dataset will also lead to the best performance as in many conventional machine learning problems.

In order to make the models comparable we use the same training algorithm as well as the same validation and test set for all models. Thus, all models will be tuned w.r.t. the same queries and they will be evaluated on the same set of queries. This means that the only remaining varying parameter is the training set. The algorithm we use for this experiment is LambdaMART optimising for NDCG@10 with the following parameters: #trees = 1000, #leaves = 10, minimum leaf support = 1, learning rate = 0.1, early stop = 100. Both the validation and the test set contain 628 queries.

Note that the search baseline is different for the *HR* dataset and the datasets containing the additional *interviewed* label despite being evaluated on the same set of queries. The reason for this is that a number of documents count as relevant in the *HIR* sets because the corresponding candidates got to the interview stage but were not hired. These candidates are considered non-relevant in the *HR* dataset. Thus, we cannot directly compare the absolute performance of the *HR* set with the *HIR* datasets. However, it is possible to interpret the increase in performance w.r.t. the corresponding baseline.

Table 19: Comparison of the models trained on different datasets. * and ** denote significance at $\alpha = 0.05$ and at $\alpha = 0.01$, respectively, w.r.t. the corresponding baseline.

dataset	**NDCG@10**	**MAP**	**P@5**	**P@10**
ES baseline	0.35	0.2707	0.104	0.0913
HR	**0.3771 (+8%)**	**0.2955* (+9%)**	**0.1183* (+14%)**	**0.0953 (+4%)**
ES baseline	0.3991	0.3507	0.2229	0.2005
HIR1	**0.4498** (+13%)**	**0.4005** (+14%)**	0.2502**	0.2181**
HIR2	0.4465**	0.3926**	**0.2506** (+12%)**	**0.2208** (+10%)**
HIR3	0.442**	0.3907**	0.2467*	0.2187**

The results in Table 19 suggest that the training does benefit from the additional label as all *HIR* datasets allow a model to be trained which performs better on most metrics than in the case of the *HR* dataset. This result confirms our intuition that a labelling purely based on hiring decisions may lose important information about candidates who were not hired but nevertheless should be considered relevant because of their qualifications or past work experience.

However, within the *HIR* datasets more (possibly heterogenous) data does not necessarily result in better retrieval performance. In fact, there is no significant difference between the model performance of the three models, though all three perform significantly better than the baseline (randomisation test with $\alpha = 0.01$).

Our conclusion from these experiments is that the learning algorithm does benefit from the additional *interviewed* label in the sense that it can learn a more performant model w.r.t. a baseline. However, having more data does not necessarily help the performance of the learned model if the data is not filtered according to some sensible criteria (cf. our filtering criteria in Section 5.1).

5.4.2 *Feature Sets*

Since there has not been any work investigating features in the domain of recruitment in the context of feature-oriented retrieval, we conduct a set of experiments focussing on the impact of different feature sets in terms of retrieval performance. We use one of our best performing datasets from Section 5.4.1, *HIR1*, for these experiments. As we are only comparing the differences of feature sets within one dataset, we use 5-fold cross-validation in order to be less dependent on one particular test set and report the averaged results of the 5 folds in Table 20. The LTR algorithm used in this section is LambdaMART optimising for NDCG.

It should not come as a surprise that the set of basic features contribute most to the retrieval performance, as they directly encode how

Table 20: An overview of the retrieval performance using different feature subsets as described in Section 5.2.4 on dataset *HIR1*.

feature sets (#features)	**NDCG@10**
baseline (no reranking)	0.3833
basic (26)	0.4065
basic + model-based (35)	0.4148
basic + doc-word-based (352)	0.4053
basic + query-word-based (183)	0.4119
basic + query-word-based + doc-word-based (509)	0.4089
basic + doc-word-based + model-based (361)	0.409
basic + query-word-based + model-based (192)	**0.4151**
all (518)	0.403

well certain fields match certain terms in the query. In comparison, the remaining sets add somewhat to the performance but have less impact. In these experiments the best combination of features w.r.t. NDCG is in fact the combination of the basic features, the model-based features and the query job title words (so not the combination of all features).

Despite the fact that the word-based features add little performance to the queries we are testing, it is important to keep in mind that we are testing a very special kind of queries which contains a lot of information directly from the text of job postings. Generally, however, when recruiters use search to issue their own manual queries, the queries are much shorter and less informative. For such situations most of the basic features might contain missing values so that the general-purpose word-based features become more important.

5.5 CONCLUSION

In this chapter we proposed several features (feature sets) for the LTR framework in the recruitment domain. In particular, we formulated features that can take advantage of off-line computed field relevance models introduced in Chapter 4, whose computation can do without any supervision (i. e. any kind of relevance labels). As an application we cast the matching/ranking problem of CVs as a common IR problem where relevance labels are represented by hiring decisions and demonstrated with a number of experiments that the retrieval performance can be improved within a LTR framework.

As a secondary evaluation measure we also tested our trained model on the relevance assessment set, which has not been included in the learning procedure. While some improvement was also possible on this set, we did notice some discrepancy between the assessment set and the hiring decisions: The learned model may improve the ranking performance on most queries in the assessments, yet the precision of a considerable set of queries dropped after re-ranking. This result suggest that while a general re-ranking model based on hiring decisions such as ours might improve performance over an already competitive baseline, an even larger improvement can be achieved if we are able to determine beforehand which queries are likely to benefit from re-ranking and which are not. This way, the re-ranking model can be plugged into the full retrieval system as an optional module, which only re-ranks suitable queries.

Our additional experiments give an idea about the impact of certain aspects of the data: While hiring decisions are helpful in the sense that they allow us to transform the matching problem into an IR problem with relevance labels and to use existing models and evaluation metrics, they are usually lacking information about relevant but rejected candidates. We show that having some additional information such as whether or not a candidate was interviewed can improve retrieval performance. Our results also suggest that more data does not automatically result in a better learned model if the additional data is likely to be more heterogenous.

Furthermore, we found that given our semi-structural setup (i. e. semi-structured documents where we know the semantics of certain fields), our best bet is to extract search scores for individual fields and use these scores as query-document level features. When these fields are rich (both in the document and in the query), they serve as a very good indication of a document's match toward a query. The remaining features add more to the retrieval performance, however, they have less impact than the "match features". Future research should investigate shorter, less informative queries (much more often produced by users) and what kind of features can be used in this case to improve ranking.

6 CONCLUSION

6.1 SUMMARY

In this thesis we outlined our problem domain and our problem setting in great detail, as our work builds on top of an existent commercial software package, which on the one hand relieved us of a number of lower-level tasks yet on the other hand imposed a contextual structure which our final system has to be plugged into. The model that we finally proposed and evaluated is inspired from a large body of previous research in IR, which we briefly introduced in the chapter of related work.

Our task was to learn a re-ranking model for lists of CVs as they are returned by a state-of-the-art search engine with respect to queries, where both the queries and the CVs (documents) are semi-structured. As a basis for learning a large set of hiring decisions was provided, i. e. a set of real-world job postings from different companies including the CVs of applicants for these jobs as well as information about the hiring decisions of recruiters. We described in this thesis how to mine and compute useful models from the set of CVs and demonstrated the application of these models in a feature-oriented learning-to-rank training procedure. The resulting re-ranking model can be applied to the search result list to re-score the retrieved documents, which results in a new ranking of the documents. When evaluated on the hiring decisions our model improved on all common IR metrics over the search baseline, indicating that people who were hired tend to be placed more on top of the list.

6.2 FUTURE WORK

To our knowledge, this thesis is the first work to tackle the matching/ranking task in the general domain of recruitment as an LTR task with hiring decisions as a proxy for relevance labels. In this line of research a lot more questions remain open. We give some suggestions for future research in the following.

- Investigate qualitatively for what kind of queries the re-ranking is likely to result in an improvement and design/learn an additional component that can make the decision whether or not to apply re-ranking or apply a adaptive weighing scheme to the search score and the re-ranking score.
- Generate a different kind of labels: Real-world e-recruitment systems can have the ability to accumulate a lot of clickthrough data of users. With a thorough analysis these data can be heuristically transformed into relevance labels, which in turn can feed into any LTR training procedure.
- Investigate the same matching/ranking problem with manual queries, which are usually much shorter and more general. Queries of this kind may require a completely different set of features in an LTR setting. Can a learning approach compete with search baseline?
- Given a vacancy parsing model that extracts relevant information from job postings, how to turn this information into a query as to maximise retrieval performance?

BIBLIOGRAPHY

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136. ACM, 2007.

Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

Fernando Diaz, Donald Metzler, and Sihem Amer-Yahia. Relevance and ranking in online dating systems. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 66–73. ACM, 2010.

Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *The Journal of machine learning research*, 4:933–969, 2003.

David Guthrie, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. A closer look at skip-gram modelling. In *Proceedings of the 5th international Conference on Language Resources and Evaluation (LREC-2006)*, pages 1–4, 2006.

Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Large margin rank boundaries for ordinal regression. *Advances in neural information processing systems*, pages 115–132, 1999.

Jin Young Kim and W Bruce Croft. A field relevance model for structured document retrieval. In *Advances in Information Retrieval*, pages 97–108. Springer, 2012.

Marios Kokkodis, Panagiotis Papadimitriou, and Panagiotis G. Ipeirotis. Hiring behavior models for online labor markets. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, pages 223–232. ACM, 2015.

John Lafferty and Chengxiang Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 111–119. ACM, 2001.

Victor Lavrenko and W Bruce Croft. Relevance based language models. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 120–127. ACM, 2001.

Hang Li. Learning to rank for information retrieval and natural language processing. *Synthesis Lectures on Human Language Technologies*, 7(3):1–121, 2014.

Jianhua Lin. Divergence measures based on the shannon entropy. *Information Theory, IEEE Transactions on*, 37(1):145–151, 1991.

Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

Jochen Malinowski, Tobias Keim, Oliver Wendt, and Tim Weitzel. Matching people and jobs: A bilateral recommendation approach.

In *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*, volume 6, pages 137c–137c. IEEE, 2006.

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

Sameep Mehta, Rakesh Pimplikar, Amit Singh, Lav R Varshney, and Karthik Visweswariah. Efficient multifaceted screening of job applicants. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 661–671. ACM, 2013.

David Mimno and Andrew McCallum. Modeling career path trajectories. Technical report, University of Massachusetts, Amherst, 2007.

Paul Ogilvie and Jamie Callan. Combining document representations for known-item search. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 143–150. ACM, 2003.

J. M. Ponte. *A language modeling approach to information retrieval*. PhD thesis, University of Massachusetts at Amherst, 1998.

Jay M Ponte and W Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281. ACM, 1998.

Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple bm25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 42–49. ACM, 2004.

Amit Singh, Catherine Rose, Karthik Visweswariah, Vijil Chenthamarakshan, and Nandakishore Kambhatla. Prospect: a system for screening candidates for recruitment. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 659–668. ACM, 2010.

Qiang Wu, Christopher JC Burges, Krysta M Svore, and Jianfeng Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, 2010.

Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391–398. ACM, 2007.

Xing Yi, James Allan, and W Bruce Croft. Matching resumes and jobs based on relevance models. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 809–810. ACM, 2007.

ChengXiang Zhai. Statistical language models for information retrieval. *Synthesis Lectures on Human Language Technologies*, 1(1):1–141, 2008.

Chengxiang Zhai and John Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 403–410. ACM, 2001.

Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214, 2004.