Universidad Autónoma de Madrid

Master's Thesis

A Language Model based Job Recommender

Author:
Carlos del Cacho

Supervisor:
Prof. Estrella Pulido

A thesis submitted in fulfilment of the requirements
for the degree of Masters in ICT Research and Innovation

in the

Departamento de Ingeniería Informática

May 2015

Declaration of Authorship

I, Carlos del Cacho, declare that this thesis titled, 'A Language Model based Job Recommender' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"If you torture the data long enough, it will confess to anything."

Ronald Coase

UNIVERSIDAD AUTONOMA DE MADRID

Abstract

Faculty Name
Departamento de Ingeniería Informática

Masters in ICT Research and Innovation

A Language Model based Job Recommender

by Carlos del Cacho

Matching candidates to job openings is a hard real world problem of economic interest that thus far defies researchers' attempts to tackle it. Collaborative filtering methods, which have proven to be highly effective in other domains, have a difficult time finding success when applied to Human Resources. Aside from the well known cold-start issue there are other problems specific to the recruitment world that explain the poor results attained. In particular, fresh job openings arrive all the time and they have relatively short expiration periods. In addition, there is a large volume of passive users who are not actively looking for a job, but that would consider a change if a suitable offer came their way. The two constraints combined suggest that content based models may be advantageous. Previous attempts to attack the problem have tried to infer relevance from a variety of sources. Indirect information captured from web server and search engine logs, as well as eliciting direct feedback from users or recruiters have all been polled and used to construct models. In contrast, this thesis departs from previous methods and tries to exploit resume databases as a primary source for relevance information, a rich resource that in my view remains greatly underutilized. Relevance models are adapted for the task at hand and a formulation is derived to model job transitions as a Markov process, with the justification being based on David Ricardo's principle of comparative advantage. Empirical results are compiled following the Cranfield benchmarking methodology and compared against several standard competing algorithms.

Acknowledgements

Juggling a job along with the attempt at gaining an education for the last year and a half has been challenging to say the least. As this work comes to an end, it is also the time of giving credit where credit is due. A warm thank you is well deserved by my advisor Estrella Pulido, who has been nagging me for as long as I remember to try to pursue further studies. Probably a bit past my prime I am rediscovering the joy of learning, a pleasure long forgotten. The good news is that this time around I have no intention to stop learning, and I declare myself a lifelong student. Thank you for your stubborness in pushing me towards graduate school.

I am very much obliged to my coworkers at Jobandtalent, who make work a fun and mostly gratifying experience. In particular, this work would have been impossible to carry out without the support of Teo Ruiz, our system administrator who dilligently cleans up the mess whenever I upload a buggy version to the production server. It has been great to enjoy the camaraderie of the people in the product development team, although listing them all would strain my neurons and alienate the ones whose names I might not recall on time. I like to entertain the thought, that behind all those numbers and charts that show improvements in our metrics there are actual people, and through more targeted recommendations I can get a chance to improve their lives, if only a little.

Finally, even though it may sound cliché, I would like to thank my friends and family for their support throughout all these years. When the time gets rough, close relatives are the people you can always count to depend on. I have had my share of bad experiences along the way, but the ride has been positive overall and you have a lot to say in that respect. Life is growth and I hope that we continue growing together.

Contents

List of Figures

List of Tables

Abbreviations

API	**A**pplication **P**rogramming **I**nterface
COTS	**C**ost **O**ff **T**he **S**helf
CTR	**C**lick **T**hrough **R**ate
CV	**C**urriculum **V**itae
GFS	**G**oogle **F**ile **S**ystem
HDFS	**H**adoop **D**istributed **F**ile **S**ystem
IDF	**I**nverse **D**ocument **F**requency
IR	**I**nformation **R**etrieval
JSON	**J**ava**S**cript **O**bject **N**otation
KLD	**K**ullback **L**eibler **D**ivergence
LM	**L**anguage **M**odel
OS	**O**perating **S**ystem
PRF	**P**seudo **R**elevance **F**eedback
PRP	**P**robability **R**anking **P**rinciple
REST	**RE**presentational **S**tate **T**ransfer
SOA	**S**ervice **O**riented **A**rchitecture
TF	**T**erm **F**requency
TTL	**T**ime **T**o **L**ive
URL	**U**niform **R**esource **L**ocator

Dedicated to my grandfather

Chapter 1

Introduction

During the past two decades we have witnessed a progressive transformation of the recruitment industry. Long past are the days of newspaper advertisements and resume mailings to target companies with the hopes of landing a better position. While personal connections still weigh heavily into the process, by 2000 it was estimated that about 20% of the vacancies were filled through online recruiting platforms [11], and this figure has been climbing steadily throughout the years, comprising 30% of hires in 2014 according to applicant tracking system vendor SilkRoad [82]. With 88% of job seekers now using some sort of online profile to aid their job hunt [32], it is safe to say that Internet recruiting is here to stay.

Coupled with this global trend, we have recently seen an uptick in the availability of tools and processing systems with the capacity to analyze large data sets. As much as the hype surrounding the expression "big data" is certainly exaggerated, there is a grain of truth to it. We live in a world where information overload manifests itself as prevalent as the common cold. It is in this context that analytics, properly harnessed, holds the potential to become a game changer across many industries, although many a stakeholder is left wondering at this point exactly how that change may come about.

One such sector ripe for disruption is that of Human Resources. Recruitment companies are sitting on a treasure trove of data waiting to be explored and we now have the analytic capacity to dive into it to extract and exploit meaningful patterns. Monster.com reportedly holds a database of an estimated 162 million resumes [58], whereas LinkedIn, the poster child of professional social media, recently crossed the 300 million user mark [62].

On the other side of the table, portals accumulate millions of active job offers at any given point in time. However, our capacity to absorb and interpret all this material

to sort the weed from the chaff remains rather limited. As a result, we routinely miss professional opportunities that could jump-start our careers and improve our personal living standards. For lack of time we fail to gather information and miss that perfect match that would suit our personalities as well as our wallets.

The matter is in fact important. Ever since the seminal work of Akerlof [2] we know that markets can fail to coordinate when information spreads inefficiently, with theories on wage formation considering search frictions taking a prominent role in labor economics [19, 67, 90]. Lowering transaction costs is then a worthy goal, and a recent study by Kroft [38] has shown that online platforms can play an active role in this regard through increased efficiency in matching.

If candidates lack the time to search on their own, it is advisable to design automated job recommender systems to carry out efficient information diffusion. This thesis presents a novel view on an old problem, that of matching a database of resumes to a database of job offers. The framework presented here lies at the heart of the recommendation engine at a well trafficked Spanish online job portal, and its development has been the source of much joy for its author.

The exposition is organized along several sections. Chapter 2 explores the existing literature on the subject and describes the main approaches that have been attempted to solve the problem, highlighting advantages as well as shortcomings when appropriate. Chapter 3 introduces probabilistic models in Information Retrieval. Chapter 4 explains evaluation metrics as the ones that have been applied in the experimental discussion. Chapter 5 explains the fundamentals of Big Data technology, focusing on the MapReduce framework, which has been used in this thesis for the phase of model building. Chapter 6 explains how a particular flavor of probabilistic retrieval model has been adapted to the task at hand, that of relevance models. Chapter 7 explains the architecture of the proposed system as has been implemented in a real job portal. Chapter 8 introduces empirical results and compares the framework with existing algorithms in a production system. Chapter 9 finally concludes with lessons learned throughout this project and hints towards guidelines for future lines of work.

Chapter 2

Literature Review

From a researcher's point of view, the topic of job matching technology presents a number of peculiarities. It may come as a surprise to the newcomer to the field that it is far easier to arrive at patented output than it is to find papers on respectable journals when performing an exploratory background check. This has two likely underlying causes worthy of mention.

The obvious one to consider is that the area, being applied in nature, is of commercial interest and has wide market impact. As it stands, this is an unsolved problem. Qualitatively judging the recommendations I hold in my email inbox, we are very far from achieving success at present. Thus any improvement is quickly understood as a competitive advantage and people first rush to the Patent Office as opposed to directly walking the traditional academic route. There may also be an advantage to withholding information from competitor's prying eyes, even if this practice places obstacles to collective advancement.

Likewise, the difficulty of accessing a properly sized data set to test with is not to be underestimated. Some methods, as the one proposed in this thesis, require huge amounts of data merely to operate. It is safe to say that this barrier acts as a deterrent for researchers confined to the walls of university departments, limiting the number of publications available.

Given all the above, to provide an adequate overview of the field, it is deemed appropriate to dwell a little bit on the patents as well as on published papers.

2.1 Introduction

The term recommender system as such was coined by Resnick and Varian [73], and it was used to describe automated assistance tools for decision making in the absence of information by the decision makers. Although the development of such systems predates the web, it was the burgeoning growth of the Internet and in particular the extension of e-commerce sites that attracted attention to the design of such systems, with the generation of up-sell and cross sell opportunities as a revenue driver for online retailers.

Traditionally we can split recommender systems along several main sub types:

- Collaborative filtering: These systems rely purely on capturing user behavior over time and making recommendations based on patterns of activity, either clicks, purchases or ratings. The idea was pioneered by Goldberg [23], with the underlying notion being that users with similar behavioral traces will overlap in tastes for unseen items [81]. There are many different ways to generate recommendations under this framework, from nearest neighbor approaches [18] to matrix factorization methods [37], with the latter being the state of the art at the time of writing. These systems typically suffer from a catch-22 or cold start problem, as with a freshly minted implementation new users and items in the system do not have predecessors on which to build upon for the initial recommendations. The power of collaborative filtering lies in the fact that it is data agnostic, thus its applicability is very horizontal in scope. As such, it has found usage with few modifications in areas as diverse as recommending news [36], books [12], music [97], and movies, etc.

- Content based: Rather than focusing on other users' actions on specific items, which is the standard collaborative filtering framework, these systems define similarity metrics based on features and either match the user with products or services close to the ones already bought according to these metrics, or directly match a user profile with a given service based on content [59, 66]. They typically approach the problem with a Machine Learning perspective in mind and are often tailored to solving a particular problem. In this sense flexibility is lost, but a specific solution exploiting knowledge of a particular domain can yield benefits over a generic one. Proper feature extraction and feature engineering as well as the formulation of the problem determine the success of the system. The proposed job recommender in this thesis falls under this category.

- Hybrid systems: As expected, they pretend to combine the best of both worlds, with varying degrees of success. Their mode of operation is to include both dynamic user behavior and individual features engineered on the content of the items

they are trying to recommend [6]. This helps solve the cold start problem while maintaining the desirable qualities of adapting to user behavior over time.

Most of the recommender systems in production to date employ at least in part the dynamic strategies of collaborative filtering, and these have been the focus of intensive work. The good standing of online retailers such as Amazon and Netflix speaks volumes for the feasibility of this method [45]. Nevertheless, job recommenders must wrestle with special requirements that seem to indicate that a content based model as per traditional Information Retrieval may hold particular advantages.

2.2 Recommender systems in Human Resources

2.2.1 Business constraints

At any given point in time, a large job portal holds several million job openings outstanding, with countless others as historical records. Furthermore, given that the openings themselves are short lived, the probability that any two users apply to the same offer is so slim that the system would be unable to generate recommendations for a majority of passive candidates that are not currently looking if it were to rely on collaborative filtering alone [72].

Another pertinent business constraint that complicates matters is that while a profit maximizing retailer does not care about the diversity of the recommendations so long as they increase revenue and can afford to concentrate sales on a subset of its product offering, a job portal has to consider the effect of the lack of diversity on customer satisfaction of all employers and give chances to receive applications in a more even setting. Likewise, the marginal value of an application in an oversubscribed job opening is much lower both for the receiving company as a result of satiation, and for the candidate on the basis of increased competition, than when a job opening has a relatively small number of applications. As a result, under-subscribed openings must be promoted more aggressively, and these are precisely the ones where dynamic data is more scarce and the cold-start issue is at play.

Finally, a major objection that has been raised in the present context is that collaborative filtering methods only consider the preference of the candidate while the hiring managers probably also have a say in the matter [50]. Ignoring the preferences of the employer in emitting recommendations would clutter the inbox of the recruiter with unqualified candidates, that in their eagerness to get a better job, introduce noise and obstruct

the matching efficiency of a one sided dynamic system. This issue has given rise to automated screening methods after the application has taken place [9, 53, 83].

2.2.2 Collaborative Filtering methods

Despite the difficulties that have been mentioned in applying collaborative filtering methods in Human Resources, some researchers have still tried to bridge the gap. To confront the short expiration periods, application of collaborative filtering strategies in the recruiting space first need to derive measures of similarity among users or job offers to reduce dimensionality as otherwise the match probability is likely to be low.

Expressions of preference can be either explicit, where a user is asked whether he likes a job or not, or implicit, where the system tries to infer the user's preferences through profiling behavior in real time. The second option is often preferred, if only because there is much more implicit data available. Several attempts have intended to exploit information revealed from mining server logs. Rafter used page views along with read times as a measure of implicit preference [72]. AlJadda et al [3] have recently extended the idea to increase match probability among users by using keyword patterns in search engine logs in order to define a distance metric among users, after which regular collaborative was applied.

2.2.3 Content Based Filtering methods

However, given the aforementioned problems, authors of job recommender systems often prefer to focus their efforts on content based models. One of the early works in this domain was the project developed by Vega for the newspaper Le Monde, dating back to 1990 [95]. The system worked by trying to match requirements with NLP techniques along with a general knowledge base and distinguished between compulsory and optional requirements.

Matching skills has been an avenue thoroughly explored, as it is a common mode of operation used by human recruiters, and it is natural to try to replicate in an automated way what people already do. William proposed a method for entering both skills and job requirements into a properly defined matrix format that would permit efficient computerized matching [100]. Anderson et al [5] offered a slight variation on this theme. In addition to detailed skill accounts, prospective employees purportedly would fill out an approximate figure of how much time they spent exercising each skill at their job. This was later used to derive measures of skill recency and total amount of experience while performing matching.

Given regular user behavior, it is unlikely that time constrained candidates would comply with such stringent obligations rather than losing patience and switching to the next job portal. Ideally, a content based recommender should be able to deal with noisy input in free text format, the very same one being handed to regular recruiters. In addition, as human nature is fallible, it is rarely the case that both employers and employees are exhaustive enough for such an approach to work.

Other less common methods in the skill domain involve applying semantic web technology in the form of Human Resources ontologies [7, 20, 93]. They employ standardized skill databases and job titles to aid in the matching function, although the empirical results are not presented in a formal manner and compared with regular IR techniques, therefore it is difficult to foretell exactly how well they fare at present. In addition, maintenance of the ontology is a difficult and time consuming task and this fact hinders widespread applicability of the proposed methods. Xing, a professional networking website with over 14M subscribers, is reported to have recently adopted ontologies in their job matching algorithm. The work has been commissioned to the Spanish start-up Playence [68]. As cumbersome and costly ontology engineering may be, the idea still holds some merit and warrants careful consideration.

Other authors, realizing the limitations in matching skills versus requirements in an explicit form, have tried to develop automated means of obtaining this information from resumes. Inn's method [26] appears as more interesting. He proposed an automatic extractor based on word patterns with the goal of automatically classifying a candidate into several predefined buckets or categories, later to be used for matching jobs within the given category under consideration.

Proper care must be exercised when implementing such a process, as relying on word patterns without taking into account the temporal evolution of the candidate is bound to fail. Consider for instance the case of someone who has worked for a decade at several jobs as an individual contributor and finally attains the desired promotion. Vanilla word matching would perhaps misclassify the individual and his future intentions, as he is unlikely to want to go back to his previous job now that he is earning a higher salary. This is a common problem in many IR methods being applied to the present domain, one that the present work has tried to overcome.

A more formal approach that points in the automation direction is given by Hyder et al [25], from Monster Worldwide. Statistics about how employers and candidates use the web site are accurately and timely recorded. Of particular importance is the information about which candidates an employer finds interesting for a given job opening. This is a crucial point, as candidates may apply imprecisely to a set of openings that are mildly relevant to their skillset, whereas companies are much more selective in their judgments.

The resumes of these top candidates are later correlated in an unspecified way with the job offer and similar candidates to the ones already tagged are then searched for within the database. The scope of the patent is deliberately broad and would apparently target any implementation loosely connected to Learning to Rank methods [46], which apply a Machine Learning perspective to sorting documents.

The issue has also attracted attention by some of the leading research groups in Information Retrieval. Yi et al. [102] collaborated with Monster.com in an attempt to introduce relevance models [43] for job matching. Relevance models are an extension to probabilistic language models that define a proper way to approach relevance purely from a statistical perspective. The idea is to construct an initial search with a traditional probabilistic algorithm or any other means, and assume that the top documents returned are relevant (pseudo-relevance feedback or blind feedback [101]). Then a second round of model estimation is performed through maximum likelihood over this initial set of documents, and with the new model in hand the search is further refined, ranking the results by means of negative cross entropy or the well known probability ratio.

As stated, in regular relevance models the seed documents are arrived at through pseudo relevance feedback. In contrast, Yi's group used a properly curated data set with relevant documents being hand picked by human agents. Furthermore, their approach differed from regular document ranking in that they incorporated some elements of structure within the document in their models. In particular, they performed an initial query across equivalent fields in the job offer to build a model with an expanded vocabulary set. This method resulted in a noticeable increase in precision over plain relevance models and has been later formalized [35] and built into Indri, a leading open source search engine.

Chapter 3

Probabilistic Information Retrieval

3.1 Historical roots

Ever since the second half of the XIX-th century, librarians have striven to categorize documents along exhaustive topical hierarchies, the most notable being the Dewey classification scheme, which has earned widespread adoption across the world. The switch towards keyword indexing can be traced back to the pioneering work of Taube, a librarian himself, who was responsible for the development of the Uniterm system [92]. The model indexed a list of documents by the words that they contained, rather than trying to fit the collection into a standardized taxonomy. Uniterm's initial success gained momentum when Cleverdon developed the Cranfield methodology for algorithm benchmarking, where it stood head and shoulders above its competitors, proving that word indexing could become a viable alternative to hand tailored categorization [13, 14]. The system, though rudimentary by today's standards, paved the way for future developments in the nascent field of Information Retrieval.

Initial computerized methods employed simple Boolean keyword matching, retrieving documents that contained the keywords in the query, without further consideration. A step forward was given by Luhn, who proposed assigning a score to each item according to the the relevance relative to the terms of the query [48]. Maron, Kuhns and Ray went one step further and implemented a system following Luhn's ideas by giving different weights to the words to arrive at a final score prior to ranking [51]. Keyword importance was manually assigned by the researchers themselves, and their approach outperformed Boolean search by a wide margin in their experiments and was later to become mainstream. A major advancement was the introduction of the vector representation for

documents. The approach had been originally developed by Switzer [91], although it gained momentum with Salton's vector space model. Under this scheme, documents were seen as a multidimensional vector of term frequencies (TF), typically scaled by Sparck Jones' inverse document frequency correction (IDF), which attempted to control for the presence of content bearing rare words, as otherwise their weight relative to common words would lose effect [80] [86]. The method allowed to sidestep manual keyword tagging and permitted weighing keywords automatically. The final score was obtained either through a cross product similarity function or by measuring the cosine of the angle among TF-IDF vectors, where the individual constituents are obtained as shown in eqs. (3.1.1) to (3.1.2) in one of its multiple incarnations. The method reigned undisputed for quite a while and is even the focus of attention up to the present day.

$$tf - idf(w_i, d, C) = tf(w_i, d) idf(w_i, C) \tag{3.1.1}$$

$$tf(w_i, d) = \frac{freq(w_i, d)}{max\{freq(w_i, d) : w_i \in D\}} \tag{3.1.2}$$

$$idf(w_i, C) = log \frac{|C|}{|d \in C : w_i \in d|} \tag{3.1.3}$$

3.2 The Probability Ranking Principle

TF-IDF was originally informal and ad-hoc, although subsequent researchers have found theoretical underpinnings that appear to justify its usage [75]. Initial attempts to model document retrieval under a more rigorous framework can be first seen in the early works of Cooper [15], Robertson [77], and Van Rijsbergen [94], who in his influential textbook defined the probability ranking principle, as follows:

> "If a reference retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data."

This guideline ignores other important problems in retrieval such as ensuring that a diverse result set is returned. It can be argued that the likelihood of relevance of an unreturned document is affected by the already returned documents in so much as it can contain duplicate information of no interest to the person that posited the query. Variations have been recently introduced that try to address these shortcomings [98]

[103]. Nonetheless, the idea has wide applicability in general and it has been a fertile ground for experimentation. The concept can be summarized in what Sparck Jones called the Basic Question: What is the probability that this document is relevant to this specific query? [87]

What is interesting is that the inquiry lends itself to Bayesian classification principles from the field of Machine Learning, more precisely, we can rank according to the Maximum A Posteriori rule. In principle, if we want to know whether a document is relevant or not, what we are interested on is in finding out $P(R|D)$ and $P(\hat{R}|D)$, with the document being classified as relevant or irrelevant depending on which of the two is higher. Assuming $P(R)$ is the a priori probability of a document being relevant, the application of Bayes rule provides the answer:

$$P(R|D) = \frac{P(D|R)P(R)}{P(D)} \tag{3.2.1}$$

$P(\hat{R}|D)$ can be obtained in a similar fashion. Because P(D) is constant in both cases, the decision rule is finally to classify the document as relevant if $P(D|R)P(R) > P(D|\hat{R})P(\hat{R})$. Since the a priori probabilities are constant as well, for ranking purposes they can be ignored. The ranking is theoretically optimal if we are able to sort documents according the probability ratio, as given by eq. (3.2.2):

$$Score = \frac{P(D|R)}{P(D|\hat{R})} \tag{3.2.2}$$

The expression bears some sort of resemblance to a regular odds ratio although, in this case, no conclusion can be gained about the relative weights of $P(R|D)$ and $P(\hat{R}|D)$ because we ignore the priors. This observation implies that no relevance judgment call can be made from the score alone. As previously stated, when interpreting the probability ranking principle formulation (PRP), it is customary to sample words assuming conditional independence among them, taking a logarithm to avoid truncation errors while dealing with small quantities. It is also often the case that words not appearing in the document are excluded from further consideration, treating documents as a sequence of words. Other methods such as the Binary Independence Model that will be presented later, try to enforce this constraint. Therefore, bearing in mind the simplifications just mentioned, we can finally spell out the previous formula as in eq. (3.2.3), with the terms w_i being the words present in the document:

$$Score = \sum_{w_i \in D} log \frac{P(w_i|R)}{P(w_i|\hat{R})} \tag{3.2.3}$$

As it stands, the summation in eq. (3.2.3) tends to favor long documents. Having two documents A and B, with B containing the content of A repeated twice, B will outrank A. This may not be important when all documents are more or less of the same length, otherwise it tends to bias the rankings. Fair retrieval of documents of different lengths under the PRP is an item that has received a great amount of attention.

A simple normalized version of the PRP that was originally proposed for the task of topic tracking [89], where it is important that scores be comparable across queries, is given by dividing the previous score by the document length $|D|$ as in eq. (3.2.4), although more formal methods that require empirical tuning give better results.

$$Score = \sum_{w_i \in D} \frac{1}{|D|} log \frac{P(w_i|R)}{P(w_i|\hat{R})} \tag{3.2.4}$$

One angle to consider is that the ratio of length normalized probabilities can also be seen as a difference between cross-entropies, which in a sense gives justification to new ranking methods that attempt to measure divergence between the word probability in the document and the background prior by means of information theoretic concepts, showing that the PRP formulation is not far removed from them [4]:

$$Score = \sum_{w_i} P(w_i|D) log P(w_i|R) - \sum_{w_i} P(w_i|D) log P(w_i|\hat{R}) = H(D,R) - H(D,\hat{R}) \tag{3.2.5}$$

Several statistical approaches have been proposed as interpretations for arriving at $P(D|R)$ and $P(D|\hat{R})$, with varying assumptions and constraints.

3.3 Negative Cross Entropy

An alternative for ranking that is more commonly used than the PRP is the one proposed by Lafferty and Zhai [40]. In Bayesian decision theory, an optimal classifier that minimizes misclassification risk can be said to be obtained if it minimizes the posterior expected value of a loss function $L(\theta, a)$, which can be interpreted as a form of negative utility:

$$\int L(\theta, a) p(\theta|x) d\theta \tag{3.3.1}$$

The retrieval model under risk minimization accommodates other probabilistic models that will be presented later as special cases. Perhaps more interestingly, it points to new ranking functions as well. Using the Kullback-Leibler divergence (KLD) as a loss function and under certain approximations, the ranking can be reduced to eq. (3.3.2), where a relevance model in relation to the query θ_q is compared to the model θ_D for each document as estimated by maximum likelihood:

$$Score = \sum_{w_i \in V} P(w_i|\theta_q) log P(w_i|\theta_D) \tag{3.3.2}$$

In the absence of a proper relevance model for the query, a maximum likelihood estimate from the explicit query can be used. The comparison metric is a similarity measure among probability distributions and it can be either seen as a form of negative cross entropy or the Kulback-Leibler divergence among the two probability distributions, both formulations being equivalent in their respective rankings. More specifically, cross entropy between p and q takes a minimum at H(p) when both distributions are equal whereas KLD has its own at zero, as can be garnered from the formula below:

$$H(p, q) = H(p) + D_{KL}(p||q) \tag{3.3.3}$$

Because Kulback-Leibler divergence is not symmetric, the order of the probability distributions when using cross-entropy affects the results. $P(w|D)$ is typically denoised using smoothing as well. The formula as expressed above involves summing over all terms in the vocabulary. However, this can be sped up over a naive interpretation. For practical purposes, given that the exact cross entropy result is not needed, we focus on the contribution on entropy of the individual terms appearing in the document and take the differential with respect to their absence. This makes the complexity of the computation linear on the terms in the document as opposed to linear in the size of the vocabulary, that can be orders of magnitude larger than the former.

Let $P(w_i|D, w_i \in D)$ be the smoothed probability of word w_i in the document given that it is present, and $P(w_i|D, w_i \notin D)$ the same probability if the word does not appear in the document. Then we can write the following:

$$\begin{aligned} Score = \sum_{w_i \in V} P(w_i|R) log P(w_i|D, w_i \in D) + \\ \sum_{w_i \in D} P(w_i|R)(log P(w_i|D, w_i \in D) - log P(w_i|D, w_i \notin D)) \end{aligned} \tag{3.3.4}$$

When using Jelinek-Mercer smoothing (see section 3.4.5.2), the term on the left is constant and can be interpreted as the cross entropy of the relevance model with the empty document. Since it does not depend on the actual document being scored, it can be safely ignored in a rank preserving transformation that substantially simplifies the calculation. For compactness, the final formula can be rewritten as:

$$Score = \sum_{w_i \in D} P(w_i|R) log \frac{P(w_i|D, w_i \in D)}{P(w_i|D, w_i \notin D)} \tag{3.3.5}$$

A different method that is often used in practical implementations consists of approximating the result by incorporating in the summation the terms with the highest probability and that contribute the most to the divergence metric. This improves run time performance for a slight reduction in theoretical accuracy.

Cross entropy typically performs better than the PRP for ranking. This may come as a surprise because the PRP is theoretically optimal, but the fact is that the probabilities obtained through maximum likelihood are subject to estimation risk. The performance degradation lies both in overconfident probability estimates and in the strong independence assumption that is done in multinomial sampling, which tends to bias the results. From personal experience, both methods tend to give sensible results though in different ways, with PRP being very sensitive to the inclusion of infrequent words in the collection into the relevance model. This observation lends itself to refinements that will be discussed later on.

3.4 Probabilistic Retrieval Models

3.4.1 The Binary Independence Model

This model was defined by Robertson and Sparck Jones [78] and approaches documents as binary feature vectors stemming from a Bernoulli distribution, where the words w_i can be either present or absent, ignoring the frequency of appearance. Additionally it assumes conditional independence among them, a constraint common in probabilistic models of this short because it simplifies the calculations.

Let p_i denote $P(w_i|R)$ and s_i denote $P(w_i|\hat{R})$. With this notation in hand, and assuming binary features, we can write out the likelihood ratio as follows:

$$\frac{P(D|R)}{P(D|\hat{R})} = \prod_{w_i \in D} \frac{p_i}{s_i} \prod_{w_i \notin D} \frac{(1-p_i)}{(1-s_i)} \tag{3.4.1}$$

This is equivalent to introducing a term that cancels out to one and does not affect the product:

$$\frac{P(D|R)}{P(D|\hat{R})} = \prod_{w_i \in D} \frac{p_i}{s_i} (\prod_{w_i \in D} \frac{(1-s_i)}{(1-p_i)} \prod_{w_i \in D} \frac{(1-p_i)}{(1-s_i)}) \prod_{w_i \notin D} \frac{(1-p_i)}{(1-s_i)} \tag{3.4.2}$$

Regrouping the terms:

$$\frac{P(D|R)}{P(D|\hat{R})} = \prod_{w_i \in D} \frac{p_i(1-s_i)}{s_i(1-p_i)} \prod_{w_i \in V} \frac{(1-p_i)}{(1-s_i)} \tag{3.4.3}$$

Because the second term does not depend on the document, it can be ignored in ranking without affecting the final document ordering. It is customary to take sums of logarithms instead of straight products because otherwise there can be truncation errors due to the probabilities being very small. In the absence of other information, p_i is taken as 0.5, because a term is equally likely to appear on relevant and irrelevant documents. On the other hand, the different s_i can be approximated by the respective word background priors in the collection, as for any given query the a priori probability of a document belonging to the relevant set is practically zero:

$$P(w|C) = P(R)P(w|R) + P(\bar{R})P(w|\bar{R}) \approx P(w|\bar{R}) \tag{3.4.4}$$

Taking a log transform the final ranking formula for the BIR model is given by:

$$Score = \sum_{w_i \in D} log \frac{0.5(1-\frac{n_i}{N})}{\frac{n_i}{N}(1-0.5)} = \sum_{w_i \in D} log \frac{N-n_i}{n_i} \tag{3.4.5}$$

In the formula above, N represents the number of documents in the collection and n_i is the number of documents containing the word w_i. Observe that when $N >> n_i$ this is a term similar to Sparck Jones' IDF. TF in this case has been dropped because we are dealing with binary feature vectors.

The initial ranking can be improved through relevance feedback, where the user marks some documents as relevant. A first ranking is obtained applying the formulas above, and then p_i is estimated as $\frac{r_i}{R}$, where r_i is the number of documents in the relevant set that contain word w_i and R is the number of relevant documents being taken into account. s_i is given by $\frac{n_i - s_i}{N-R}$ or the number of irrelevant documents that contain the term divided by the total number of irrelevant documents.

$$Score = \sum_{w_i \in D} log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \tag{3.4.6}$$

The Binary Independence Model, chiefly because it ignores term frequency, is not a very competitive retrieval model, lagging behind TF-IDF in performance. Nonetheless, the initial work by Robertson and Sparck Jones marked a starting point for future developments and more successful models.

3.4.2 The 2-Poisson Model

As an alternative to the Bernoulli formulation, terms can be assumed to arrive over time occupying certain slots or positions in the document. Then they are amenable to be approximated by a Poisson distribution: $P(f_{d,t}) \approx \frac{\lambda^k}{k!} e^{-\lambda}$. One problem that appears when fitting term frequencies according to a single Poisson distribution with a certain λ parameter is that words pertaining to the topic being searched for appear much more frequently than expected in the result set than in the collection as a whole. Harter's model [24] assumes they belong to an elite set with a different frequency of arrival and interpolates between the two, where π is a parameter that represents the probability that a given word belongs to the elite set:

$$P(f_{d,t} = f) = \pi \frac{\lambda^k}{k!} e^{-\lambda} + (1 - \pi) \frac{\mu^k}{k!} e^{-\mu} \tag{3.4.7}$$

Eliteness is not the same concept as relevance, therefore we can express the probability that a term appears with a certain frequency in the relevant document set by summing over all the possible states of eliteness:

$$P(f_{d,t} = f|R) = P(f|E)P(E|R) + P(f|\hat{E})P(\hat{E}|R) \tag{3.4.8}$$

$$P(f_{d,t} = f|\hat{R}) = P(f|E)P(E|\hat{R}) + P(f|\hat{E})P(\hat{E}|\hat{R}) \tag{3.4.9}$$

Given these probabilities and suitable estimates for the different parameters involved, ranking can be performed by means of the PRP.

3.4.3 BM25

One problem with the 2-Poisson model is that it is difficult to attain a reliable estimate for $P(E|R)$ and $P(E|\hat{R})$. A successful approximation is given by Robertson and Walker,

that, unlike the Binary Independence Model, tries to capture the nuances introduced by modeling term frequency. In a seminal paper, they proposed several methods for attacking the problem in the 2-Poisson model, deriving a very competitive model known as BM25, which was the basis for the Okapi search engine [79]:

$$Score = \sum_{w_i \in D} log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \frac{(k_1 + 1) f_i}{k_1((1 - b) + b(L_d/L_{avg})) + f_i} \tag{3.4.10}$$

In this context, f_i is the frequency of word w_i in the document, and k_1 and b are constants whose values are arrived at through empirical means. b oscillates between 0 and 1 and defines the degree of document length normalization that is applied, whereas k_1 limits the effect of increasing term mentions in the document. The model can be seen as a variant of the vector space model, with a sub linear term frequency saturation term and pivoted length normalization to penalize long documents. The original formula also contained a frequency saturation term for the terms in the query, although it is customary to leave out that modification, as most queries are typically short and repetitions are infrequent. The formula above is thus a simplification of the one proposed by Robertson.

There exist several variations of BM25 for various purposes. One of the most successful is BM25F [76], which includes document structure and gives different weights to specific fields when ranking. This lends itself to the practice of field boosting, common to tweak search results according to the perceived importance of different sections. For example, the title of a web site may be very informative of the topic addressed in the document, and it would be a good idea to give it more weight than the one given to other words appearing inside the document. Another recent variant, BM25+, attempts to correct for the overpenalization of long documents by adding a small constant to the term frequency [49]. It has shown slight improvements in empirical tests over the original BM25 formula.

3.4.4 Divergence from Randomness

This is a family of models that generalize Harter's two Poisson model. Harter measured the statistical difference of occurrence of terms among an elite document set and the remaining ones in the collection, which can be assumed to follow a background prior.

The idea that is exploited for ranking is that the more informative a term is, the more it should be differentiated from what is expected to result purely by chance. Then we can measure the cross-entropy of the document with this background distribution:

$$Score = \sum_{w_i \in D} -P(w_i|D) log P(w_i|C) \tag{3.4.11}$$

Various models differ in the way they come up with these probabilities, optionally introducing normalization terms. Other than the simple Binomial model, the idea can be attacked by using the Bose-Einstein distribution or the G approximation to the Bose-Einstein distribution. For a formal treatment of the topic, please see Amati and Rijsbergen [4].

3.4.5 Language Models

In contrast, a slightly different path is taken by generative models. In principle, these move away from the concept of relevance and borrow the idea of language models from the field of Natural Language Processing. These methods have in common that words are assumed to be sampled independently either from a multinomial or a multiple Bernoulli distribution, and they differ in the way they arrive at their estimates for this distribution that is used in ranking. Once the distribution is known, the idea is to estimate the probability that the documents to be ranked come from this empirical distribution, or rather they are merely compared through information theoretic means.

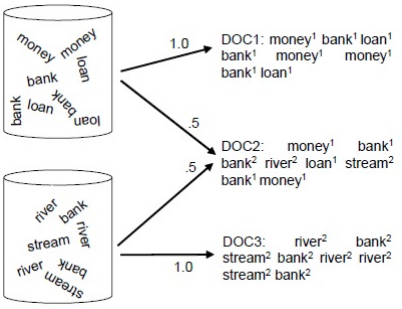

FIGURE 3.1: Language modeling's generative process

If we were to ascribe a language model M_q to the query that is given as an input, we could rank documents according to decreasing probability $P(D|M_q)$ of the document being generated by this language model.

Language models themselves are estimated directly through maximum likelihood. But given that the query itself is short, it is difficult to come up with a reliable approximation

for M_q directly. In addition, documents vary in length, which further complicates the mathematical treatment. Ponte and Croft [70] applied Bayes theorem and proposed to perform the task the other way around, in what is called the query likelihood or LM model:

$$P(D|q) = \frac{P(q|D)P(D)}{P(q)} \approx \frac{P(q|M_d)P(M_d)}{P(q)} \propto P(q|M_d) \tag{3.4.12}$$

In query likelihood, the probability of the query itself is irrelevant because the query is taken as given as it has already occurred and the term is constant for all documents. In addition, the prior $P(M_d)$ is assumed to be uniform among all possible document models. This term, again being constant, can be safely discarded. The advantage with this formulation is that documents are typically longer than the query and the estimates are more stable.

The original formulation by Ponte and Croft tries to estimate both the appearance of positive and negative appearances of words in the query, following the multiple Bernoulli model of retrieval:

$$p(q|M_d) = \prod_{w \in q} P(w|M_d) \prod_{w \notin q} (1 - P(w|M_d)) \tag{3.4.13}$$

However, most subsequent developments dropped the rightmost productory and treated the query as a sequence of words instead under multinomial sampling [54, 84].

Although empirical results favored the query likelihood model over other probabilistic models, it has been criticized for its lack of theoretical foundation, spurring debate among researchers in the field [85], because it departed from the well regarded probability ranking principle. Nonetheless, it can be reconciled with the PRP under a different factorization than the Sparck-Jones model. A formal derivation and theoretical justification was later given by Lafferty [41].

3.4.5.1 Estimating Language Models from data

Given a word w in the vocabulary and a document d, its maximum likelihood approximation under multinomial sampling is obtained by counting its frequency of appearance and dividing by the document length. This is the estimate that gives the maximum probability of generating the document, any other choice of parameters would give it a lower probability of appearance.

$$P(w|M_d) = \frac{freq_{w,d}}{|d|} \tag{3.4.14}$$

The estimate, being theoretically sound, is not directly employed, as words not appearing in the document would be given a probability of zero, biasing the rankings by dragging the final probability to zero. It is customary to smooth out the calculation against a background a priori probability. There are several ways to perform this task, with the most popular ones being discussed in sections 3.4.5.2 and 3.4.5.3.

3.4.5.2 Jelinek-Mercer smoothing

Jelinek-Mercer smoothing is simply fixed weight interpolation between the observed probability in the document and the collection a priori probability. The strength of the method relies on its simplicity and on the fact that it is very aggressive when not much evidence is available. Far from being a hindrance, this is a nice property in Information Retrieval, as placing good results on top of the ranking (precision) is a very important metric of success. In most cases of practical interest it is more important that the top documents are relevant rather than all relevant documents are retrieved (recall). Reasonable values for λ are in the range of 0.6 - 0.7 for long queries, with values around 0.1 being more effective in short ones. This is problem dependent and may vary depending on the nature of the collection being searched.

$$P(w|M_d) = \frac{(1-\lambda)freq_{w,d}}{|D|} + \lambda P(w|C) \tag{3.4.15}$$

3.4.5.3 Dirichlet smoothing

Dirichlet smoothing approaches the issue from a Bayesian setting, and incorporates pseudo-counts to the observed maximum likelihood estimates that drag the probability towards the a priori background probability. It is considered as more robust from an statistical perspective and usually gives better results in Information Retrieval. The formula is given by:

$$P(w|M_d) = \frac{freq_{w,d} + \mu P(w|C)}{|D| + \mu} \tag{3.4.16}$$

3.4.6 Relevance Models

Other than direct application of Bayes rule, another way to approach the problem of inconsistent maximum likelihood estimates is that of relevance models developed by Lavrenko and Croft [42], that at least in its simpler formulation, can be coarsely summed up as model averaging. An initial query is performed against a collection of documents. The top n documents are retrieved and assumed to be relevant, as in blind feedback or pseudo relevance feedback. Then an individual model is estimated for each and every single one of them and these models are combined to generate a relevance model for the query. Finally, a second search is performed with this model in hand, ranking documents by probabilistic means.

The crucial observation is that of providing a formal probabilistic framework to model relevance in a pseudo relevance feedback search, a task that had eluded previous researchers. What we would want to do is to be able to estimate the probability of occurrence of each word in the relevant document set, in order to use it for ranking documents.

Our best bet, given that we ignore the set constituents, is simply to assume that the relevant documents have generated the query itself and use that as a proxy for the true relevant set:

$$P(w|R) \approx P(w|q_1....q_k) \tag{3.4.17}$$

Then, application of basic probability theory dictates:

$$P(w|q_1....q_k) = \frac{P(w, q_1....q_k)}{P(q_1....q_k)} \tag{3.4.18}$$

Lavrenko proposes two methods for coming up with the joint probability in the numerator, whose differences stem from the assumptions made with regards to the independence of the random variables involved. In what he calls Method 1 (RM1), a strong independence assumption is put forth following Naive Bayes (see fig. 3.2):

$$P(w, q_1....q_k) = \sum_M P(M)P(w|M)\prod_i P(q_i|M) \tag{3.4.19}$$

The words being sampled are assumed to be independent once the model is fixed. Obtaining the desired joint probability is then a matter of marginalizing over the model universe in the relevant set, where we restrict it to enclose just the top n documents

obtained in the relevance feedback round, with the paper suggesting a value of 50 being reasonable according to empirical tests in the TREC tracks.

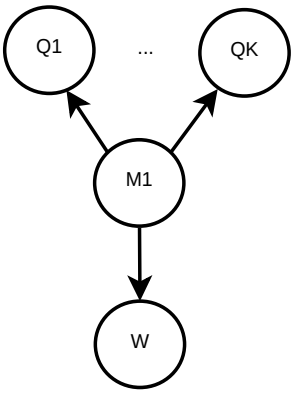

FIGURE 3.2: Bayesian network for Method 1

Another way to estimate the joint probabilities is to relax the aforementioned independence assumption, as in Method 2 (RM2). In this case we keep the notion that query words are sampled independently from each other but they are still dependent on the words appearing in the documents. The Bayesian network that represents its dependencies is given in fig. 3.3. Method two is slightly more computationally expensive while it yields a slight improvement in accuracy as measured by cross-entropy on a curated data set.

The factorization to obtain the desired joint probability is given by the formula in eq. (3.4.20):

$$P(w, q_1....q_k) = P(w) \prod P(q_i|w) \tag{3.4.20}$$

The individual $P(q_i|w)$ are computed marginalizing over all the document models and $P(M_i|w)$ is finally obtained by applying Bayes rule.

$$P(q_i|w) = \sum P(M_i|w)P(q_i|M_i) \tag{3.4.21}$$

3.4.6.1 Variations on Relevance Models

From the initial publication in 2001, relevance models have attracted interest given their superior retrieval performance over query likelihood. However, term expansion appears to hurt precision in some domains as a result of estimation noise, and improvements have been proposed to address some of these problems.

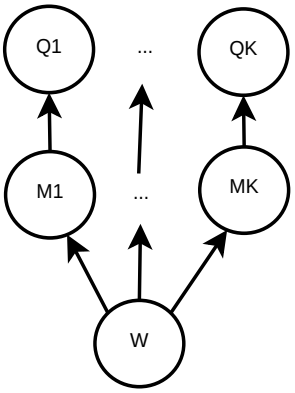

FIGURE 3.3: Bayesian network for Method 2

One area of research has tried to improve the probability estimates derived from the model. Wei and Croft introduce Latent Dirichlet Allocation [8] smoothing into the model building phase to correct for individual document sampling bias [99]. Another approach involves smoothing each document against the pseudo feedback document set word probabilities taken as a cluster in addition to the collection priors [47]. Parapar and Barreiro have recently introduced the idea of promoting divergent terms, that is, terms that appear with higher probability in the model as opposed to their regular frequency in the collection using KLD divergence [64].

Other authors have worked on weighing documents differently. For example, in both RM1 and RM2 methods the model a priori probabilities, $P(M_i)$, can be tweaked for further increases in retrieval accuracy, as they need not be equiprobable. If the initial search algorithm is of any use, the document that ranked at the first position is likely more relevant than the one that ranked in position fifty, therefore it stands to reason that the former could have a prior a little bit higher than the latter. Although this is factored in by the probability of generating the query, which affects relative model weights, it is often insufficient. Li suggested employing rank related priors in this fashion and including the query as a special document [44], although it is also possible to incorporate domain knowledge or other features to give extra weight to more important documents by altering the probability of their model being selected. This could include a function of PageRank calculations or any other useful metric pertaining to the problem being considered.

Along similar lines, Keikha challenged the idea that top documents must be weighed in proportion to the probability of generating the query [34], introducing the concept of document effectiveness, or the mean average precision resulting from performing queries with a relevance model composed from a specific document. Then he tried to estimate effectiveness with several features under a regression framework and use the predicted

effectiveness scores as weights for the documents in the model. His approach showed performance improvements over relevance models in the TREC Robust04 track.

Finally, other modification that has caught ground due to increased performance highlights the nature of relevance models as a term expansion method. It consists of interpolating the original query against the relevance model using Jelinek Mercer smoothing [1]. The alternative model, known as RM3, was used by the UMass team at the TREC 2004 competition, and it persists in the most successful implementations to date.

Chapter 4

Evaluation Metrics

One of the most important aspects of building a real world search engine is in the evaluation stage. While creating better retrieval models and trying out ideas is attractive to most researchers, experimenting and testing that they actually work to ensure incremental progress is a tedious though necessary aspect of the whole process. Like in many other aspects in life, it pays off to be rigorous. In this section the attention is focused on offline evaluation, although in a live setting with access to users online evaluation takes precedence due to the massive amount of data points that can be collected in a short period of time. Nonetheless, offline evaluation is what makes results among different groups of researchers comparable, if only because there are standardized test collections to play with.

All the metrics that are going to be discussed have in common that relevance to an specific query is known in advance and can be measured without a shed of doubt. For this assertion to be true, the collection of documents that are searched has to be static. A painful manual process of tagging documents is necessary, because for better or for worse, people are still much better than machines at pattern recognition purposes. A common practice is to pit several search engines against each other and assume that the set comprised by the union of the top results coming from them encloses the full relevant set. Then individual value judgments are restricted to a small document set instead of scouring the whole document collection. This is known as search engine pooling [33] and is the basis for the elaboration of the international TREC tracks. It has the advantage that search engines can be evaluated in a real sized data set, where priors can be adequately estimated, at the expense of introducing a little bit of noise into the process.

4.1 Common Performance Metrics in Information Retrieval

4.1.1 Precision

In simple terms, it is the fraction of relevant documents returned by a search engine out of a subset of results. It is typically evaluated at several points within the ordered ranking of results, therefore it is customary to report metrics such as P@10, where precision is reported exclusively for the top ten documents of the ranking. In any decently functioning search engine, P@K decreases as K grows, because the ranking is purported to capture some underlying notion of relevance and places relevant documents on top with higher probability when doing a good job.

4.1.2 Recall

Recall is the fraction of relevant documents returned out of the full relevant document set. The figure is a close cousin of precision, because as recall grows, precision diminishes as the search engine is forced to return more and more results. A common practice is to plot the two of them in a 2-D chart, with recall in the X-axis and precision represented in the Y-axis. Once again, practitioners in Machine Learning will recognize that this is nothing but a lift curve, where the problem is treated as a classification task and the idea is to place documents classified as relevant on top, listing them in descending order of prediction confidence.

4.1.3 Average Precision

Precision over a subset of documents tells only half the story, as it gives equal weight to a relevant document that is placed at the top of the ranking than to one that appears at the last rung being considered. Therefore unless it is evaluated at several different points it is a bad metric for summarization purposes. Average precision (AP) attempts to synthesize precision at several points in one single figure, and it is a much more informative metric for discerning search engine performance. It is an average of the P@Ks along the way in the segment of documents being considered, where $rel(k)$ is a Boolean indicator that tells if a document at a given rank was relevant or not:

$$AP = \sum_{k=1} \frac{P(k)rel(k)}{|R|} \tag{4.1.1}$$

4.1.4 Mean Average Precision

MAP is simply the arithmetic average of AP over a set of queries. It is considered a very good indicator for performance and is reported in most research papers. Sometimes GMAP is used instead, or geometric mean average precision, which is the exponential of the mean of the log transformed APs. The idea for this formulation is that since the logarithm grows sub linearly with increasing values, low performing queries are given more weight in the final figure. The exponential transform that is applied in the end brings the result to the same scale than the original AP.

Nonetheless, MAP has been shown to display inconsistencies when applying search engine pooling at different depths [57], although it stabilizes for a deep enough pool. A visualization of the problem is given in fig. 4.1, where the points should fall in a straight line when charting a scatter plot. Even so, the realization has led to new evaluation metrics that attempt to correct these instabilities.

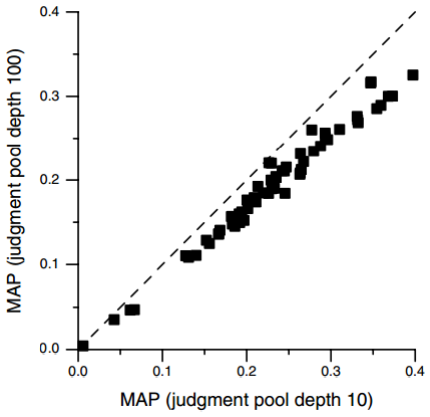

FIGURE 4.1: MAP evaluation of IR systems compared at depth 10 and depth 100

4.1.5 NDCG

New metrics that have been developed in recent years focus on the importance of the first results in the ranking. Behavioral studies in real world settings have discovered that users are an impatient bunch and will leave a site if they are not satisfied after the initial experience. Since they scroll from top to down, their judgment about the quality of the results is heavily influenced by what they see in the top results. It does not matter much that a relevant document is buried in position 20 or later if the documents on top are completely irrelevant for the end user. NDCG brings the idea of utility as used in economics and assumes that higher ranking documents are more useful if indeed relevant than lower ranked ones [30]. For one thing, the user experience is not disrupted

as the individual is satisfied earlier without the need for scrolling. Additionally, it increases the chances that he will not walk away empty-handed. NDCG or normalized discounted cumulative gain captures this idea with a utility as a function of rank and graded relevance, normalizing the score for the maximum gain in utility that could have been realized if the most relevant documents would have been placed on top (IDCG). This last step constrains the metric to fluctuate from zero to one.

$$NDCG_p = \frac{DCG_P}{IDCG_p} \tag{4.1.2}$$

$$DCG_p = \sum_{i=1..p} \frac{2^{rel_i} - 1}{log(1+i)} \tag{4.1.3}$$

4.1.6 Expected Reciprocal Rank

ERR is another metric that also accepts relevance graded judgments as input. It was created by Chapelle [10] with the idea in mind to improve some problems attributable to the DCG term, in particular that its additive properties assume independence. ERR has been shown to correlate well with click-through rates obtained from query logs. It is modeled following the cascade condition of browsing behavior. A user is said to be satisfied by the results with probability R_i, which is dependent on the position and can be estimated from log data or set directly by means of human judgments. If he is not satisfied, he continues browsing with probability $1 - R_i$. Then the formula for ERR is the reciprocal rank times the probability that the user stops at the given position:

$$ERR = \sum_{r=1}^{n} \frac{1}{r} \prod_{i=1}^{r-1} (1 - R_i) R_r \tag{4.1.4}$$

4.1.7 Ranked Biased Precision

In RBP, each user receives receives a certain utility for reaching a particular document equal to the relevance grade of the document. But similarly to NDCG, there is a certain amount of discounting to correct for the fact that some positions are more likely to be seen than others. In its simpler formulation, it assumes that the user scrolls down the ranking with a constant probability p and stops browsing with probability $(1 - p)$.

The formula is given by the following expression, where k is the rank of the relevant document d_k, and $g(d_k)$ is an integer indicating whether the document is relevant or not. It potentially accepts several grades of relevance, such as NDCG.

$$RBP = (1-p)\sum_{d_k \in R} p^{k-1} g(d_k) \tag{4.1.5}$$

4.1.8 BPREF

BPREF is a metric inspired by Kendall's τ. We analyze documents up to a given d_k and define $|NR|$ as the cardinality of the non-relevant documents enclosed in the ranking from d_1 up to d_k. The idea is to capture the proportion of documents that satisfy the relevant pair comparison, where a relevant document's score is expected to be higher than a non-relevant one. The formula is as follows, where N_{d_r} is the number of non relevant documents above in the ranking from the relevant document d_r:

$$BPREF = \frac{1}{|NR|}\sum_{d_r}(1 - \frac{N_{d_r}}{|NR|}) \tag{4.1.6}$$

4.2 Is the Difference Statistically Significant?

It is commonplace to compare two retrieval methods by running queries against a given data set. Once the execution finishes, the researcher hopefully generates a bundle of metrics either on his own (discouraged) or through TREC eval (he'd better do this) and is often left wondering what the different numbers mean. On some particular metric, one approach may appear as better, but on another it may not, and it is hard to gauge which is better without a formal comparison method. Fortunately for IR researchers, statisticians have been studying the subject for decades and have come to develop some tools to aid in the decision making process.

However, a word of caution must be put forth. Frequentist statistics are often abused when declaring that statistically significant relationships have been uncovered. Many scientists are starting to question the robustness of their methods along with publication bias because there is a worrying tendency for conclusions to be overturned by repeating the same experiment over a larger sample size. For more material on this subject, please see the work of Ioannidis [27–29].

The core of the argument is Bayesian in nature, as the probability of H_A being true conditioned to the test being positive is subject to an a priori probability $\frac{|H_T|}{|H|}$, where H_T

is the set of true hypotheses available from the hypothesis universe. When the cardinality of H is much larger than H_T, many claims will be published that are essentially false. This is in some way similar to mammographies being used as a screening tool for the detection of breast cancer. One positive mammography result does not alter the a posteriori probability of being ill by much, however two independent positive results are a much more serious indicator of the illness being present.

Thus retesting or testing over large samples is an essential ingredient for scientific integrity and reproducibility. Nonetheless if the pressure to publish is high and journals favor the report of positive results, this may not be in the best interest of the researchers themselves, as they can refute months of their own work and come out empty handed in the end.

Ioannidis' area of focus is genetics, where the number of hypotheses is large, as many genes can cause or affect a disease. In Information Retrieval our set of hypotheses is much smaller and this is usually not an item of concern. Normally the cardinality of the hypothesis universe H is restricted to two or three hypotheses at most. Nonetheless this is something to be aware of as the parameter space grows. Tools like the Bonferroni correction in ANOVA tests are used when we know in advance our hypothesis universe H, but it is a good practice to keep in mind what other alternative hypotheses we may be leaving out when we carry out an hypothesis test before declaring a significant relationship exists. A brief explanation follows for the hypothesis tests most commonly used for comparing metrics in Information Retrieval.

4.2.1 Fisher's Randomization Test

This is a procedure that can be applied over metrics of any kind, hence its usefulness. For example, there is no statistical significance test specifically designed for medians. While a t-Student test is inappropriate here, either bootstrapping, which will be covered later, or Fisher's method can produce reliable results. The idea is to examine the likelihood that an observed result or a more extreme case is seen given that there is no difference among the two metrics. The no difference claim is H_0 or the null hypothesis, and the probability just described is what is called a *p-value*.

Fisher's method tries to estimate the *p-value* through simulation. Given two samples A and B, we can compute the cross product of pairs $(A_i, B_j) : \forall i A_i \in A, \forall B_j \in B$. Then we measure the difference for the metrics among the constituents of the pair and see if it is as far or farther from the null hypothesis (the difference is zero) than the point estimate (the observed difference in the sample means). We can test for deviation in one or both directions. The *p-value* is the proportion of pairs that fulfill this condition

from the cardinality of all possible pairs. If it is lower than a certain threshold, typically 0.05, we reject the null hypothesis because there is a low probability that the observed difference is due to chance if H_0 were to be true.

4.2.2 Bootstrapping

Bootstrapping is similar to the previous method, in the sense that it is simulation based. The idea is to generate samples of the same size than the empirical ones by sampling from A and B with replacement. Then we measure the difference in sample means and repeat the process many times over to find out the proportion that is as far away from the null hypothesis as the point estimate. The rationale behind the method is to treat the original samples as exemplars from the population distribution. For every single data point, if we were to draw from the real population distribution, there are likely many of them waiting to be retrieved. Repeated sampling with replacement then assumes that the size of the population is practically infinite. As such, bootstrapping is only valid when estimating confidence intervals and *p-values* for small samples relative to the real population size. The procedure for finding the *p-value* once a sample is generated is the same as for the randomization test, averaging the result over many different trials.

4.2.3 Student's t-Test

The *t-Test* was originally conceived to test for statistical significance across means of Gaussian variables when a regular test employing a *Z-statistic* won't suffice because the sample size is too small. *t-Tests* have an interesting story that would make Statistics lectures a little bit less dry and more engaging. William Gosset was working in quality control for the Guiness beer at the beginning of the twentieth-century and was confronted with the problem of deciding if an observed difference was significant but only counted with small samples. When the number of samples is lower than 30 the *Z-statistic* is known to be unreliable. His invention was to create a probability distribution very much resembling the normal distribution but flattened out and with a wider spread, to cover up the uncertainty in the standard error estimate when using small samples. When he was ready to publish his findings, his employer would not let him use his real name because this would alert competitors in the beer industry about the statistical methods they employed to ensure a high quality product. They agreed to publish under the pseudonym of a student's name, and hence the name of the distribution up to the present day.

Figure 4.2 plots the probability distribution function for several *t-distributions* with varying degrees of freedom and compares it to the normal distribution. As the number of degrees of freedom grows, the closer the *t-distribution* is to the normal distribution.

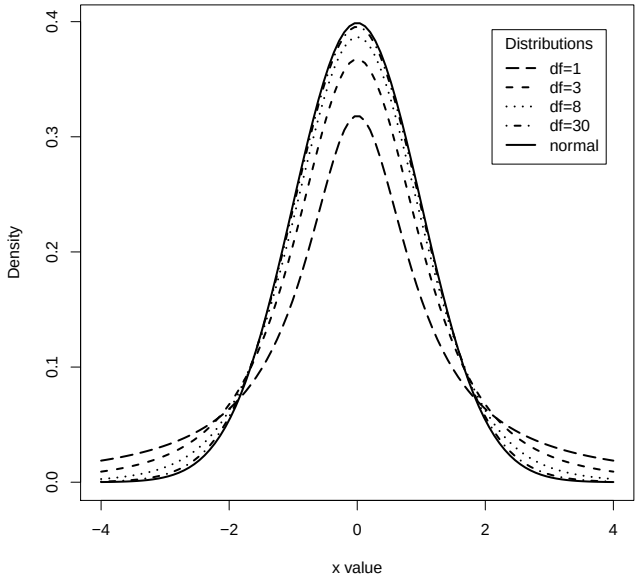

FIGURE 4.2: t-distribution with several degrees of freedom overlaid

When evaluating samples of queries originating from two search engines, the results are said to be paired, because both engines are executed over the same data, thus when a system performs poorly on one query the other system is likely going to perform poorly as well, and so on and so forth. In Statistics, when dealing with paired data, the habitual form of analysis is to subtract the means pairwise in both samples and perform a test to see if the result is different than zero. The t-statistic is tabulated and can be looked up or computed in statistical packages such as R. The formula for obtaining a *t-statistic* is similar to the *Z-statistic*:

$$t - statistic = \frac{PointEstimate - NullValue}{StandardError} \tag{4.2.1}$$

When looking up the value in tables there is an additional parameter for the degrees of freedom, which is chosen conservatively to be sample size minus one. This gives us the probability of observing a higher or equal difference than the one we are aiming for in one tail. If performing a two-tailed test, it is important to remember that the *p-value* is twice the amount obtained.

4.2.4 Wilcoxon's Sign Test

Sometimes it is not adequate to assume normality in the sample obtained, as in the Student's *t-Test*. An alternative to Fisher's Randomization procedure is to carry out a Wilcoxon's signed rank test in said circumstances.

Given two samples x and y of the same size that are drawn from two population distributions, the variable of interest is whether their respective population means are different or not. Let z_i be the difference between paired samples $x_i - y_i$. Wilcoxon then orders the resulting z_is in increasing order giving a rank according to the position in this sorted list, with ties being resolved by setting them to the mean of the ranks they occupy and the z_is that are equal to zero being removed from the sample. Wilcoxon's statistic is the absolute value of the sum of the signed-rank values: $W = |\sum_i sign(z_i)Rank_i|$.

As the sample size increases, this metric approximates a normal distribution and a z-statistic can be computed: $z = \frac{W-0.5}{\sigma_w}$. The resulting value can be looked up in a Gaussian probability distribution table for the required significance level of the test being performed. If the size of the reduced sample, once the zeroes are removed, is lower than 10, the normal approximation is unreliable and there are tabulated lists where the appropriate critical value can be derived.

Chapter 5

An overview of Big Data Technology

According to Bae Brandtzæg from SINTEF ICT, it is estimated that 90% of the information available today has been produced during the last two years [16]. Pinning the exact figure is a challenge as the subject is a moving target, what is not up to discussion is that the growth rate has been staggering, rising at an approximately exponential rate. This silent revolution was initially fueled by the growth of the Internet and the adoption of Information Technology within corporations, followed by the spread of mobile devices and the arrival of personal sensor technology. Add to that falling costs of storage that permit recording nearly any meaningful interaction and it is not surprising that analysts are claiming that businesses dealing with data are posited to become a multi billion dollar industry by 2020, with a projected growth in employment of 4.4 million data related roles by the end of this year [21].

This trend has brought with itself a new wave of tools and technology specifically designed to withstand the issues that surface while dealing with terabyte and petabyte scale data sets. Many companies are routinely confronted with the dilemma of abandoning their well known relational database systems for their relatively unknown NoSQL counterparts. What was once the domain of Internet giants such as Google or Facebook is becoming commonplace as scalability problems start to appear now in even a mid sized Internet company.

In addition to handling transaction volumes and meeting storage requirements, data is worth nothing if insights and new perspectives to aid in decision making are not derived from it. Traditional analytics packages are used to operating with the full data set in memory, a constraint that falls apart with the dimension of today's data centers. While sampling may be appropriate in some circumstances, oftentimes it is an imperative to

be able to work with the full data set to extract full value from it. It is an old adage in Data Mining that more data beats complex algorithms, and new techniques and implementations are being developed to work in distributed settings. The next section covers the paradigm that has arguably emerged as the industry standard for Big Data processing, the MapReduce framework.

5.1 History

In 1996 two Stanford graduate students started working on a search engine for the web that was to be powered by one of the most successful applications of eigenvectors to date [63]. Back in the time the leaders of the day relied heavily on human categorization as Information Retrieval techniques were subject to web spam, with people trying to take advantage of the systems to position their web sites in a high ranking position. The algorithms had been previously applied to controlled collections of documents and it was becoming increasingly clear that in a hostile environment a new approach was needed. Initially named Backrub, the academic experiment worked out incredibly well, gaining momentum and leading to the birth of a new company, Google.

As the business grew, and so did the size of the web, it started to face difficulties in the distributed processing tasks needed to handle large data sets. In 2003, a paper was published revealing the details of Google's internal distributed file system [22], the company's response to its storage demands. It provided a consistent fault-tolerant scheme, resistant to data node failure and designed with the idea in mind of preserving data locality while dealing with files. It must be noted that while storage capacity had been increasing steadily throughout the years, bandwidth remained a limiting factor when transmitting data, and so did the reading speed from secondary storage.

A year later, MapReduce was born out of the need to process web scale graphs, providing a simple programming framework to help in writing distributed programs dealing with large datasets [17]. Before it made its appearance, distributed data processing was arcane and error prone, with every programmer having to deal with low level details such as locking, caching, and load scheduling on their own. MapReduce provided a clear and structured approach to writing software on top of Google's File System where developers could focus on the business logic and leave the remainder of the work to the platform.

When the two papers made its appearance, a senior engineer named Doug Cutting was working on the evolution of Apache Lucene, an open source information retrieval system. The software consisted of a pack of retrieval algorithms and indexing modules but it

lacked other components available in commercial grade systems such as a distributed crawler. The new project, Apache Nutch, promised to transform Lucene into a full fledged search engine. Cutting was working on his own distributed framework and was struck by the simplicity of the ideas developed at Google, along with the flexibility that they provided in writing certain search engine tasks. He set out to rewrite the core of his system and by 2007 he revealed his open source MapReduce implementation to the world: Apache Hadoop. The project gained traction as it came under the umbrella of Yahoo!, that heavily subsidized its development.

And as the saying goes, the rest is history. Hadoop has become the industry standard in the world of Big Data, although as of late Apache Spark has started to appear as a serious contender. Nonetheless numerous vendors offer their services and solutions on top of it, and competing technology is based on the same core principles, therefore it is important to understand the foundations well.

The next two sections give a bird's eye view of the technology covering its main technical aspects. To keep the exposition short, details have been deliberately omitted and the interested reader is referred to the papers in question, where an in depth understanding of the topic can be gained.

5.2 The Google File System

When data resides in one processing unit, hardware failures are a rare occurrence, and a simple backup strategy can suffice for protecting from its undesirable effects. However, when moving to thousands of servers, that unlikely event takes place every single day, causing headaches to system administrators and data consumers alike. In order to prevent data loss and at the same time guarantee information access at all times, internal mechanisms for data replication and node coordination must be built in within a storage network.

Google File System (GFS) was created out of the need of dealing with such problems. Inspired by the success of the Domain Name System [56], one node in the cluster is designated as the Master node. Files are partitioned in 64 MB chunks called blocks and can be stored anywhere in the network, typically replicated across several computers to prevent data loss. The purpose of the Master node is to act as a central repository for meta data information, keeping track of the location of all blocks in the network.

The remaining computers act as Chunk servers, and serve as storage nodes in the system. Random access to data is prohibited, as the system is optimized for streaming large files. When a client application wants to either read or write a file in the distributed file system,

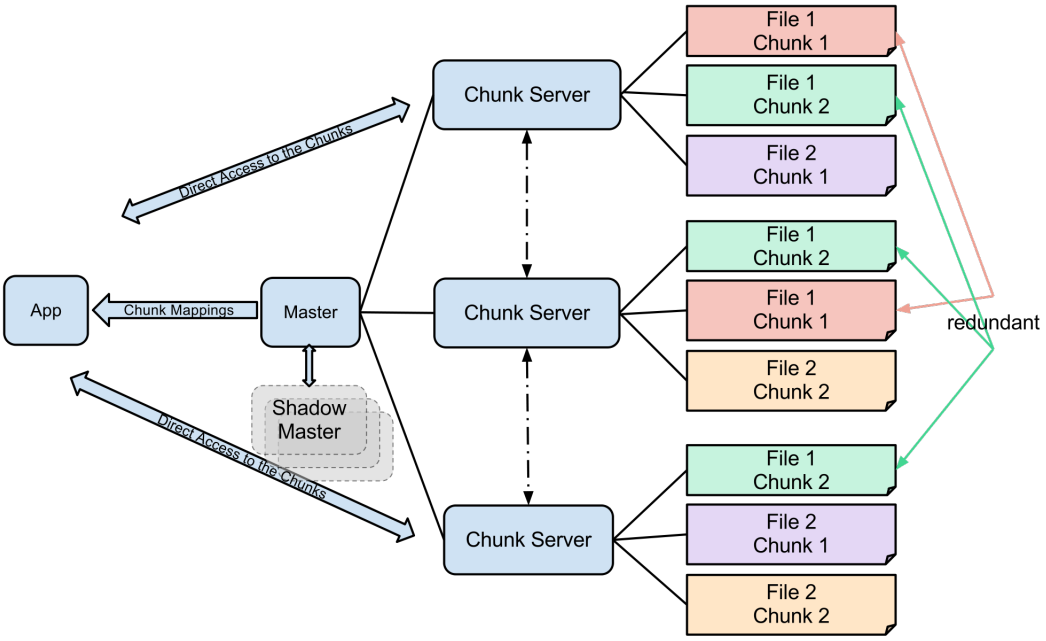

FIGURE 5.1: Google File System architecture

it first contacts the Master Node to either locate the blocks corresponding to the file or to orchestrate the data write operation. An overview of the GFS layout can be found in fig. 5.1.

Redundancy is built in into the system and, as a result, the storage network is resilient to node failures. In the event of failure of the Master node, operations are continuously written to a transaction log, functioning with system recovery checkpoints to avoid data corruption. Master node downtime results in the impossibility of accessing data in the cluster for a period of time. It is important to notice though that given that the functionality is concentrated in one node, it is as rare an event as your home PC failing due to hardware related issues.

5.3 MapReduce

MapReduce is a distributed programming paradigm [17] using as a metaphor the composition of two simple operations that have its roots in functional programming:

- Map: In this stage the data is partitioned into disjoint blocks that are amenable to parallel processing. Since the GFS already stores its data split in blocks, it is a good idea to define the partition unit size as a multiple of the chunk size. The system operates in such a way that the source code is pushed to the data nodes, that also act as processing

units. This mode of operation minimizes data transmission over the network, which is a slow operation.

The fundamental building block of MapReduce is the $< key, value >$ pair. Data in a map operation is viewed as a sequence of pairs and the output of this phase is another set of $< key, value >$ pairs, potentially of different types. The programmer's role is to interpret the incoming stream of data, perform some potentially useful transformations on it, and create another stream of pairs. After the map stage is finished, data is written to disk in the local processing nodes. It is kept in a secure place just in case an operation needs to be retried later on, though the overhead of storing it in the distributed file system is sidestepped as the transient nature of the operation does not call for extra consideration. Afterwards a functional unit called a Partitioner decides where to send the data for further processing.

- Reduce: The reduce operation starts with what is called the shuffle and sort phase. To be precise, data is sorted at various points, starting in the Map phase, and the reducer's role is one of merging sorted data streams prior to further processing. All the pairs that hold the same keys are sent to the same reducer for aggregation, with the record pairs being handed to the programmer's business logic ordered by value. The output of the reducer code is another stream of $< key, value >$ pairs, which is finally written to the distributed file system.

Optionally, the data flow can be sped up through the usage of a Combiner, which is a Reducer that is executed locally within the Map stage. If data is aggregated in intermediate stages it can be compressed, resulting in less bandwidth use. For using a Combiner the processing operation performed must be associative and commutative, otherwise it will not result in the same output.

The canonical MapReduce example is counting words in a divide and conquer approach, as shown in fig. 5.2. At the outset, several files or portions of the same file are processed by different map tasks, in this case four. The mappers proceed by iterating its text chunk and emit key-value pairs with the words as they are streamed, along with the number of times that they appear (treating words atomically, the starting count is always one). In the reduce stage, pairs with the same key are grouped together. For example, *Mary* appears in map tasks #1 and #3, and this key is always processed by reducer #2. The final task is merely counting the word appearances that have been generated by the different mappers. Since each key is sent to only one reducer, the result is consistent to what is expected.

The simple processing framework just described is incredibly flexible and accommodates many data processing tasks. At the same time it isolates the developer from all

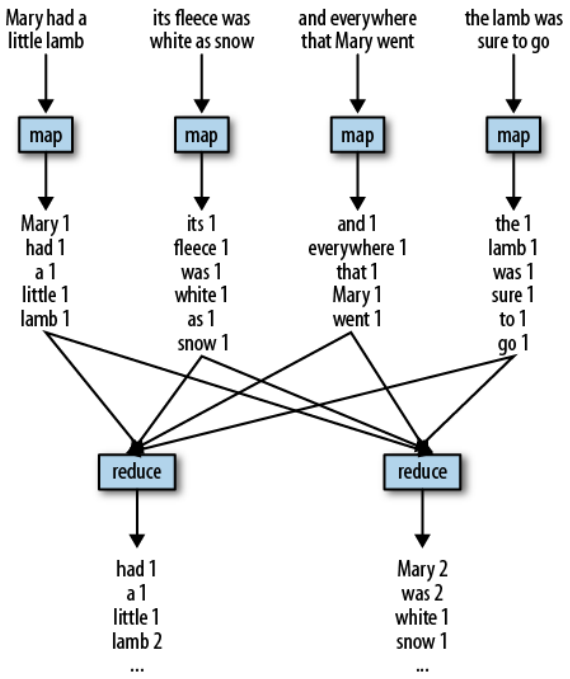

FIGURE 5.2: The MapReduce Processing Framework

the difficult moving parts needed to build a distributed system. All the aspects pertaining to job scheduling, including but not limited to load distribution, task monitoring, failure detection and recovery, are dealt with without exposing intricate details to the programmer.

For example, one common task in Information Retrieval is building what is called an inverted list index. The input data is a set of files which contain words, and the desired output has the relationship reversed: for each word we want to know in which files it appears, to simplify retrieval of documents given a certain user query. In MapReduce, the Map stage would emit a stream of $< word, documentName >$ pairs, and by the own nature of the system, all document names related to a certain word would be distributed to one computing unit. The role of the Reduce phase is merely to write down the output in whichever data structure is needed for the search engine later on, as the final output in a given reducer can also be seen as a sole $< key, value >$ pair where the key is the word in question and the value is the list of document names or paths.

Chapter 6

Language Modeling for Job Titles

Generation of job recommendations from a content perspective can be approached in several ways. A straightforward method is to index all resumes in a general purpose search engine and perform specially crafted queries against the resulting index, typically following bag of words models. This simple approach, which reduces the matter to operations over frequency counts, is easily confused by information that is no longer up to date. As we accumulate experience, we typically evolve from low paying jobs to higher paying ones, and our resume may be peppered with past positions that we no longer want to perform. An implicit assumption in Information Retrieval, that is rarely verbalized as such, is that a majority of content present in the document being indexed is relevant and pertinent. In dealing with resume data, this assumption is often violated. The signal is thus lost in the noise and retrieval of relevant documents becomes difficult with standard techniques.

There are probably many courses of action possible to handle this problem. One of them is to try to determine the skills a candidate may possess, ascertain the market value of each of them, and make recommendations based on the likelihood that the person under consideration may want to perform a job requiring certain skills given its current skill set, following David Ricardo's comparative advantage [74]. This is an interesting angle to consider although difficult to carry out, as individuals may be more or less verbose when detailing their current knowledge base. The fact that applicants are encouraged to write short resumes does not help the intended task either. Because of this, resumes are not expected to be comprehensive representations of the professional capacities that a person brings to the table. Error prone inference would have to be performed, undermining the effectiveness of the system.

A less ambitious approach is to define a model that looks at the likelihood of job transitions by observing actual people's choices, under the assumption that they behave

following their best interests. Precisely stated, they maximize their utility, which involves mainly salary but also factors other considerations that depend on personality and individual tastes. Without looking at actual market level compensations or people's skills, we can look for people that have held similar jobs in the past, and take notice of which job they chose afterwards. This indirect observation process, while being blind to the underlying reasons that cause people to change jobs and pick some opportunities to the detriment of others, is sufficient for the task that we pretend to undertake, if only inaccurate at times.

To carry out this task, a similarity metric is needed. For this purpose, another important assumption is made in that people that hold the same job title perform similar functions that require equivalent skill sets. While there are bound to be differences across companies, the recruitment industry does tend to standardize on titles, because market players need a common ground when communicating job requirements. If people did not agree on the expected duties of the title "university professor", it would not be informative for the receiver of the message, and precious time would be wasted giving uncalled for explanations in social gatherings.

To illustrate this point, skill based template matching can be performed on resume databases to use it as an input for collaborative filtering algorithms. Table 6.1 shows the top skills for some professions as gleaned by this method. As can be seen from visual inspection, the results appear to be sensible and the hypothesis that a certain title is associated with a given skill set seems to hold true. The underlying collaborative filtering algorithm used to produce the table is in fact a relevance model, as it has been recently demonstrated to be useful for this purpose as well, aside from its usual document retrieval application [65].

Data Scientist	*CEO*	*Recruiter*
Data Analysis	Strategic Planning	Recruiting
Python	Management	Technical Recruiting
Machine Learning	Business Strategy	Contract Recruitment
R	New Business Development	Human Resources
Statistics	Marketing Strategy	Recruitment Advertising
SQL	Entrepreneurship	Talent Acquisition
Matlab	Start ups	Sourcing
Data Mining	Customer Service	Executive Search
Java	Business Development	Interviews
C	Marketing	Permanent Placement

TABLE 6.1: Top skills for several professions

Through the succession of positions in a resume we can not only infer how the individual has gained skills over time, but we can also probably guess their relative market value. A fair postulate is that the job title associated to a position reveals a person's most valuable skills at a certain point in time, as otherwise the person would be working at something else. This is a strict efficiency criterion that most economists would agree can be presumed at least partially accurate. Given that utility increases with salary and therefore a person prefers exercising more valuable skills at the next job, and a person's most valuable skills are the ones exercised at the current job, in the absence of other information the task of generating recommendations can be modeled as a Markov process, as in fig. 6.1. Under this view the next job depends exclusively on the current one, the outcome then being path independent. The unpredictability of the jump is dictated by a candidate's learning rate and personal interests.

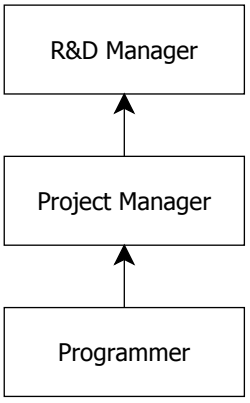

FIGURE 6.1: Modeling transitions in job titles

The Markov property is a rather strong assumption that works well in practice and is insensitive to the noise introduced by old jobs, at the expense of losing some context information that may or may not be important for a particular candidate. It also has the advantage of increased run time performance because all the candidates possessing a given job title can be given the same treatment regardless of their past history, caching recommendations between requests.

If we were able to estimate all our probabilities, given a present job or state, to generate a recommendation we would only need to rank successor states sorted by descending probability of appearance. Theoretically speaking, we could define a huge matrix with all possible job titles on one axis and the preceding job titles on the other. Then the remaining work is merely a matter of estimating the probabilities to fill all the cells, which is reduced to counting title frequencies. Space considerations aside, it turns out that people are really creative when it comes to describing their positions. Mimno and

McCallum analyzed a data set of 9722 resumes and found out that 85% of job titles appear exactly once [55]. My own estimate for this figure computed over a 2 million corpus of Spanish resumes leaves it at 75%, which is still significant. The universe of job titles follows a heavy tail distribution, where a few titles accumulate many repetitions, whereas most job titles are otherwise unpopular. This makes direct estimation of the parameters through maximum likelihood impossible and practical methods must rely on approximations of one form or another.

One way to solve this problem is the one tried by Mimno and McCallum, who also attempted to model career path trajectories. In the paper mentioned above, they created a probability model with latent factors by using topic modeling with Latent Dirichlet Allocation. However, their model was limited in the sense that it could not generate probability estimates for titles not seen in the training set, because it counted straight title transitions restricting the sum over the inferred topics s_i in the resume:

$$P(t_{dst}|t_{src}) = \sum_{i=1}^{|s|} P(s_i|t_{src})P(t_{dst}|s_i) \tag{6.0.1}$$

The alternative path chosen for this project consists in adapting relevance models instead, which are known for good document retrieval performance. Dimensionality reduction is nonetheless an interesting avenue to pursue and not incompatible with the approach finally selected.

According to the language generative process, each job transition in a resume can be seen as the target job title being created by the origin job title, introducing the simplification that words are sampled independently to create a word stream. The underlying idea is that "data scientist" is more likely to generate the words "big" or "data" on the title exactly on top than the word "financial". Each transition can be thought of as a micro-document with two fields: the source and the destination title, as seen in the resume. Thus there are two language models, one per field, and smoothing is performed in maximum likelihood estimation when considering word counts.

The model can follow either multinomial sampling, where documents are treated as sequences of words, or the multiple Bernoulli model, where frequency in a document is discarded and in which case negative occurrences are taken into account while ranking. The multiple Bernoulli approach holds merit at first sight in the present domain, given that job titles are expected to be short with no word repetitions.

The procedure for generating recommendations is as follows. Given a candidate's latest job position, query the inverted index by key, looking for a match in the source title field. This task can be performed employing any IR general purpose method, the method

chosen for the implementation being plain TF-IDF. Afterwards, take notice of the highest ranking k micro-documents retrieved and create a relevance model to find out the distribution of words in the target title in the relevant document set. An important modification to the original relevance model formula is applied. When computing the joint probability, words from the query are assumed to be sampled from the language model attached to the source title, whereas the remaining words are sampled from the language model according to the destination title. By proceeding in this fashion we obtain the probability of the words in the destination field being conditioned to the source field given the query.

The rationale for this modification can be intuitively understood from the following example. Imagine that we are trying to generate recommendations for a candidate with "accountant" as the current position and the following micro-document is retrieved: "senior accountant" $\rightarrow$ "founder". For simplicity, assume no smoothing is applied. If we were to generate the relevance model exclusively from the content of the target job title then the probability for the query is zero and the document's content is ignored in the relevance model, when in fact it is pertinent. The correct treatment is to sample the query from the source title, with both language models being paired to compute the joint probability.

The formula when applying Method 1, which was introduced in section 3.4.6, is then modified as follows:

$$P(w, q_1....q_k) = \sum_M P(M)P(w|M_{dst}) \prod_i P(q_i|M_{src})$$

Since both language models are paired, $P(M)$ is the same for both of them, as any of the two language models implies the other with probability 1. It can be left uniform as in the original paper or it can be tweaked at will. Method 2 is modified following the same guideline. Once the relevance model is created and we know the distribution of words in the target title in the relevant document set, this model is finally matched against the titles of the job openings, which are stored in a separate index that is updated periodically. Ranking of job openings can be performed through direct application of the PRP formulation, or using cross-entropy instead.

In principle, using relevance models entails computing the a priori probability of the query, which involves a marginalization over all the terms in the vocabulary. Ideally, this step can be ignored, because the resulting probability that is used for ranking is proportional to the probability of the query, which does not vary. When using negative cross entropy for scoring, ignoring the query does not affect the final ranking, it only

results in scores displaced by a constant that is document independent and therefore safe to remove. With respect to the probability ratio, ignoring the calculation introduces a constant term in the score that is multiplied by the document length. This fact must be confronted, as each summation in the PRP will otherwise yield a negative result, favoring retrieval of short documents. One simple way to deal with this issue is to use the normalized version of the PRP, dividing each summand by the document length, which yields an equivalent ranking to the one that would be obtained if the probability had been computed and the division had been applied, albeit with the scores displaced by a constant. This is the method used in the current implementation.

However, business stakeholders might not be content with negative scores, which offer no meaningful interpretation and a means for eye-balling comparisons across rankings. To content management, computation of the probability of the query can be attempted. The complexity of a direct interpretation of this summation is $O(|V||M|)$, and under the assumption that $|V| \gg |M|$ it can be considered practically linear with the number of terms in the vocabulary. Since the number of terms ranges in the hundreds of thousands, it is a heavy cost operation performance wise if confronted directly. However, in Method 1 both for Dirichlet and Jelinek-Mercer there exists a formulation where all zero occurrence events can be lumped together due to the distributive property of the product, yielding a summation that is $O(|V_m||M|)$, where $|V_m|$ is the average number of terms present in each document model. This is typically a one to two orders of magnitude improvement and makes the computation feasible (due to its simplicity the details are omitted). For Method 2, an accelerated summation can also be obtained with Jelinek-Mercer smoothing. The probability $P(M|w)$, when w does not appear in any document inside the relevance model, is equal to the prior $P(M)$, and afterwards the distributive property can be applied to arrive at a formulation of similar time complexity as stated for Method 1.

In table 6.2 we see top words in the relevant documents according to the model across several different queries. The words themselves have been reduced to root form through application of Porter's Snowball stemming algorithm [71]. This is a practice that is meant to increase recall by matching singular and plural forms together, as well as other types of inflections. The effectiveness of Porter stemmer in particular has been called into question due to its very aggressive behavior, because often it conflates words together that have no underlying semantic similarity [31]. Krovetz's stemmer [39] does appear to help precision at the expense of reducing recall when compared to Snowball, although there is no implementation for the Spanish language readily available.

Canguro	*CFO*	*Project Manager*
tecnic=0.03027	cfo=0.06292	manager=0.13492
comercial=0.01520	director=0.04730	project=0.07576
administrativ=0.01475	manager=0.03446	director=0.02268
profesor=0.01170	tecnic=0.02192	senior=0.02143
departament=0.00792	controller=0.01451	engineer=0.01946
practic=0.00763	response=0.01310	tecnic=0.01803
auxiliar=0.00741	financier=0.01295	consultant=0.01467
becari=0.0058	finance=0.01227	response=0.01176

TABLE 6.2: $P(w|R)$ for various queries

6.1 Further Modifications over Relevance Models

In empirical tests, several problems arose with the model as described above when using the probability ranking principle. Some infrequent words can be given excessive weight if they happen to be included in the relevance model, whether they happen to be on topic or not. In particular, it was identified that cities and towns, while not very common in the resume database, when present they could distort the results. Some people may state in their resume "Medical Doctor at the Torrejon de Ardoz hospital". This immediately triggered low quality job recommendations of the class "Urgent waiter needed in Torrejon" and such, because Torrejon, being a low frequency word, carries much weight in the final score due to its presence in the divider. The solution was to filter out locations and treat them as stop words, although the underlying problem remains for other words and is only masked. It is presumed that locations are non-informative in the Human Resources domain, hence the treatment. A database of geolocated places was downloaded from the NGA GEOnet Names Server, a service provided by the US government. The names were indexed into a Patricia trie data structure [61], habilitating data compression and fast retrieval. Prior to indexing and querying, substrings in the query or the job title are searched for in this database, and if a match is found, it is removed from further consideration.

This process partially solved a more general issue that warrants careful attention. Since language follows a Zipf distribution, there are many low frequency words in the collection. And because their number is quite high, some of them will unavoidably find its way into the relevance model simply by chance, while being unrelated to the topic of the query. Thus treating locations as stop-words only removes a portion of the noise introduced by words of this kind.

A possible idea to explore is to perform probabilistic topic modeling to find out the distribution of the word across several topics. Uninformative words are expected to present

a nearly uniform distribution when examined under this light, and can be penalized or removed from the model if the analysis shows them to belong to this group. A simpler idea is to ignore low frequency words in the scoring function, as given by their collection prior, if they appear in the relevance model with low probability, because the relative error in the likelihood ratio is expected to be high and can greatly pollute the results. When implementing this idea positive results were obtained, but given that such thresholds have to be periodically adjusted when other parameters change, it was clear that a more formal statistical method was called for, which led to the proposed smoothing function presented in the next section.

6.2 Relative Error Smoothing

The idea for this type of smoothing is to try to control the risk incurred in the PRP through dynamically adjusting smoothing parameters for specific words. On some words it pays off to be aggressive while on others it would be a good idea to be more conservative. The main risk in ranking is that we claim that $P(w|R) > P(w|C)$ when in fact the word occurs in the relevant set with the same frequency than in the collection. There is also the risk that the weight of the word is underestimated, which would decrease recall, although the top of the ranking is governed by the former assertion, and this is the most important set of returned documents because they are seen by the end user with higher probability.

Assume we work under the multiple Bernoulli model. Conceptually, we observe a stream of random words coming from several Bernoulli processes in the documents used to build the model and we have to make a call to judge whether the estimate for $P(w|R)$ differs or not from the background prior. Let us first suppose that we are trying to estimate the parameter p of a Bernoulli process and all we have is a single trial to come up with an approximation $\hat{p}$. As shown in fig. 6.2, the problem is that the lower the true parameter p of the Bernoulli process, the higher the variance of the relative error in the estimate $\hat{p}$:

The idea is to assume that all documents are unrelated to R and thus $P(w|R) = P(w|C)$, which is the parameter for the Bernoulli processes. The variance in the estimate for each document is $P(w|C)(1 - P(w|C))$. Then, given that the final estimate for $P(w|D)$ is arrived at through interpolation of the maximum likelihood estimate with the prior, which can be taken as a known constant, we would pick a value for the interpolating parameter λ such that the variance of the probability ratio after interpolation remains constant, as this error is what distorts the final ranking when words enter the relevance model simply by chance.

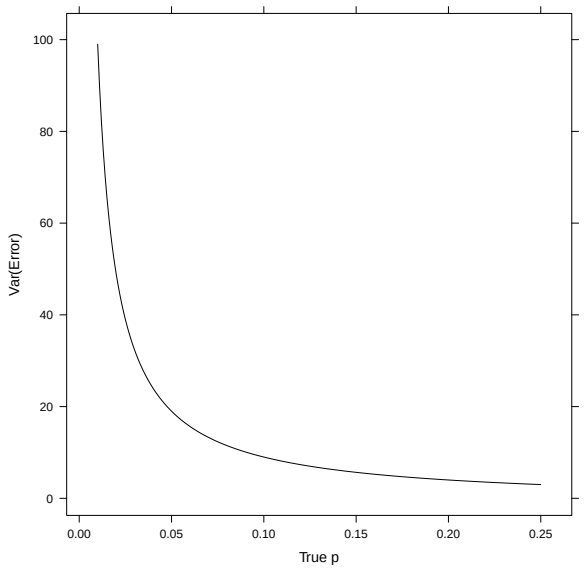

FIGURE 6.2: Increasing error as a function of p

Since we are holding a portfolio of documents to estimate the population probability, assuming they are equally likely and conditionally independent, the variance of the whole lot is $\frac{P(w|C)(1-P(w|C))}{n}$. Alternatively, documents can contribute to the variance in proportion to their respective weight in the relevance model as per the query likelihood weights. When we perform Jelinek-Mercer smoothing we interpolate between the prior and the estimate, therefore our variance is scaled by a constant value $1 - \lambda$ for every document. Dividing by $P(w|C)$, which is the purported actual population parameter, and taking it as a constant to compute the variance, we arrive at:

$$\sigma^2 = \frac{P(w|C)(1 - P(w|C))(1 - \lambda)^2}{nP(w|C)^2}$$

Rearranging the terms as a function of lambda, we get the following equation:

$$\lambda = 1 - \sqrt{\sigma^2 * \frac{nP(w|C)}{1 - P(w|C)}}$$

Finally, to control the risk in the estimate, we force σ^2 to remain constant as $P(w|C)$ becomes smaller, which is a tunable parameter. If the target variance is too high, λ becomes negative, in which case we saturate it to zero and the full maximum likelihood estimate is used. Conversely, as the target variance approaches zero, λ becomes one, and more smoothing is applied. As a function of $P(w|C)$, the product is dominated by the

numerator. When $P(w|C)$ is low, the whole product approaches zero and more smoothing is required to keep the variance constant than when $P(w|C)$ is higher. The target variance, along with the prior of the word in question, and the number of documents n used for the relevance model, fully define the lambda smoothing parameter to be used for Jelinek-Mercer, which is adjusted dynamically for each specific word.

Chapter 7

Architecture of a Job Offer Recommender

One of the joys of carrying out research in industry as opposed to academia, is that with a little bit of luck you get to see your work being used and it gets to impact the lives of real people outside the lab. Unfortunately, this also means that a prototype is not enough, and that there are many details that must be dealt with to build a production ready implementation.

Initially designed as a prototype, the project became increasingly complex as modifications were introduced to address performance issues. This section pertains to the engineering aspects of building such a system. The following sections describe the architecture of the initial version and the refinements that are being introduced in the product road map, presenting the rationale behind them. There is often a focus on theoretical foundations to the neglect of practical considerations on describing proposed solutions. From an engineering perspective it is also important that these scale and that the cost in terms of human resources is reasonable.

7.1 System components

The job recommender is composed of several modules, and these interact with the remaining components of the web site. As any non trivial project, a job portal contains its own share of legacy systems, therefore modularity with clearly defined interfaces and responsibilities is important to keep overall system complexity under control.

7.1.1 Indexer

Every IR system needs a way to efficiently access the data to be able to resolve queries at an acceptable rate. For the described system, there are two indices needed to be able to generate recommendations:

- model index: This store is typically created once and does not change very often, if at all. It is the result of processing a data set of resumes to annotate job transitions. The purpose is that for a given job title we can infer the probability distribution of the words appearing in the succeeding jobs that appear exactly on top, assuming reverse chronological order, as is customary in enumerating experience in CVs. The task is merely one of word counting and can be accommodated into a classical inverted list data structure.
- offer index: This relates to a batch process that is executed every night to take notice of the new job openings that have been created throughout the day and to delete the ones that have expired, either because the job opening has been filled or simply because the company involved does not want to continue paying for the advertisement. Offers are imported from the external Elastic Search system that serves the incoming queries to the search engine of the web site. An import task is required for two reasons: web site load would be too high if the process of generating recommendations were to be done directly over the Elastic Search server, and the performance would affect active users, and because of implementation ease as well.

7.1.2 Query Resolver

The query resolver admits a job title for a given user, along with several parameters to constrain the query such as geographical location and salary preferences, and exposes a REST interface returning a JSON payload with the appropriate recommendations. It is self contained and is invoked from the Ruby back-end that renders the Web site, as well as by a batch process for generating recommendations. This scheme follows a service oriented architecture (SOA) that allows decoupling functions in loose components that can be composed to arrive at meaningful actions, increasing re-usability of the involved sub modules.

Both the model index and the offer index can be selected at run time, for example to take into consideration resumes in several languages or job offers from different countries that can be partitioned for faster retrieval, under the plausible assumption that a user is

only interested in job openings from a certain country. There are also cultural differences that must be accounted for in naming professions. For example, although "community manager" has become popular in Spain to describe someone who handles social media and related marketing activities, this phrase does not carry the same meaning either in the UK or the US.

Other parameters in the interface include a restriction on the maximum number of results that are returned, a minimum score threshold that the job offer must satisfy to be included in the result set, and geographical constraints that express the candidate's preferences for location in the form of a list of countries or provinces. In fig. 7.1 there is an example query to the recommendation server:

```
curl --data "count=10&threshold=3.1&country_ids=56&
model_index=index/resumes&job_index=index/job_openings&
query=project+manager" http://localhost:8080
```

FIGURE 7.1: Sample CURL request

The payload result includes a JSON document such as the one in fig. 7.2, containing the identifiers for the job openings, whose content is queried elsewhere to display it to the end user, the scores assigned to each of them, along with the title of the returned offers:

```
{
    "time": 105339,
    "results": [{
        "title": "\"Senior Project Manager\"",
        "job_opening_id": "20157482",
        "score": 4.10707950592041
    }, {
        "title": "\"PROJECT MANAGER\"",
        "job_opening_id": "20280675",
        "score": 3.743000030517578
    }, {
        "title": "\"Project Manager\"",
        "job_opening_id": "21587015",
        "score": 3.743000030517578
    }]
}
```

FIGURE 7.2: Server response

7.1.3 Experimenter Module

Carefully tuning parameters and evaluating selected configurations to ensure that changes introduced indeed make progress over a previous version is a fundamental step in any Information Retrieval system. A module to perform grid search of parameters and to execute queries automatically against a test set collection was programmed. It runs over the full parameter universe given incremental steps defined by the programmer and outputs different metrics for each configuration according to the TREC eval tool, saving the log files for each run in case further work is required to verify correct functioning.

Auxiliary classes to iterate over each parameter type are coded. They contain iterators which cycle repeatedly between a minimum and a maximum value and the operation is stored when all variables reach their maximum. An example source code listing to run a constrained grid search can be found in fig. 7.3.

```
public static void main(String args[]) throws Exception {

    List<Range> parameters = new ArrayList<Range>();
    parameters.add(
        new StringRange("ranking_method", new String[] {"crossentropy","prp"}));
    parameters.add(
        new StringRange("smoother", new String[]{"jelinek-mercer"}));
    parameters.add(
        new NumberRange("lambda", 0.05, 1, 0.05));
    parameters.add(
        new StringRange("model", new String[]{"method1","method2"}));
    parameters.add(
        new NumberRange("num_models",50,500,50));

    int parameterConfigId=0;

    for(;;) {
        printConfigFile(parameters, "relevancio.ini");
        runTest(parameterConfigId++);
        if (allIsMax(parameters))
            break;
        next(parameters);
    }
}
```

FIGURE 7.3: Source code for grid search

The initial implementation, as shown, executed tests sequentially. However, testing configuration options lends itself to parallelization because each task is self contained and does not depend on others for input. As a result, it was decided to program a distributed version of the experimenter module running on top of Hadoop. The program above is modified so that each configuration option is written out to a separate file, one per line, and then this is used as input for a MapReduce job. Each line contains a full set of parameters and since the default Hadoop input format treats a line as a key-value pair (along with the byte offset in the file), there is no further action needed for task

allocation. The whole process can be accommodated in a Map-only job and sent to a cluster of computers.

As the company does not presently count with its own Hadoop cluster, the experimenter was executed in the cloud through Elastic MapReduce, Amazon's version of Hadoop as a service. This setup has the advantage that financial resources are not tied up in computer assets and they can be deployed when needed, with the flexibility to incorporate heterogeneous resources depending on the nature of the task demanded. Some jobs may be more computing intensive than others and higher end machines are needed, as is the case for the grid search mentioned in this section. Other times the task is just a matter of transforming data and regular commodity hardware is more suitable to keep costs under control. Distributed grid search allowed full exploration of the parameter space in one day, where it would have taken about two months in my development laptop, all for a person's day worth of salary.

As a result, it is important as well to rely on actual users for evaluation purposes. Almost all run time parameters that can affect the performance of the system can be configured dynamically without the need for recompiling and redeploying the whole system, to make the life of our system administrator a little bit easier. The idea is to be able to run A/B tests within the same server, running different versions of the software concurrently and obtaining click through rates on which to base a decision. This second evaluation stage is in an early state of development as cascade models for log analysis have to be implemented, otherwise extracting meaningful conclusions from raw CTR data is error prone, because the position where a result is displayed plays a role in the final user actions.

In addition to the aforementioned variables described in the previous section that control query output, configurable options that can be added to the URL include:

- *smoother*: Whether to use Dirichlet or Jelinek-Mercer smoothing.
- *lambda*: Jelinek-Mercer interpolation parameter.
- *mu*: Controls the pseudo-counts added in Dirichlet smoothing.
- *num_models*: Number of top documents used to build the relevance model.
- *ranking_method*: Procedure employed for ranking, either PRP or cross entropy.
- *model*: Method use to estimate the relevance model coming from the first pseudo relevance feedback round, either Method 1 or Method2.
- *silent*: Whether to output extra debugging information to the console or not.

Because the URL would become unnecessarily complex when selecting options, default parameter bundles can be configured in a separate configuration file that is loaded when starting the server and an additional config parameter can be passed in the query to select one of those bundles. This reduces errors in the invocation stage and provides a means for tracking how specific sets of parameters perform in an online setting by first giving them a name.

7.2 Implementation details

Rather than building the project from scratch, the idea was to reuse as many COTS components as possible to trim down costs and reduce the likelihood of introducing difficult to spot programming errors. As such, a platform in Information Retrieval that has been under development for many years and that has been deployed in commercial settings is Apache Lucene. It would be innocent at best to attempt to code a search engine in a few months with all the required components. From Lucene it was deemed desirable to reuse the inverted list and all the associated indexing infrastructure, but redefining the ranking mechanisms. While as many an open source project it is sparsely documented aside from its basic usage, it nonetheless boasts a a strong user community ready to answer questions on its specifics and is well designed to allow developing extensions.

For speed, ease of implementation, and scalability, the remaining portions of the indexer module are implemented in Hadoop, a parallel distributed framework that follows the MapReduce processing paradigm. This fits the task at hand because each resume can be dealt with independently from all the remaining resumes and therefore it is amenable to be split and executed in a farm topology, where the servers work with minimal communication among each other.

After setting up a working prototype with satisfying retrieval accuracy, improvements were necessary to see the system scale. As described in the preceding chapter, the process of generating suggestions for a given candidate entails two queries and is very heavy in terms of resources if naively implemented. One initial query to find out the distribution of words associated to the current job title in the next position, and finally a query against the job offer index to arrive at the recommendations. There are several items that can be addressed to improve performance in order to obtain a usable system.

For example, in the simpler of the two proposed strategies, the output model from the first query depends only on the job title as a result of the Markov property and the resulting relevance model can be cached. Even though most job titles are unique, there

is a small subset of them that are repeated across a wide segment of the population. Caching the model for one candidate allows re-usage for other candidates, therefore the boost in run time performance is quite noticeable, shaving some hours off the daily batch process that generates recommendations. In a distributed setting, this component can be centralized to maximize the hit rate.

For this purpose, a NoSQL key-value storage medium was deployed. The business constraints when selecting a provider were that it operated in memory so as to maximize throughput, that it allowed persistent storage because the job title index typically does not change and models can be reused even if the system is rebooted, and that it supported direct binary storage to minimize marshaling and unmarshalling of data upon serialization time. After completion of a formal vendor comparison, it was decided to use Redis as it satisfied all the requirements, had adequate support, and it was still under active development at the time of writing. As an API with a Java port, Jedis was chosen, because it exposed the required functionality except for a binary connection pool, feature that was added to the implementation.

Additionally, to enhance throughput the inverted list indexes can be mapped directly onto memory so that retrieval and query times are faster. Latency from reading secondary storage is a bottleneck of many applications, and reading from main memory results in a performance improvement. Even though the operating system may resort to buffering of recently accessed segments by means of memory mapped files, a special purpose structure easily accessible within the application performed about 20% faster than delegating this task on the general strategies of the OS, if only by reducing communication overhead across processes.

Other than that, shall query answering times over one machine prove inadequate, special care must be exercised in relation to designing a distributed architecture to ensure that the system scales horizontally when adding new servers and that the performance increases approximately linearly. This is important because machines are generally much cheaper than programming resources, perhaps increasingly so. Sometimes flexibility takes precedence over precise fine tuning and it is important that the system is easily decomposed into replaceable pieces that play well with each other, allowing it to be deployed over a cluster as opposed to delivering the fastest possible sequential response times on one node.

Common approaches in the distributed domain include index partitioning across many servers or sharding. The idea is that a query can be formulated through a load balancer that distributes the work across a cluster in farm topology with no communication whatsoever among slave nodes, and each of them produces a partial result in isolation that is aggregated at the joint point of entry to arrive at a final answer after sorting and

possibly filtering individual results. Other times we can be sure that certain portions of the index are never going to be accessed at once by the queries and partitioning with full delegation of responsibility is called for. General purpose IR solutions such as Solr and Elastic Search support this mode of configuration out of the box.

Conversely, if the index is small, as is the case for the proposed job recommender where only job titles and job offer headers are being indexed, another possible action that can be undertaken is direct index replication over individual nodes. In this co-location scenario each node is fully able to generate a final result without involving the load balancer in the aggregation stage. This means that the intermediate proxy can handle incoming requests by fully delegating the task at hand at the beginning of the query, redirecting the client to an appropriate node and, as a result, is less likely to become a bottleneck as more processing nodes are added to the cluster. This is similar to the role taken by the Name Node in the HDFS architecture in Hadoop. Another desirable property is that data locality is preserved and thus queries are faster on average.

Aside from these advantages, shall the processing node fail, the full request will err and additional fail over mechanisms must be implemented depending on the criticality of the assignment being carried out. For a job recommender it is not that important that suggestions for a given user are not created due to network or hardware failure so long as the system administrator is notified and corrective steps can be taken to remedy the situation. However, it is important that the full system fails gracefully, without disrupting the end user experience. Appropriate exception handling must be exercised to fulfill this condition.

With all the above in mind, the layout of the query module is presented in fig. 7.4. There is a load balancer that acts as a point of entry redirecting the traffic to query resolving servers. It has a local cache in case the job title has been seen before, with a time to live (TTL) period of 24 hours. The rationale for a limited lifespan is that job offers may expire or new ones may be added to the index that must be taken under consideration for generating recommendations. Each individual server has a co-located Lucene index that can fully process the request, and caches the aforementioned model building query into a centralized storage shared by all query resolvers, resulting in a query dependent speedup of 2x to 10x, the actual time being conditioned by the number of documents accessed by the keywords in the model. More popular words will access larger portions of the index than infrequent ones and benefit more from caching, hence the run time disparities.

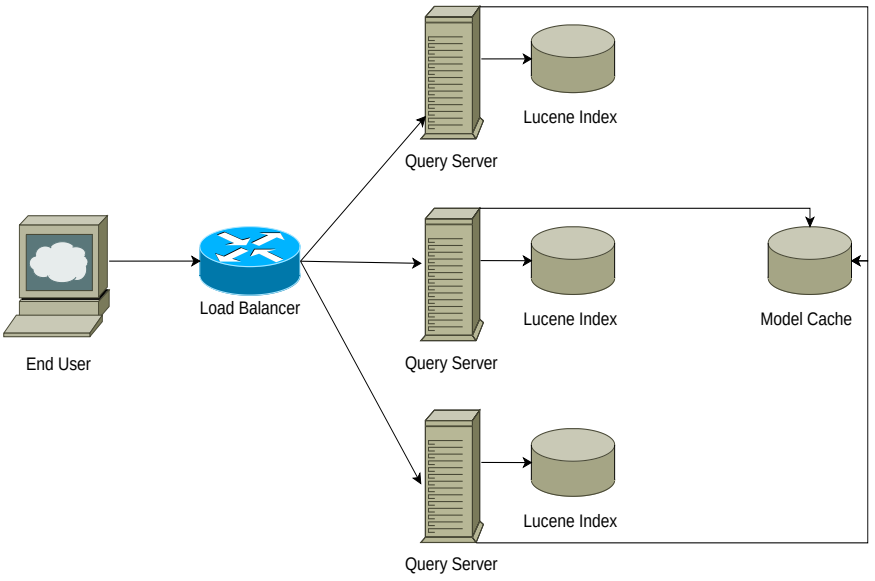

FIGURE 7.4: Physical architecture of the query module

The processing servers are spread out over seven specially dedicated machines hosted in the cloud, equipped with 64 GB of main memory. For the cache server, a lower end machine is used, because the expected usage is not as computationally intensive as in the processing servers. Redis was configured in LRU mode with a memory limit of 40 GB to ensure that the database does not grow out of manageable proportions, and at the same time prioritizing highly frequently accessed records so that they are not removed from the cache.

Feature	*Value*
Type	PX90
Provider	Hetzner
RAM	64 GB
HD	2x2TB
Cores	12
CPU	Intel(R) Xeon(R) CPU E5-1650 v2 @ 3.50GHz
Price	99 EURO/month

TABLE 7.1: Query server specifications

7.3 Architecture evolution

The previous section describes the initial version. What follows is an enumeration of constraints in the environment a job recommender belongs to that may not be immediately

Feature	*Value*
Type	EX6S
Provider	Hetzner
RAM	32 GB
HD	2x3TB
Cores	8
CPU	Intel Xeon E3-1245 Quadcore
Price	66.39 EURO/month

TABLE 7.2: Redis server specifications

apparent to the uninitiated in the Human Resources field and that have conditioned the subsequent development work.

- Most resumes are stale and do not change. End users switch jobs every two or three years and a majority of the time they are what a head hunter would describe as a passive job seeker. Not strictly looking, but receptive to a suitable offer shall it come along. Therefore it is not uncommon for months and years to pass between resume updates. This has the straight implication that the query in the job title index and the resulting relevance model can be precomputed in advance and stored for future use, with significant savings in execution time. In addition, if following the Markov property where the next job strictly depends on the direct predecessor, the number of models to be precomputed is lower than the number of users, which is a desirable precondition.

- There are many active job offers at any given point in time, counting in the millions, but there are not that many being created every single day. At the time of writing this figure stands at 30.000 new job offers per day. If instead of performing recommendations by following the user to offer relationship we do it the other way around, from new offers to users and keeping an up to date historical cache with matches, a substantial amount of computation can be saved. This reasoning lies in the fact that users far outnumber new offers and that there are additional geographical restrictions that limit the number of matches in the result set. For instance, a particular user may wish to settle for a job within his province due to family constraints, and is unavailable for country wide opportunities.

Work is under way to derive a more scalable solution going forward through the reversal of the recommendation pipeline. The idea is to have a virtual mailbox per user with all the recommendations available and updated in real time, so that whenever the user logs in to the system or the newsletter batch process wants to access the module, the

recommendations can be retrieved expediently without constantly hitting the job opening index. The process is possible because resumes are rarely updated and candidates far outnumber incoming job openings per day. The expectation is that the same quality of results can be obtained with a one to two orders of magnitude speedup or the corresponding decrease in the number of servers required to run the recommender pipeline. The new design, which follows the lambda architecture of Marz [52], also permits that, aside from responding in real time to the users with recommendations, a heavy weight and supposedly more accurate model can be run behind the scenes, as much of the work can be precomputed and reused repeatedly.

Chapter 8

Empirical Results

For the algorithm benchmarking we resort to both the Cranfield methodology as well as reporting user behavior upon deployment in an online setting. Since the application has been delivered in a real world scenario, measurements can be derived from actual user behavior when interacting with the recommendations in the web site and email newsletters.

8.1 Static evaluation

In the first instance, the human resources department of the company invested in manually categorizing a small subset of job openings and trying to find the most appropriate one for a selection of candidates. This data set, while useful in itself as a benchmark, suffers from some inconveniences.

There are 389 candidates to be matched to 126 jobs. Among these, up to three job openings per candidate have been selected as an ideal match for each person, while it is often the case that there may be in fact several available within the collection. There are 658 such matchings in the database. As a result, the precision reported probably understates performance in a real world setting and cannot be directly compared with related publications. The structured profile of the candidate can be seen as a query against the information provided for the job openings.

The test set is composed of several human curated XML files, one for candidates, another for job openings, and finally an expected matchings file that relates both of them. An excerpt from the candidates.xml file is shown in fig. 8.1.

```
<employments>
      <employment>
        <position>Legal Council &amp;amp; Adviser</position>
        <employer>Auxideico Gestión</employer>
        <sector>Real Estate</sector>
        <highlights>Asesor legal en las operaciones de consultoría de gestión
        de activos.</highlights>
        <start_date>2013-02-04</start_date>
        <end_date></end_date>
      </employment>
      <employment>
        <position>Ejecutivo Comercial y Desarrollo de Negocio</position>
        <employer>EUROPA PRESS</employer>
        <sector>Medios de Comunicación</sector>
        <highlights>Ejecutivo Comercial y Desarrollo de negocio en el
        Departamento Comercial y Relaciones Institucionales.</highlights>
        <start_date></start_date>
        <end_date></end_date>
      </employment>
</employments>
```

FIGURE 8.1: Candidates data file

A parser was programmed to process all files and to store the job openings in Lucene digestable format, to be able to use them from the job recommender. Code was also created to query the recommendations generated and output the matchings in TREC format. Finally trec_eval was employed to generate metrics for the whole process and statistical tests were carried out in R.

Because of its small size, the data set was not involved in hyper parameter selection and was relegated to the final stage of testing to compute metrics. Rather, the approximation used for model tuning focused on deriving relevance information directly from resumes. A certain portion of job transitions was reserved for training and the remainder was used as a validation set, to adjust tunable configurations. The validation set was arrived at through reservoir sampling [96], an algorithm to select a uniform sample of elements from a stream with constant memory requirements.

The algorithms that were compared are the following:

- TF-IDF searching with the candidate's current job title against the full job opening, which includes the title and the description.

- TF-IDF searching with all the candidate's previous positions against the full job opening.

- BM25 with the candidate's current job title against the full job opening. K was set to 2 and b to 0.5. The parameters where tweaked with a grid search in the held out transitions database.

- BM25 with all the candidate's previous positions against the full job opening. K was set to 2 and b to 0.5.

- JM-RM1, a conditional relevance model over job titles with Lavrenko's Method 1, ranking with the probability ratio and Jelinek-Mercer smoothing. The matching field is the title of the job opening only. The number of documents for the pseudo relevance feedback round was set to 350, $\lambda_s rc$ was set to 0.9999 (the smoothing parameter for the source title) and $\lambda_d st$ was set to 0.1 (smoothing parameter for the destination title).

- RE-RM1, a conditional relevance model over job titles with Lavrenko's Method 1, ranking with the probability ratio and relative error smoothing. The number of documents was once again set to 350 and $\sigma^2 = 0.01$. $\lambda_s rc$ was set to 0.9999 while $\lambda_d st$ was adjusted dynamically as a function of σ^2.

Algorithm	*MAP*	*NDCG*	*R-Precision*	*Reciprocal Rank*
TFIDF single	0.1596	0.2507	0.1048	0.1811
TFIDF full	0.1604	0.2888	0.0934	0.1839
BM25 single	0.1613	0.2527	0.1072	0.1879
BM25 full	0.1632	0.2914	0.1035	0.1910
JM-RM1	0.1566	0.3133	0.0971	0.1851
RE-RM1	0.2003	0.3509	0.1199	0.2233

TABLE 8.1: Test run metrics report

Given the limited number of documents used for evaluation purposes, all algorithms were modified to compute the priors from a different job opening index with expired offers. The relevance model implementation computed the priors based on the transitions of the resume database instead. Tests of the five a priori hypotheses were conducted using Holm adjusted alpha levels to compare NDCG metrics over the given queries, a less strict criterion than Bonferroni. Adjusted p-values are reported. The procedure employed was a pairwise Wilcoxon test. Results indicated that the average NDCG was significantly higher for the RE-RM1 model over TF-IDF single (p-value $<$ 2e-16), TFIDF full (p-value = 2.3e-05), BM25 single (p-value $<$ 2e-16) and BM25 full (p-value = 3.8e-05). No significant relationship was found with regards to JM-RM1 (p-value = 0.08), therefore there is not enough evidence to claim that relative error smoothing is superior to Jelinek-Mercer smoothing for the task given the data gathered from the experiment. It is also

posible that the improvement, if present, is only applicable to very short documents as the ones treated, because otherwise the distortion introduced for content bearing words can nullify its protective effects against random unrelated words.

Figure 8.2 shows the trade off between precision and recall for the queries in the test set. Under normal circumstances, precision would fall to zero at the maximum point of recall. However, in this case for many queries all relevant documents manually tagged as such were returned and the drop off is less pronounced. In practical scenarios this is probably unrealistic as there are bound to be many more non-relevant documents than relevant ones, and this is merely a byproduct of the data used, whose limitations have already been mentioned.

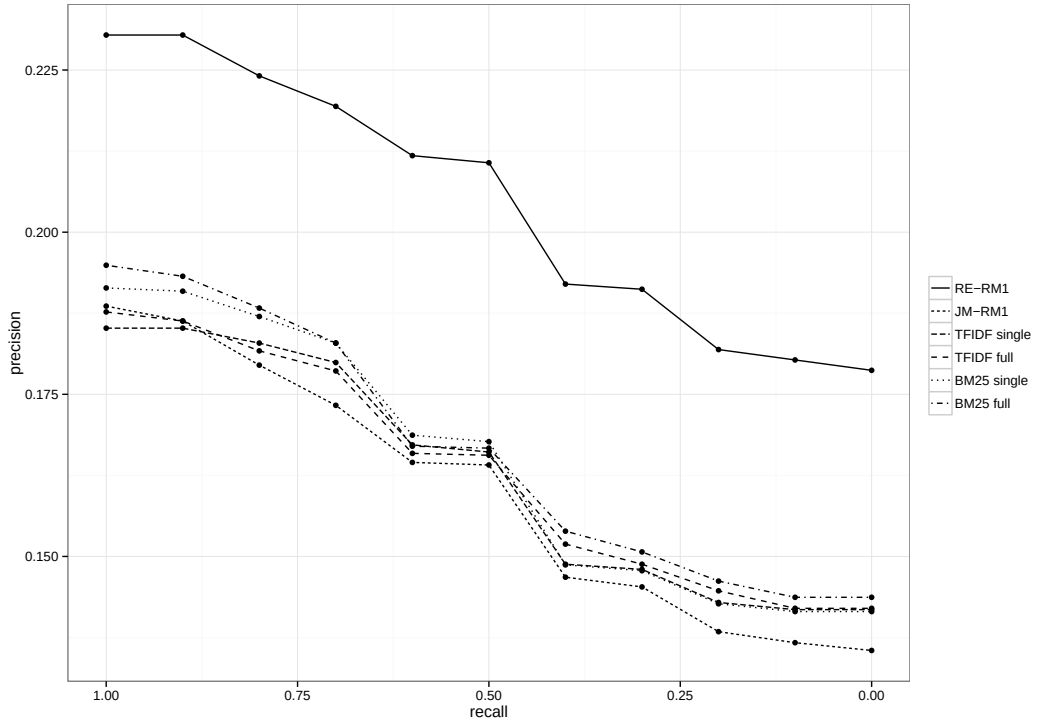

FIGURE 8.2: Precision-Recall curve over the internal HR data set

8.2 Dynamic evaluation

In this section, some empirical results are reported that point to the usefulness of the method proposed in a real world setting. The conditions in the production environment difficult greatly extracting definite conclusions in a purely scientific manner, as no truly randomized controlled trial can be carried out in the web server, because offers change from day to day, users differ in their information needs, there are seasonal and weekly patterns that must be accounted for, there are often unpredictable bugs in the pipeline or in the web application, etc. While some attempts can be applied to control for each of these individual effects, it is complicated to control for all causes that influence user behavior and that could alter metrics. This is an area of improvement in the current work, although hopefully some intuitions about performance can be ascertained from aggregate metrics when available.

At any given point in time, there are several algorithms generating a variable number of recommendations that are mixed together in an undetermined way and are displayed to the user. Offers are presented across several channels. A candidate can receive job openings that purportedly match his profile either through email, push notifications in a mobile application shall he have it installed, or finally he can browse for them in a dashboard, where again the presentation varies according to the medium on which it is displayed. As shown in fig. 8.3, mobile users get to see a compact view of the dashboard where they can easily scroll up and down the list, accepting and discarding offers, while the user interface for the web application is slightly more contrived due to technical limitations that preclude a more seamless user experience.

All these various channels show different sets of information about a job opening. For example, in the dashboard the user visualizes a ranking of job openings, and he may only see the name of the company and the title of the offer. He can either apply directly without looking further, or he can request more information by clicking to see the details of the job opening. In this case salary information is displayed if available, along with the set of required skills the employers ask for, and an extended description about the position.

Conversely, when sending information by email, there are a different number of campaigns that run routinely each week. One such email is the job opening invitation newsletter, that is sent twice a day to each candidate if there are new recommendations available. Instead of displaying just the job title and the company, the content of the email contains roughly the same amount of information that the user would see when he clicks through to the landing page, except for the fact that the description is limited in the number of characters that are rendered in the email. Once again, the possible

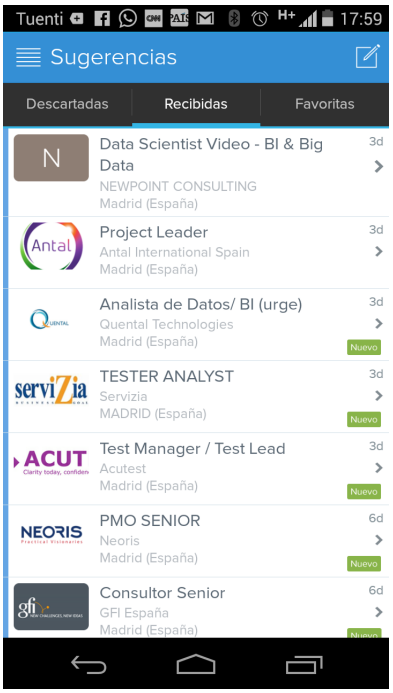

FIGURE 8.3: Mobile application dashboard view

actionable items are either apply directly in the email or visualize the details of the job opening.

Each user action triggers an event that is recorded to compute different statistics. As can be seen from the two scenarios shown in figs. 8.3 and 8.4, there are semantic differences in apparently similar user actions. In principle, an application that is performed when visualizing more information about the job opening carries a higher signal in terms of inferred relevance from the candidate as to the quality of the recommendation. Aggregating applications as a global metric, while useful from a business perspective as a summarization device, misses useful nuances from an analytical perspective.

No positional information is recorded at this point in the dashboard, all that is tracked is that a recommendation was rendered and which algorithm generated it. This limitation hinders the capacity to use it as a source of relevance from a dynamic perspective. Ideally user tracking through Javascript should be performed, to make sure that a recommendation that has been rendered has been actually read. One such way is to track the scrolling behavior of the browser, reporting the pixel of the page that was reached when browsing the site. This assumes that there are more recommendations available that do not fit initially in the user display, and that browsing is required to visualize them.

Other idea that can be used is to mark as read all job openings above the last user click, under the cascade assumption of browsing behavior. In this model the user is presumed

FIGURE 8.4: Screen capture of a job opening invitation

to see results sequentially scrolling from top to down. A click in a result low in the rankings presupposes that the user has actually read all job openings that are present above. While this is useful, this approach would tend to under count read offers, as the user may bounce directly from the initial page without clicking at any result, and it is probably accurate that he has at least read some of them and was dissatisfied with the recommendations. Attribution of read offers from bounce rates would have to be performed, as otherwise the read offers will be biased, misrepresenting all metrics that are based on this statistic.

From all sources of recommendations in the company, perhaps the best one to track algorithmic performance is the recommendations being sent by email. Unlike search engine rankings, which contain several offers and where recording read status is problematic, in some of the email campaigns that are used at present only one recommendation is being sent. This allows to remove the positional bias that is usually seen in a search engine ranking. Furthermore, bulk email companies allow to track the open event of an email. This is normally performed through a tiny image or an script embedded in the email that triggers a request to the web server, allowing to record the user action. For users that either block Javascript or image loading in the email browser, once again under counting occurs. However, this bias is probably lower than the bounce rate problem mentioned earlier in a search engine ranking, at least before adjustment.

Because of these reasons, email tracking statistics are reported in this thesis as opposed to user actions on search engine rankings. Applications stemming from an email campaign can be accurately tracked through embedding information in the target URL or via HTTP cookies. By carefully measuring click through rates and conversion rates, and taking into account that there are bound to be variations due to the differing volume

of recommendations generated by each algorithm, we can gain an understanding about actual performance.

What follows is a description of the algorithms that existed prior to my arrival at the company, which are briefly described describe, highlighting their strong and weak points. Later on, comparison figures are drawn to assess their performance relative to the proposed solution:

- *Constraint based query in the Elastic Search index or direct search algorithm.* The purpose of this method is to provide high recall at the expense of accuracy. All offers that match a set of restrictions, such as salary requirements of the candidate, functional area and geographical restrictions are returned. Run time performance along with high job opening coverage are probably the main advantages relative to other more complex algorithms.

- *Expert derived ontology approach (Job position algorithm).* This is an evolution of the previous approach, still based on hand tailored queries. It incorporates the job title along with synonyms, positive words that can be present, as well as negative keywords that cannot appear, along with restrictions on years of experience. The value of TF-IDF in retrieval given the short length of the job title is questionable in any case. Precision is high although given that more keywords have been incorporated into the mix, recall is improved over the usage of job titles alone. Much effort has been devoted to this approach company wise, leveraging the experience of the headhunters in the firm. Its virtues constitute shortcomings as well. Given that queries are hand tailored by humans, maintenance is time consuming and the method does not scale very well going forward.

- *Areas of activity algorithm.* Upon registration a user self-selects himself and chooses some fields of work along a predefined hierarchy. Manually crafted queries are then ran that depend both on the user's area and the words present in the resume. For all intents and purposes, this is rather similar to the previous approach, with the added noise by users selecting categories that may not be suited to their expertise and that merely sound appealing.

- *Job Application algorithm*: This is an attempt to include dynamic behavior into the recommender pipeline by adapting to user behavior over time. The idea is to try to find a job opening with a similar title to one of the last five job applications of a given candidate. This way, if a candidate is applying to novel areas that are out of scope of what normally would be considered for him given his resume content, the system tries to pull offers in accordance with his last actions. It performs well though with a lower level of volume. Bear in mind that the first experience

of the user is fully determined by pure content approaches, and it was measured by the business intelligence department that users active in the first session, that is right after sign up, tend to have a higher recurrence over time, although it is also possible that by being active at the beginning, they self select themselves into a user group with intrinsic higher levels of activity than the overall users of the application.

- *LDA / K-Means Pompeu Fabra university algorithm.* The system first takes a collection of job offers and, through unsupervised learning, arranges them in several clusters, ninety at the moment. As in Bag of Word methods in image classification, that also rely in k-means clustering as well in a preprocessing stage, the choice of k is likely to be important and can impact the performance of the algorithm, therefore it is amenable to tuning. The idea of using offers instead of resumes is that offers are more uniform content wise, whereas resumes may display U-turns in terms of career paths, with candidates going back to school to retrain for different occupations and such, and thus the offers are believed to display strong topical consistency. It is questionable though to assume that the terms and expressions present in the job offers must match the language of the resumes themselves.

  Once the clusters are inferred, a supervised algorithm is trained to predict the cluster from a vectorized representation of the document, at the moment this algorithm being support vector machines. Then the classifier is applied to each resume for which we would like to arrive at recommendations, and document ranking is restricted among documents whose cluster matches that of the offer. The ranking method can be switched back and forth among any IR traditional option, either the TF-IDF vector state model, query likelihood (Naive Bayes), divergence from randomness, document likelihood or any other probabilistic method. The internals of the algorithm have been described elsewhere [69]. Nonetheless, the present status of this algorithm is that it does not perform very well and it has been switched off, therefore comparison will be restricted to the remaining group already mentioned.

The process for generating recommendations follows both a real time and batch setting. Recommendation triggers function both in a push and a pull setting. If a user updates his resume, salary or geographical preferences through the web site, recommendations are generated immediately. Otherwise, his recommendations are generated offline and delivered throughout the day. In this latter instance there is a queue that directs the batch processing for all users in the platform. It is filled at midnight and it is emptied throughout the day. Thus it is important that the batch generation ends in a twenty four hour period, because otherwise it would grow indefinitely. This process typically

	WK 31	*WK 32*	*WK 33*	*WK 34*	*WK 35*	*WK 36*	*WK 37*
TOTAL	368,247	400,846	404,221	396,983	422,879	535,988	843,043
SENT	315,319	346,553	348,590	341,722	366,251	472,453	564,862
OPEN	81,232	84,735	84,428	83,824	88,739	115,310	120,662
CLICK	18,802	19,902	20,470	21,455	23,173	30,054	32,223
OPEN/SENT	25.76%	24.45%	24.22%	24.53%	24.23%	24%	21%
CLICK/SENT	5.96%	5.74%	5.87%	6.28%	6.33%	6.36%	5.70%
CLICK/OPEN	23.15%	23.49%	24.25%	25.60%	26.11%	26.06%	26.71%

TABLE 8.2: Global newsletter statistics

consumes several hours and once suggestions are generated they are handed to another system that enqueues the tasks for sending out emails. The architecture of the full system is out of the scope of this document, as we presently focus on the recommender, which has been the subject of my work for the last few months.

After initial tests, including but not limited to the static evaluation mentioned in the preceding section, hopes in the company were bullish and it was decided that the system was good enough to be rolled out to production. The introduction was done in August 2014, initially to a very small subset of 5% of the users because the server was overloaded, increasing the load iteratively to 40%, 60% and finally all users in the platform as the distributed setup was finalized to arrive at a final setup with five servers, the process being aided with the incredibly helpful support of my coworkers. In tables 8.2 and 8.3 several key metrics of the newsletter response rates are displayed. Read status is ascertained through embedded images in the email, and may under report the actual emails being read, because some email clients block images from displaying. Absent any biases in the percentage of opened emails that are not captured from week to week, the results show a steady improvement both in the ratio of clicked to read emails as a result of increased precision and higher volume as a result of increased recall.

One item that must be commented on is that the click through rates (CTRs) for Spain are higher and the difference is statistically significant. This is to be expected as the model was initially trained on resumes from Spain and applied elsewhere. Notice that it was also applied in the United Kingdom, and although there are many resumes in the data set of 2 million career trajectories used to construct the model, the majority of them are in Spanish and thus the accuracy in English suffers.

Additional tracking was performed by Mixpanel, a third party web analytics vendor. An unfortunate occurrence was that we migrated from Kissmetrics to Mixpanel while the roll out was being performed and lost some data during the process, here we report the differences between Week 36 to Week 37, where the improvement is noticeable. Aside from higher CTRs, metrics showed improvements in several other measures, as the users

	WK 31	*WK 32*	*WK 33*	*WK 34*	*WK 35*	*WK 36*	*WK 37*
TOTAL	261,638	272,970	273,410	249,812	271,392	356,869	543995
SENT	224,632	235,718	235,543	213,242	234,333	315,015	364647
OPEN	58,580	56,785	56,294	50,163	55,812	75,380	79188
CLICK	13,757	12,858	13,438	11,967	14,608	19,944	21660
OPEN/SENT	26.08%	24.09%	23.90%	23.52%	23.82%	24%	22%
CLICK/SENT	6.12%	5.45%	5.71%	5.61%	6.23%	6.33%	5.94%
CLICK/OPEN	23.48%	22.64%	23.87%	23.86%	26.17%	26%	27.4%

TABLE 8.3: Spain's newsletter statistics

that clicked on a job offer were also more likely to apply for a job afterwards. This can be gleaned from the fact that although clicks on offers increased by 77.63% on a week to week basis, applications did by 101.54%, for a 13.4% improvement in the apply to click ratio, which is an indirect proxy for recommendation relevance. The raise is more prononuced if we restrict it to unique visitors as opposed to page views, where the ratio rose by 19.1%. Overall engagement figures over the site were impacted as well, because the ratio of visits to unique visitors across all pages increased by 7%.

Algorithm	*Generated*	*Read*	*Applications*	*Discards*	*ATR*	*DTR*
Direct search	25,171,112	1,377,981	52,112	198,374	3.8 %	14.2%
Areas	906,205	88,357	2,866	7,540	3.2%	8.5%
Job Position	748,311	120,679	9,510	7,530	7.9%	6.2%
Job Application	1,077,869	204,331	27,628	19,165	13.5%	9.4%
Relevancio	1,662,775	316,086	24,587	19,688	7.8%	6.2 %
Internal Relevancio	8,745,162	992,207	61,068	84,265	6.2%	8.5%

TABLE 8.4: Weekly applications

Starting on March this year, the company has also started tracking more finer grained statistics about the performance of the different algorithms. Metrics for the week from March 23th to March 29th are reported in table 8.4. Since the user has either the possibility to apply to a given suggestion or to discard it, giving feedback that can be used for learning, two types of conversion rates are tracked. The first one, the ATR or application through rate is positive, while the DTR or discard through rate is negative and it should be minimized. In the table we can see that algorithm effectiveness varies wildly. The direct search or database algorithm is not very effective, as evidenced by the high DTR, although it drives a sizeable number of applications. The job application algorithm shows both high ATRs and DTRs, a fact that is explained by its bias towards more active users. Finally, the relevance models approach, in its two incarnations, one for internal job openings and another for external job openings, drives more than half of the total number of applications and has a relatively high ATR given the recommendation volume, along with the lowest DTR of the compared algorithms.

Chapter 9

Conclusions and Future Work

This thesis presented an application of Relevance Models to eliciting job recommendations from information available in resume databases. Whereas the framework is applied in Information Retrieval to search documents, the idea was extended and applied to transitions in job titles. For the analogy to work, rather than indexing and retrieving words present in a document, we indexed words appearing in the job title immediately over the one under consideration while parsing resume information. One of the requirements of the project was to be able to recommend job openings even in cases where the job title had never seen before, which was attained by finding a close neighborhood through TF-IDF and combining the transition data to arrive at a unified relevance model.

While the method works well in many instances, other times it does not. Important context information is lost if job transitions are restricted to depend exclusively on the preceding job title. One possible idea to explore is to reintroduce this context information through the smoothing function after applying topic modeling. In a paper submitted to SIG IR in 2006, Wei and Croft show a way to introduce information from topics detected by Latent Dirichlet Allocation that increases retrieval performance over relevance models across several collections [99]. Other idea that can be explored in this area is dimensionality reduction through latent factor models like those of Restricted Boltzmann Machines to arrive at better neighborhoods.

In addition, the formulation from relevance models permits tweaking a priori probabilities to give more weight to some documents in the final model. Some months ago I was involved in a project that downloaded data from Glassdoor to predict salaries from unigrams and bigrams present in job titles. It is quite possible to exploit models like these to fine tune the weight of certain documents. One assumption was that people with the same job titles perform similar functions. As a general guideline it is roughly true, but it is clearly not the case in many instances. Real world examples are the following two

transitions: "CTO" → "intern" and "CTO" → "CIO". It is evident that they represent two very different individuals. Under the assumption that a person transitions from a lower wage to a higher wage when changing jobs, unless the switch was involuntary, we can clearly see that in the first example a junior candidate was maybe working at a start up and opted for using a glamorous job title that is usually associated with someone of a certain seniority and rank. In the second example, maybe the person is an established professional, middle aged, and works in a medium or a large sized company. Salary estimates, years of experience, and company volumes can be determined and affect the neighborhood weights to improve system performance. For example, a sigmoid function can be fit whereby an applicant is likely to apply if the estimated salary of the offer is higher than his own, and unlikely if it is deemed considerably lower. Under the premise that in transitions salary grows, the salary associated to a transition is the lowest of the one estimated by the two positions involved. Henceforth, the presence of "intern" would penalize the outlier just mentioned and it would not weigh much when generating recommendations for the remaining CTOs.

Other item to consider is that the attractiveness of a position to a candidate depends on several factors besides the words present in the job title or the offer description. Ever since the study on the subject by Moore over a century ago [60], economists are aware that large companies, aside from being less prone to bankruptcy, typically offer better working conditions than small companies. This is known in the literature as the firm size wage effect, and its cause is disputed among economists today. Be it as it may, candidates are well aware of this fact and apply in droves based on perceived brand equity.

Similarly, a candidate may be attracted to a company where it is known in the market that it is very difficult to get hired, because in doing so he communicates his ability to future employers by sending a signal [88], and this fact may work to his advantage when negotiating a higher salary later on in his career. Proxies for prestige that can be easily obtained are number of employees and revenue. A probabilistic interpretation of this data is likely to lead to higher recommendation satisfaction from the point of view of the end user, although it must be approached with care, in order not to alienate small employers that would get an insufficient response rate to their advertisements if implemented naively.

Finally, the feedback that is gathered from the user could be used to build a learning to rank framework, where the relevance models for the job title constitute one among many different signals that are weighed to arrive at a list of recommendations. Building such a framework would allow to grow in features faster, reducing human judgment in the process by tuning weights automatically. One of the problems of the recommender

pipeline at present is that there are many different algorithms working independently, each generating a set of recommendations, while they could work together and their results reflect different aspects of relevance that are combined into a global ranking function.

Bibliography

[1] Nasreen Abdul-Jaleel, James Allan, W Bruce Croft, Fernando Diaz, Leah Larkey, Xiaoyan Li, Donald Metzler, Mark D Smucker, Trevor Strohman, Howard Turtle, et al. Umass at trec 2004: Notebook. In *The Thirteenth Text Retrieval Conference (TREC 2004) Notebook*, pages 657–670, 2004.

[2] George A Akerlof. The market for" lemons": Quality uncertainty and the market mechanism. *The quarterly journal of economics*, pages 488–500, 1970.

[3] Khalifeh AlJadda, Mohammed Korayem, Camilo Ortiz, Chris Russell, David Bernal, Lamar Payson, Scott Brown, and Trey Grainger. Augmenting recommendation systems using a model of semantically-related terms extracted from user behavior. *arXiv preprint arXiv:1409.2530*, 2014.

[4] Gianni Amati and Cornelis Joost Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Transactions on Information Systems (TOIS)*, 20(4):357–389, 2002.

[5] Erik Anderson, Andrew Evans, Brian Farmer, Leslie Ferry, Pam Koczara, Robert J McGovern, Howard Rosen, and Brent Smith. Match-based employment system and method, September 28 2010. US Patent 7,805,382.

[6] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[7] Christian Bizer, Ralf Heese, Malgorzata Mochol, Radoslaw Oldakowski, Robert Tolksdorf, and Rainer Eckstein. The impact of semantic web technologies on job recruitment processes. In *Wirtschaftsinformatik 2005*, pages 1367–1381. Springer, 2005.

[8] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[9] Alan B Cayton, J Brandt Hamby, and J Kevin Leonard. System and method for automated screening and qualification of employment candidates, February 24 2009. US Patent 7,496,518.

[10] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 621–630. ACM, 2009.

[11] Joanne Charles. Finding a job on the web, 2000. URL http://www.blackenterprise.com/mag/finding-a-job-on-the-web/.

[12] Pei-Yu Chen, Shin-yi Wu, and Jungsun Yoon. The impact of online recommendations and consumer feedback on sales. *ICIS 2004 Proceedings*, page 58, 2004.

[13] Cyril Cleverdon. The cranfield tests on index language devices. In *Aslib proceedings*, volume 19, pages 173–194. MCB UP Ltd, 1967.

[14] Cyril W Cleverdon and Michael Keen. Aslib cranfield research project-factors determining the performance of indexing systems; volume 2, test results. 1966.

[15] William S Cooper. The inadequacy of probability of usefulness as a ranking criterion for retrieval system output. *University of California, Berkeley*, 1971.

[16] Science Daily. Big data, for better or worse: 90% of world's data generated over last two years. sciencedaily, 2013. URL http://www.sciencedaily.com/releases/2013/05/130522085217.htm.

[17] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *To appear in OSDI*, page 1, 2004.

[18] Christian Desrosiers and George Karypis. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook*, pages 107–144. Springer, 2011.

[19] Peter A Diamond. Wage determination and efficiency in search equilibrium. *The Review of Economic Studies*, 49(2):217–227, 1982.

[20] Maryam Fazel-Zarandi and Mark S Fox. Semantic matchmaking for job recruitment: an ontology-based hybrid approach. In *Proceedings of the 8th International Semantic Web Conference*, 2009.

[21] Gartner. Gartner says big data creates big jobs: 4.4 million it jobs globally to support big data by 2015, 2013. URL http://www.gartner.com/newsroom/id/2207915.

[22] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[23] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[24] Stephen P Harter. A probabilistic approach to automatic keyword indexing. part ii. an algorithm for probabilistic indexing. *Journal of the American Society for Information Science*, 26(5):280–289, 1975.

[25] Adam Hyder and Changsheng Chen. Intelligent job matching system and method, May 25 2006. US Patent App. 11/441,997.

[26] Yul J Inn, Ka L Leung, David Sobotka, and Lance Tokuda. Method and apparatus for automatic categorization of applicants from resumes, March 23 1993. US Patent 5,197,004.

[27] John Ioannidis and Thomas A Trikalinos. Early extreme contradictory estimates may appear in published research: The proteus phenomenon in molecular genetics research and randomized trials. *Journal of clinical epidemiology*, 58(6):543–549, 2005.

[28] John PA Ioannidis. Why most published research findings are false. *PLoS medicine*, 2(8):e124, 2005.

[29] John PA Ioannidis. Why most discovered true associations are inflated. *Epidemiology*, 19(5):640–648, 2008.

[30] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[31] Anjali Ganesh Jivani et al. A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, 2(6):1930–1938, 2011.

[32] Jobvite. 2012 social job seeker survey, 2012. URL http://web.jobvite.com/rs/jobvite/images/Jobvite_JobSeeker_FINAL_2012.pdf.

[33] Karen Sparck Jones and RG Bates. *Report on a design study for the'ideal'information retrieval test collection*. Computer Laboratory, University of Cambridge, 1977.

[34] Mostafa Keikha, Jangwon Seo, W Bruce Croft, and Fabio Crestani. Predicting document effectiveness in pseudo relevance feedback. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2061–2064. ACM, 2011.

[35] 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.

[36] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[37] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[38] Kory Kroft and Devin G Pope. Does online search crowd out traditional search and improve matching efficiency? evidence from craigslist. *Journal of Labor Economics*, 32(2):259–303, 2014.

[39] Robert Krovetz. Viewing morphology as an inference process. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 191–202. ACM, 1993.

[40] 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.

[41] John Lafferty and Chengxiang Zhai. Probabilistic relevance models based on document and query generation. In *Language modeling for information retrieval*, pages 1–10. Springer, 2003.

[42] 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.

[43] 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.

[44] Xiaoyan Li. A new robust relevance model in the language model framework. *Information Processing & Management*, 44(3):991–1007, 2008.

[45] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

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

[47] Xiaoyong Liu and W Bruce Croft. Cluster-based retrieval using language models. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 186–193. ACM, 2004.

[48] Hans Peter Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, 1(4):309–317, 1957.

[49] Yuanhua Lv and ChengXiang Zhai. Lower-bounding term frequency normalization. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 7–16. ACM, 2011.

[50] 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.

[51] Melvin E Maron and John L Kuhns. On relevance, probabilistic indexing and information retrieval. *Journal of the ACM (JACM)*, 7(3):216–244, 1960.

[52] Nathan Marz and James Warren. *Big Data: Principles and best practices of scalable realtime data systems*. O'Reilly Media, 2013.

[53] 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.

[54] David RH Miller, Tim Leek, and Richard M Schwartz. A hidden markov model information retrieval system. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 214–221. ACM, 1999.

[55] David Mimno and Andrew McCallum. Modeling career path trajectories. 2008.

[56] Paul Mockapetris and Kevin J Dunlap. *Development of the domain name system*, volume 18. ACM, 1988.

[57] Alistair Moffat and Justin Zobel. Rank-biased precision for measurement of retrieval effectiveness. *ACM Transactions on Information Systems (TOIS)*, 27(1):2, 2008.

[58] Monster. Monster for brand media solutions, 2013. URL http://advertising.monster.com/brand/brand-monster/info1.aspx.

[59] Raymond J Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 195–204. ACM, 2000.

[60] Henry Ludwell Moore. *Laws of wages: An essay in statistical economics*. The Macmillan Company, 1911.

[61] Donald R Morrison. Patricia—practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM (JACM)*, 15(4):514–534, 1968.

[62] Deep Nishar. The next three billion, 2014. URL http://blog.linkedin.com/2014/04/18/the-next-three-billion/.

[63] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. 1999.

[64] Javier Parapar and Álvaro Barreiro. Promoting divergent terms in the estimation of relevance models. In *Advances in Information Retrieval Theory*, pages 77–88. Springer, 2011.

[65] Javier Parapar, Alejandro Bellogín, Pablo Castells, and Álvaro Barreiro. Relevance-based language modelling for recommender systems. *Information Processing & Management*, 49(4):966–980, 2013.

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

[67] Christopher A Pissarides. *Equilibrium unemployment theory*. MIT press, 1990.

[68] Playence. Playence - success story xing, 2014. URL http://playence.com/taiger2/wp-content/uploads/2014/08/Success_story_XING_.pdf.

[69] Marc Poch, Núria Bel, Sergio Espeja, and Felipe Navıo. Ranking job offers for candidates: learning hidden knowledge from big data.

[70] 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.

[71] Martin F Porter. Snowball: A language for stemming algorithms, 2001.

[72] Rachael Rafter, Keith Bradley, and Barry Smyth. Automated collaborative filtering applications for online recruitment services. In *Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 363–368. Springer, 2000.

[73] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[74] David Ricardo. On the principles of political economy and taxation. 1817.

[75] Stephen Robertson. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, 60(5):503–520, 2004.

[76] 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.

[77] Stephen E Robertson. The probability ranking principle in ir. *Journal of documentation*, 33(4):294–304, 1977.

[78] Stephen E Robertson and K Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information science*, 27(3):129–146, 1976.

[79] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *NIST SPECIAL PUBLICATION SP*, pages 109–109, 1995.

[80] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[81] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[82] SilkRoad. Top sources for hires 2014, 2014. URL http://pages.silkroad.com/rs/silkroad/images/SilkRoad-Source-Hire-2014.pdf.

[83] 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.

[84] Fei Song and W Bruce Croft. A general language model for information retrieval. In *Proceedings of the eighth international conference on Information and knowledge management*, pages 316–321. ACM, 1999.

[85] K Sparck-Jones and S Robertson. Lm vs pm: Where's the relevance? In *First Workshop on Language Modeling and Information Retrieval, Pittsburgh, PA*, 2001.

[86] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[87] Karen Sparck Jones, Steve Walker, and Stephen E. Robertson. A probabilistic model of information retrieval: development and comparative experiments: Part 1. *Information Processing & Management*, 36(6):779–808, 2000.

[88] Michael Spence. Job market signaling. *The quarterly journal of Economics*, pages 355–374, 1973.

[89] Martijn Spitters and Wessel Kraaij. Language models for topic tracking. *Language Modeling for Information Retrieval, W. Bruce Croft and J. Lafferty (eds.). Kluwer, Dordecht*, pages 95–124, 2003.

[90] George J Stigler. Information in the labor market. *The Journal of Political Economy*, pages 94–105, 1962.

[91] Paul Switzer. Vector images in document retrieval. *Statistical association methods for mechanized documentation*, pages 163–171, 1965.

[92] Mortimer Taube et al. The uniterm coordinate indexing of reports. *The technical report: its preparation, processing and use in industry and government. New York, Reinhold*, pages 319–32, 1954.

[93] F Trichet, M Bourse, M Leclere, and E Morin. Human resource management and semantic web technologies. In *Information and Communication Technologies: From Theory to Applications, 2004. Proceedings. 2004 International Conference on*, pages 641–642. IEEE, 2004.

[94] C van Rijsbergen. Information retrieval: theory and practice. In *Proceedings of the Joint IBM/University of Newcastle upon Tyne Seminar on Data Base Systems*, pages 1–14, 1979.

[95] Jose Vega. Semantic matching between job offers and job search requests: Coling 90. In *Proceedings of the 13th conference on Computational linguistics-Volume 1*, pages 67–69. Association for Computational Linguistics, 1990.

[96] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[97] Jun Wang and Marcel JT Reinders. Music recommender system for wi-fi walkman. *Delft University of Technology*, 23, 2003.

[98] Jun Wang and Jianhan Zhu. Portfolio theory of information retrieval. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 115–122. ACM, 2009.

[99] Xing Wei and W Bruce Croft. Lda-based document models for ad-hoc retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 178–185. ACM, 2006.

[100] Null Defoor William Jr. Method for matching job candidates with employers, November 15 2001. US Patent 20,010,042,000.

[101] Jinxi Xu and W Bruce Croft. Query expansion using local and global document analysis. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 4–11. ACM, 1996.

[102] 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.

[103] Guido Zuccon, Leif A Azzopardi, and Keith van Rijsbergen. *The quantum probability ranking principle for information retrieval.* Springer, 2009.