Expert Systems With Applications 60 (2016) 169–182

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

RésuMatcher: A personalized résumé-job matching system

Shiqiang Guo, Folami Alamudun[*], Tracy Hammond

Department of Computer Science & Engineering, Dwight Look College of Engineering, Texas A&M University, College Station, TX, 77843, USA

A R T I C L E   I N F O

Article history:
Received 23 August 2015
Revised 13 April 2016
Accepted 14 April 2016
Available online 27 April 2016

Keywords:
Recommender system
Ontology
Résumé
Job search
Cosine similarity

A B S T R A C T

Online jobs search through popular websites are quite beneficial having served for many years as a prominent tool for job seekers and employers alike. In spite of their valuable utility in linking employers with potential employees, the search process and technology utilized by job search websites have not kept pace with the rapid changes in computing capability and machine intelligence. The Information retrieval techniques utilized by these websites rely primarily on variants of manually entered search queries with some advanced similarity metrics for ranking search results.

Advancements in machine intelligence techniques have enabled programmatic extraction of pertinent information about the job seeker and job postings without active user input. To this end, we developed a resume matching system, RésuMatcher, which intelligently extracts the qualifications and experience of a job seeker directly from his/her résumé, and relevant information about the qualifications and experience requirements of job postings. Using a novel statistical similarity index, RésuMatcher returns results that are more relevant to the job seekers experience, academic, and technical qualifications, with minimal active user input.

Our method provides up to a 34% improvement over existing information retrieval methods in the quality of search results. In addition however, RésuMatcher requires minimal active user input to search for jobs, compared to traditional manual search-based methods prevalent today. These improvements, we hypothesize, will lead to more relevant job search results and a better overall job search experience for job seekers.

As an alternative to the fragmented organization-centric job application process, job recruitment websites offered the promise of simplifying and streamlining the job search process. However, these websites offer limited functionality using generic and simplistic information retrieval methods, which being non-domain lead to a poor and frustrating search experience. In this paper, we present RésuMatcher, a personalized job-résumé matching system, which offers a novel statistical similarity index for ranking relevance between candidate résumés and a database of available jobs. In our experiments we show that our method offers a 37.44% improvement over existing information retrieval methods in the quality of matches returned.

© 2016 Elsevier Ltd. All rights reserved.

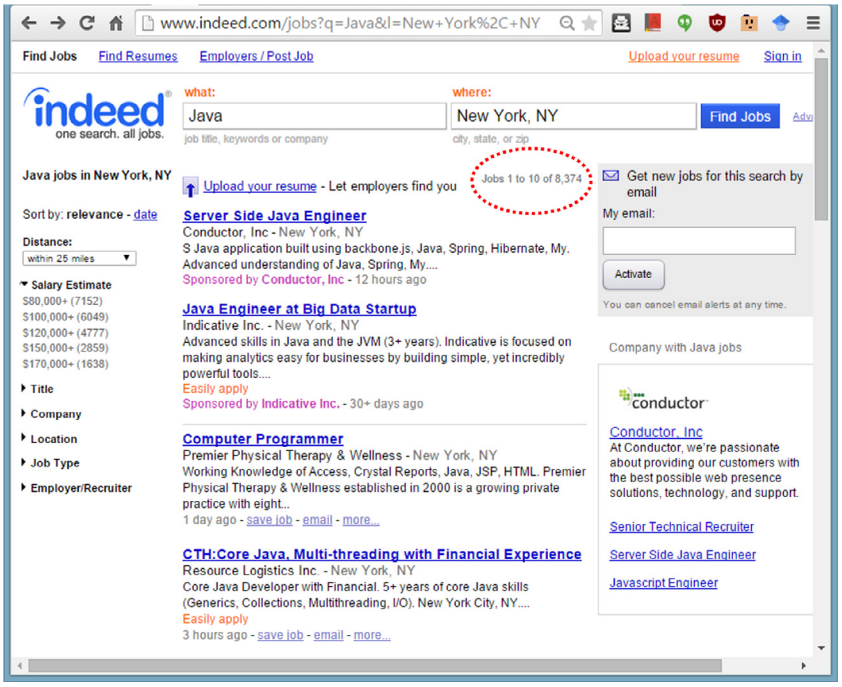
1. Introduction

Job recruitment websites serve as the major channel for job search for a majority of job seekers. While these job search websites (such as www.indeed.com and www.monster.com) serve to reduce the difficulty and duration of the job-search process, their search functionality is insufficient. Search functionality is limited to keyword based search, often resulting in poor, irrelevant search results. For example, a job search using the keyword “Java” to search for jobs within a limited geographical location (New York, NY) on www.indeed.com returned over 8000 jobs (Fig. 1). In this example, we observe that the number of search results is very large. Secondly, results are returned in an order that is of little value to the job seeker. The job seeker is left to comb through search results, a lengthy and tedious process, to examine each job description for relevance often resulting in information overload.

Search engines offered by recruitment websites are predominantly based on simplistic information retrieval methods, which match keywords to documents using data from an “Inverted index” (Zobel & Moffat, 2006). Documents matching the entered search query are sorted by order of importance as determined using a ranking algorithms such as Pagerank (Page, Brin, Motwani, & Winograd, 1999). While this methodology works very well for generic document retrieval (such as www.bing.com and www.google.com), results are less than desirable when applied to job search.

[*] Corresponding author.
E-mail addresses: pkushiqiang@tamu.edu (S. Guo), fola@cse.tamu.edu (F. Alamudun), hammond@cse.tamu.edu (T. Hammond).
URL: http://srl.tamu.edu (T. Hammond)

http://dx.doi.org/10.1016/j.eswa.2016.04.013
0957-4174/© 2016 Elsevier Ltd. All rights reserved.



www.indeed.com/jobs?q=Java&l=New+York%2C+NY

Find Jobs | Find Resumes | Employers / Post Job | Upload your resume | Sign in

indeed one search. all jobs.

what: Java (job title, keywords or company)
where: New York, NY (city, state, or zip)
Find Jobs

Java jobs in New York, NY

Upload your resume - Let employers find you

Jobs 1 to 10 of 8,374

Sort by: relevance - date

Distance: within 25 miles

Salary Estimate
$80,000+ (7152)
$100,000+ (6049)
$120,000+ (4777)
$150,000+ (2859)
$170,000+ (1638)

Title
Company
Location
Job Type
Employer/Recruiter

Server Side Java Engineer
Conductor, Inc - New York, NY
S Java application built using backbone.js, Java, Spring, Hibernate, My. Advanced understanding of Java, Spring, My....
Sponsored by Conductor, Inc - 12 hours ago

Java Engineer at Big Data Startup
Indicative Inc. - New York, NY
Advanced skills in Java and the JVM (3+ years). Indicative is focused on making analytics easy for businesses by building simple, yet incredibly powerful tools....
Easily apply
Sponsored by Indicative Inc. - 30+ days ago

Computer Programmer
Premier Physical Therapy & Wellness - New York, NY
Working Knowledge of Access, Crystal Reports, Java, JSP, HTML. Premier Physical Therapy & Wellness established in 2000 is a growing private practice with eight...
1 day ago - save job - email - more...

CTH:Core Java, Multi-threading with Financial Experience
Resource Logistics Inc. - New York, NY
Core Java Developer with Financial. 5+ years of core Java skills (Generics, Collections, Multithreading, I/O). New York City, NY....
Easily apply
3 hours ago - save job - email - more...

Get new jobs for this search by email
My email:
Activate
You can cancel email alerts at any time.

Company with Java jobs

conductor
Conductor, Inc
At Conductor, we're passionate about providing our customers with the best possible web presence solutions, technology, and support.

Senior Technical Recruiter
Server Side Java Engineer
Javascript Engineer

Fig. 1. Search results for JAVA keyword search on www.indeed.com.

Individual résumés contain information unique to each job seeker. This information can be used to build a candidate model, which when combined with a similar model created using information found in job descriptions, can provide a more useful method for ranking job search results. In this paper, we present RésuMatcher, a system which uses job seeker résumé as the primary query element for finding the most appropriate jobs.

RésuMatcher determines the appropriateness of a job by calculating the similarity between the candidate model and the job model, generated from the résumé and the job description respectively. This method changes the fundamental nature of job search from keyword-based search to model matching. Since search results are sorted by order of similarity score, our algorithm not only finds the most appropriate jobs, but also provides a ranking based on the similarity score (Gueutal & Stone, 2006). Intuitively, a higher similarity score indicates a more appropriate job for the respective job seeker, which we hypothesis will improve job application outcomes.

Unlike existing job search systems, where job seekers with different skills and experiences will retrieve the exact same results, RésuMatcher provides a search experience that is truly customized to job applicant since search results are based on the candidates past qualifications, skills, and prior work experience. The subsequent sections are organized as follows: Section 2 describes what has been done in terms of prior work. Section 3 gives an overview of our system, RésuMatcher, the Personalized Résumé-Job Matching System. In Section 4 we provide a detailed description of how we resolve challenges in information extraction. In Section 5, we describe the model similarity computations used in our system. Section 6 describes how to construct the Ontology and how to calculate the similarities in ontology. Section 7 provides results from the evaluation of the RésuMatcher system. Finally Section 8 shows the conclusions and future line of work.

2. Related works

Boolean search algorithms and filtering techniques are insufficient for the purpose of candidate-job matching requirement (Malinowski, Keim, Wendt, & Weitzel, 2006). Using such limited algorithms, a computing system is unable to understand job requirements and subsequently differentiate between required qualifications and preferred skills. As an alternative, recommender systems attempt to address this problem. Recommender systems are broadly accepted in various areas to suggest products, services, and information items to latent customers.

2.1. Recommender systems

For over a decade we have seen a marked increase in the number of websites dedicated to providing job search functionality. Job recommendation systems, a specialized area under a larger class of recommender systems, have also been the subject of numerous research studies. Wei et al. defined recommender systems for broad categories (Wei, Huang, & Fu, 2007).

2.1.1. Content-based recommender system

Content-based recommendation systems (CBR) analyze item descriptions to identify items of particular interest to the user. This type of recommendation system may be used in a variety of different domains such as web page recommendations, television programs, news articles, and social media content (Brusilovsky, Kobsa, & Nejdl, 2007; Singh, Rose, Visweswariah, Chenthamarakshan, & Kambhatla, 2010).

2.1.2. Collaborative recommender system

Collaborative filtering recommendation Systems (CFR). Collaborative filtering is a process of filtering or evaluating items using the opinions of others. Recommendation systems exploit this information to provide meaningful recommendations. Intuitively, this is something human beings have done for centuries - that is, sharing and making decisions based on the opinion of others (Rafter, Bradley, & Smyth, 2000; Schafer, Frankowski, Herlocker, & Sen, 2007).

2.1.3. Knowledge-based recommender systems

Knowledge-based recommendation systems (KBR) suggest products based on inferences about a user's needs and preferences. Inferences are derived explicitly from mapping product features with specific user needs (Burke, 1999; Lee & Brusilovsky, 2007; Trewin, 2000).

2.1.4. Hybrid recommender systems

Hybrid recommendation systems combine two or more recommendation techniques, each technique having its own strengths and weaknesses, to gain better performance and overcome drawbacks of any individual component. Collaborative filtering is often combined with some other technique in an attempt to avoid the ramp-up problem.

2.1.5. Job recommender systems

Rafter et al. designed case-based profiling for electronic recruitment (CASPER), which used both automated collaborative filtering (ACF) and personalized case retrieval (PCR) as a complementary service to improve the usability of the JobFinder web site search engine (Rafter et al., 2000). The system tracks user behavior within the JobFinder site, and constructs a user profile with which to generate personalized recommendations based on preferences of users with a similar profile. Färber et al. proposed a hybrid recommender system integrating two methods: content-based filtering, and collaborative filtering (Färber, Weitzel, & Keim, 2003). Their framework attempts to overcome limitations resulting from data sparsity by leveraging a combined model on.

2.2. Information extraction for job recommender systems

Information Technology companies face similar problems of information overflow with the large influx of résumés per job opening. Recruiters perform a manual screening process, which is tedious and time consuming. For this reason, each organization designs or purchases recruiting software systems to improve efficiency of the résumé screening process.

Amit et al. at IBM presented the "PROSPECT" system to shortlist candidates for job openings (Singh et al., 2010). Prospect uses a résumé miner to extract meaningful information from résumés using a conditional random field (CRF) model to segment and label the résumés. HP also built a system to solve the similar problem, which is introduced in Gonzalez et al.'s paper (Gonzalez et al., 2012). The system uses a layered information extraction framework to processing résumés. The goal of the systems built by IBM and HP is to help the companies to select good applicants, but cannot help job seekers to find appropriate jobs. For the mere fact that these systems are designed to meet company specific requirements means they cannot be generalized. This design approach places significant burden on job applicants since each applicant is required to visit individual company websites to perform a job search.

Yu et al. use a cascaded IE framework to extract information from résumés (Yu, Guan, & Zhou, 2005). Their framework uses Hidden Markov Model (HMM) for segmenting résumés into blocks. Subsequently, a support vector machine (SVM) is trained using these blocks to obtain the detailed information specific to each block. In this process, the framework is able to extract information such as name, address, and education. Finite-State Transducers (FST) have also been applied for pattern matching and information extraction (Roche & Schabes, 1997). This approach was demonstrated to be very effective for extracting information from text in the CIRCUS and the FASTUS system (Lehnert, Cardie, Fisher, Riloff, & Williams, 1991; Hobbs et al., 1997). In the widely used natural language processing toolkit, GATE, the semantic tagger JAPE (Java Annotations Pattern Engine) is able to describe patterns used to match and annotate tokens (Cunningham, Maynard, Bontcheva, & Tablan, 2002). JAPE adopts a version of CPSL (Common Pattern Specification Language), which provides FST over annotations (Appelt & Onyshkevych, 1998). Chang et al. also presented cascaded regular expressions over tokens, which proposed a cascaded pattern matching tool over token sequences (Chang & Manning, 2014).

2.3. Ontology in job recommender systems

Celik Duygua and Elci Atilla proposed an ontology-based method for processing résumés (ORP), which uses ontology to assist with the information extraction process (Çelik & Elçi, 2013). Their system processes a résumé by first converting the résumé file into plain text, sentence segmentation, applying an ontology knowledge base to find concepts in each sentence, term normalization, and finally classifying each sentence. Sánchez et al. summarized ontology-based similarity assessment into three types and provided a set of advantages and disadvantages for each approach (Sánchez, Batet, Isern, & Valls, 2012). The three types of categories include: Edge-Counting approach, Feature-Based measures, and measures based on information content.

In Edge-Counting approaches, ontology is viewed as a directed graph, in which the nodes are concepts, and the edges are taxonomic relation (e.g. is-a). Rada, et al. measure similarity as the distance between two nodes in the graph (Rada, Mili, Bicknell, & Blettner, 1989). Wu and Palmer noted that depth in the taxonomy also impacts the similarity measure between two nodes, because the deeper the nodes are within the tree, the smaller the semantic distance (Wu & Palmer, 1994). Based on a similar idea, Leacock and Chodorow also proposed a similarity measure combining the distance between terms and the depth of taxonomy (Leacock & Chodorow, 1998). There are some limitations to path-based approaches. Firstly, they only consider the shortest path between concept pairs. When faced with more complex structures, such as multiple taxonomic inheritance, the accuracy similarity measures is significantly reduced. Another limitation of the path-based approaches is assumption that all links in the taxonomy have uniform distance.

Feature-Based measures assess the similarity between concepts as a function of their properties. They consider the degree of overlap between sets of ontological features, in a manner similar to Tversky's model, which subtracts non-common features from common features of two concepts (Tversky, 1977). Rodríguez and Egenhofer computed similarity by calculating the weighted sum of similarities between synsets, features, and neighbor concepts (Rodríguez & Egenhofer, 2003). Feature-Based measures are dependent on semantic knowledge and therefore require information from a large database of ontologies and thesauri such as Wordnet (Miller, 1995).

Content-based measures try to overcome the limitations inherent in edge-counting methods. Resnik proposed a similarity measure, which depends on the amount of shared information between two terms (Resnik, 1995). Lin, as well as Jiang and Conrath both proposed extensions to Resnik's work (Jiang and Conrath, 1997;Lin, 1998. They also considered the Information Content (IC), which is the negative log of the probability of occurrence, of each of the evaluated terms, and they proposed that the similarity between two terms should be measured as the ratio between the amount of information needed to state their commonality and the information needed to fully describe them. There are however, disadvantages to using content-based measures. Firstly, content-based measures are unable to process leaf nodes, because they do not have subsumers. Secondly, similarity measures between concepts that share a small number of subsumers in common tend to be very inaccurate.

2.4. Computing relevance in job recommender systems

Lu et al. used latent semantic analysis (LSA) to calculate similarities between jobs and candidates, but they only selected two factors “interest” and “education” to compare candidates (Lu, El Helou, & Gillet, 2013). Yi et al. used structured relevance models (SRM) to match résumés and jobs (Yi, Allan, & Croft, 2007). Drigas et al. presented an expert system to match jobs and job seekers, and to recommend unemployed to the positions (Drigas, Kouremenos, Vrettos, Vrettaros, & Kouremenos, 2004). The expert system used Neuro-Fuzzy rules to evaluate the matching between user profiles and job openings. Daramola et al. also proposed a fuzzy logic based expert system (FES) tool for online personnel recruitment (Daramola, Oladipupo, & Musa, 2010). In the paper, the authors assumed that the information was already collected. The system uses a fuzzy distance metric to rank candidates’ profiles in the order of their eligibility for the job.

2.5. RésuMatcher highlights

Many recommender systems are designed based on the assumption that both résumés and job descriptions have uniformity in their structure. Based on these assumptions, they use methods such as template matching, for extracting information. The resulting information extraction modules differ between organizations and favor an employer centric approach, which does not account for structural differences between candidate résumés. This design forces job seekers to adapt their résumés to each employer, which greatly increases the complexity and difficulty of the job search process. By using an object based design approach, the RésuMatcher system offers the advantage of a uniform information extraction and matching module, which factors these structural differences between job descriptions and candidate résumés.

To incorporate the interrelationships between skills and experience within the context of job search, the RésuMatcher system generates a domain specific ontology of skills and uses machine learning to develop a novel statistical similarity measure to establish a formal relationship between skills. This results in a fine grained relationship between skills that are otherwise deemed unrelated in more traditional recommender systems, which use more coarse grained methods such as keyword matching and other similar methods.

3. Résumatcher system overview

3.1. Usage scenario

The RésuMatcher system receives automatic updates from daily job postings through a *jobs web crawler*. These job postings are processed and stored in a database. A potential user seeking a job will upload a résumé onto the system. Subsequently, the system parses the user’s résumé building a résumé model, which is used as a query object to the database to retrieve relevant jobs. A job is determined to be relevant based on a novel similarity metric, which compares features from a résumé object with features from job objects in the database. A list of jobs is returned to the user in sorted order of similarity to the user’s résumé (illustrated in Fig. 2).

3.2. System architecture overview

In this section, we provide an overview of the main components of the RésuMatcher system. As illustrated in Fig. 3, the RésuMatcher system comprises of the following main components: Job Data Processor, Search Interface, and the Résumé Matcher.

3.2.1. Jobs data processor

The Jobs Data Processor component executes a daily batch job, which processes all new job listings posted on the web. It consists of the following modules:

1. Web crawler: This component systematically browses job websites to help create an index of new job posts. The jobs web crawler downloads web pages and intelligently extracts data specific to job posting (this filters and excludes all non-job related content such as advertising). To accomplish this, the web crawler uses a customized template, which is unique to each website. Upon retrieval, jobs are identified using a simple template matching algorithm.
2. Jobs model builder: The jobs model builder parses and processes sentences contained in a job listing to extract job related information (such as location, required skills, required experience, etc.). Content extracted from the job listing is further processed using a feature extraction algorithm to create a job model described by four features: a job title, major, academic degree, and required skills. The output of the Model Object builder is a job object which consists of a job description (content displayable to end user) and a job model, which consists of features as described above.

3.2.2. Search interface

The résumé query interface provides a front-end interactive interface through which the prospective job seeker submits a résumé and subsequently reviews relevant search results. The Search Interface consists of the following modules:

1. Query input module: This component allows the user to upload a résumé document as the primary search parameter for a job search. This interface accepts résumés in Microsoft Word, PDF, and plain text format.
2. Résumé model builder: The Résumé Model Builder in the Search Interface processes content in a résumé document to extract candidate related information. This includes such data related to academic or professional training and education, and skills. This data is then used to build a résumé model object, which is used as the primary query item in the Résumé Matcher component.

3.2.3. Résumé matcher

This is the core component of the RésuMatcher system. The Résumé Matcher receives as input, a résumé object from the Query Interface, and queries the job database using a novel similarity method to retrieve the most relevant jobs. The similarity between a résumé object and a job object is calculated as a weighted sum of the computed features. This procedure is described in greater detail in Section 6. A list of jobs is returned and displayed to the user in sorted order of similarity to the résumé object via the Search Interface.

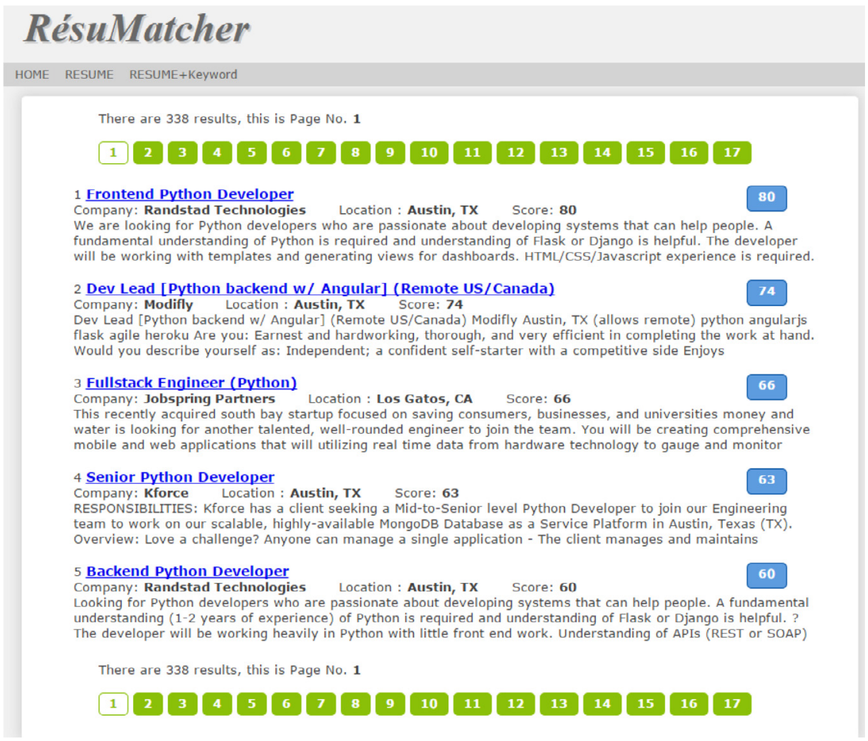
4. Information extraction

4.1. Information extraction overview

In this section, we describe procedures for extracting job related content from job posting websites. Content from Internet job postings and content from uploaded résumés undergo a similar procedure, which is comprised of three major steps: preprocessing, semantic labeling, and pattern matching. These steps are illustrated in Fig. 4.

4.2. Preprocessing

The preprocessing stage involves parsing, segmenting, and tokenizing. Content from job postings or uploaded résumés are parsed



There are 338 results, this is Page No. **1**

1 Frontend Python Developer
Company: **Randstad Technologies** Location : **Austin, TX** Score: **80**
We are looking for Python developers who are passionate about developing systems that can help people. A fundamental understanding of Python is required and understanding of Flask or Django is helpful. The developer will be working with templates and generating views for dashboards. HTML/CSS/Javascript experience is required.

2 Dev Lead [Python backend w/ Angular] (Remote US/Canada)
Company: **Modifly** Location : **Austin, TX** Score: **74**
Dev Lead [Python backend w/ Angular] (Remote US/Canada) Modifly Austin, TX (allows remote) python angularjs flask agile heroku Are you: Earnest and hardworking, thorough, and very efficient in completing the work at hand. Would you describe yourself as: Independent; a confident self-starter with a competitive side Enjoys

3 Fullstack Engineer (Python)
Company: **Jobspring Partners** Location : **Los Gatos, CA** Score: **66**
This recently acquired south bay startup focused on saving consumers, businesses, and universities money and water is looking for another talented, well-rounded engineer to join the team. You will be creating comprehensive mobile and web applications that will utilizing real time data from hardware technology to gauge and monitor

4 Senior Python Developer
Company: **Kforce** Location : **Austin, TX** Score: **63**
RESPONSIBILITIES: Kforce has a client seeking a Mid-to-Senior level Python Developer to join our Engineering team to work on our scalable, highly-available MongoDB Database as a Service Platform in Austin, Texas (TX). Overview: Love a challenge? Anyone can manage a single application - The client manages and maintains

5 Backend Python Developer
Company: **Randstad Technologies** Location : **Austin, TX** Score: **60**
Looking for Python developers who are passionate about developing systems that can help people. A fundamental understanding (1-2 years of experience) of Python is required and understanding of Flask or Django is helpful. ? The developer will be working heavily in Python with little front end work. Understanding of APIs (REST or SOAP)

There are 338 results, this is Page No. **1**

Fig. 2. Screenshot of RésuMatcher query user interface.

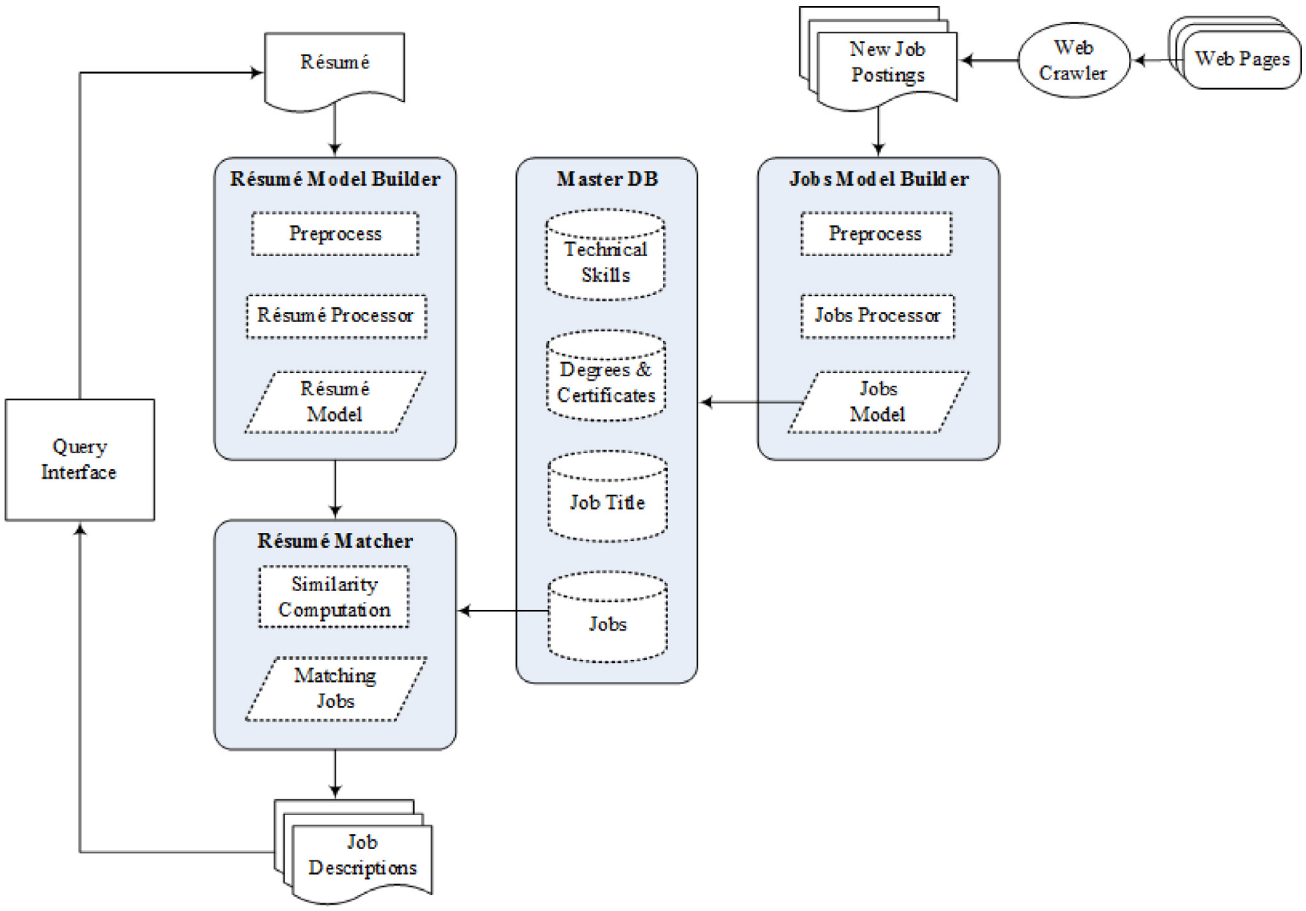


Fig. 3. RésuMatcher system architecture.

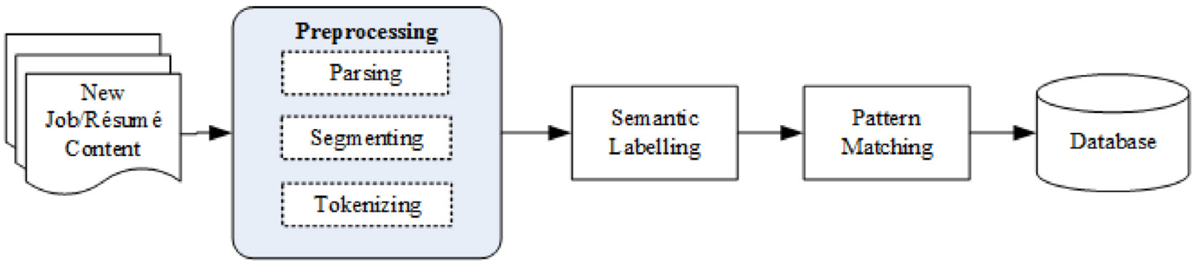


Fig. 4. Information processing pipeline.

Table 1
Degree tokens and variants.

Word	Variants					
Baccalaureate	bachelors	bachelor	B.S.	B.A.	BA/BS	Undergraduate
Masters	M.S.	M.E.	Meng	MBA	MSc	MS
Doctorate	PhD	Ph.D.	Doctor	Ph.D		

Table 2
Semantic labels.

Input token	Hyponym	Hypernym
Baccalaureate	BS-LEVEL	
Bachelor's		
BS		
Master	MS-LEVEL	DE-LEVEL
MS		
Doctorate	PHD-LEVEL	
PhD		

to extract related attributes in the form of paragraphs and sentences. In the segmenting stage, attributes such as job title, job location, company, and job description are extracted from job postings, while professional skills, education, and experience are extracted from uploaded résumés. In the next stage, characters such as '/' and '-', which do not convey any meaningful information are replaced with white spaces prior to tokenizing. Sentences are then tokenized into arrays of tokens using NLTK (Bird, 2006).

4.3. Semantic labeling

Natural Language Processing (NLP) poses a unique challenge in that the relationship between word and meaning is very often a many-to-many relationship, that is a single word can be used to express several concepts and vice versa. For example, the expression *Bachelor's degree* can be expressed several ways in a job description including *B.S., BS, 4-year degree*, and so on. This challenge is further illustrated in Table 1, which provides a list of words that can be disambiguated to mean a semantic value of "Bachelor's degree".

Using the tokens generated from job or résumé documents will result in a significant amount of overlap. Further, encoding this information in a finite state transducer (FST) will result in higher space complexity. To overcome this problem, we implement a pattern matching library which supports regular expressions over tokens. This method of semantic labeling maps multiple tokens to a unique label or hyponym. Each hyponym is subsequently mapped to a hypernym. Applying this process to the inputs in Table 1, results in more simple semantic labeling map illustrated in Table 2.

This method of structured hierarchical mapping is achieved by implementing a dictionary mapping tokens to hyponyms, and hyponyms to hypernyms. Table 3 shows the application of this method to a sample sentence from a job description.

The output pattern "DE-LEVEL DEGREE IN MAJOR OR MAJOR" provides a condensed representation of the input token string, thereby reducing the spatial complexity in encoding a large number unique patterns represented in our FST, and a faster running time in the pattern matching process.

4.4. Pattern matching library

Finite-state machines have been used in various domains of natural language processing especially in computational linguistics. Linguistically, in using finite automata we can easily describe the relevant information encountered in a language. This results in a compact representation of lexical rules, or idioms and cliches (Gross, 1989). From a computational point of view however, finite-state machines serve to improve time and space efficiency.

A finite state transducer (FST) is a finite state machine which, given an input string, is able to generate a unique output string. For the RésuMatcher system, we developed a flexible, lightweight FST framework, capable of processing regular expression matching over labeled tokens. Our FST library supports regular expression, operator and object-oriented expressions as input. Providing support for operator and object-oriented expression allows for greater flexibility in customizing the library for other domains. The library supports seven types of Matchers, they are: UnitMatcher, the basic token matching unit; SequenceMatcher, which is a list of Matcher sequence of characters; QuestionMatcher, which matches one or more of the preceding tokens; StarMatcher, which matches zero or more of the preceding tokens; PlusMatcher, which matches zero or one of the preceding tokens; DotMatcher, which matches any token, and RegexMatcher, that any token matches the regular expression in it.

The framework supports three styles of creating patterns: regular expression style, operator style and object-oriented style. The second and third styles are flexible because developers can create their own matcher class to extend the feature of the library. We use examples to show how the three styles work. The most common style is defining pattern expression in a string, which is much like traditional regular expression.

5. Skills similarity measures for résumés and jobs

In this section, we describe a methodology for extracting skills from both job postings and résumé submissions to build a domain specific skills library. Using this library, we develop an ontology of skills as a framework for relationship between skills. From the con-

Table 3
Sentence labels.

Semantic layer	Expression					
Token	*bachelors*	*degree*	*in*	*computer science*	*or*	*information systems*
Hyponym	BS-LEVEL	DEGREE	IN	MAJOR-CS	OR	MAJOR-INFO
Hypernym	DE-LEVEL	DEGREE	IN	MAJOR	OR	MAJOR

Table 4
Skill requirements as listed in job description.

1	A high-level language such as Java, Ruby or Python; we use Java and Groovy extensively
2	HTML5/CSS3/JavaScript, web standards, jQuery or frameworks like AngularJS would be great
3	HTML CSS and Javascript a must
4	Experience with AJAX, XML, XSL, XSLT, CSS, JavaScript, JQuery, HTML and Web Services

straints provided by this framework, we describe how to quantify the similarity between skills.

5.1. Domain specific skills library

One major challenge in the job search process is the ability to match required job skills with a candidate possessing such skills. The first step in the process is to understand the skills required for the job. Using sentences extracted from actual job descriptions in Table 4, we observed the following characteristics:

1. Job postings provide information on required skills explicitly.
2. Required skills in job postings are enumerated in one or more sentences.
3. A majority of job postings require more than a single skill.

To take advantage of these constraints, we have developed a template matching machine learning algorithm to automatically learn what job skills are and extract them from both job postings and résumés. First we construct a skills dictionary using a few known skills. This skills dictionary is updated iteratively using a training set of job descriptions. For each job description, we extract one or more skills sentences. A skill sentence is any sentence within a job description identified as having at least two known skills (known skills are those present in the skills database). The skill-sentence is tokenized and processed to remove stop words. This process results in a collection of tokens, some of which are known skills. Each unknown token is validated to be an actual skill by cross-referencing with DBpedia. DBpedia is a crowd-sourced community effort to extract structured information from *Wikipedia* and make this information available on the Web. DBpedia allows you to ask sophisticated queries against *Wikipedia*. After successful validation, each new skill is added to the skills dictionary thereby expanding the size and effectiveness of discovering new skills with each new training iteration.

5.2. Skills similarity measures for résumés and jobs

Current systems base search results on keyword matching. While this method provides plausible results, it does not use linguistic properties specific to the domain. When we consider text used in job descriptions and résumés to describe job requirements and personal qualifications respectively, we find hierarchical relationships between skills and qualifications. Secondly, a significant amount of ambiguity exists between domain specific words and their respective interpretation. This ambiguity is illustrated in Table 5, which shows sentences extracted from résumés and job descriptions.

We observed that simple keyword matching is not a good similarity measure, because job descriptions and résumés both contain richer and more complex words that cannot be described simply by keywords. In these documents, some concepts can be written in different ways, and other concepts can have close relationships. For example, Table 5 shows portions of a résumé and a job description.

Table 5
An example of job description and résumé.

Job description
3+ years development experience in Java and OOA/OOD
2+ years in Java, JSP, J2EE, SQL, VXML, Web-services environment
2+ years XML, XSL, XML Beans
Strong MS SQL Server, MySql
Familiarity with Apache, Tomcat, JBoss, Struts and Web Services
Familiarity with test driven development
Experience working in Agile methodology
Résumé
Technical skills:
Languages Java, C++, HTML, HTML5, XML, PL/SQL
J2EE Technologies Java J2EE, Servlets, JSP, Struts, Java web application servers
Version Control CVS, Bugzilla, SVN, PVCS: Ant, Maven
Experience in Oracle 11g, PostgreSQL
NoSQL: MongoDB, HBase, Cassandra

From the résumé illustrated in the Table 5 we observe that a keyword search will incorrectly conclude that this résumé is not a suitable match for the job description illustrated above it (see Table 5). However, we observe intuitively that relationships exists between some of the skills listed in the candidate résumé and the requirements from the job description. For example, *experience in Oracle database* demonstrates a level of competence for a job that requires *Mysql*, and *MS-SQL* skills. Similarly, knowledge and experience with object oriented programming and design viz *OOA/OOD*, is evidenced with *Java* and *C++* programming experience. In the same manner, *Tomcat* and *JBOSS* are both considered *Java web application servers*. To overcome these deficiencies, we propose a novel ontology based similarity measure, details of which are presented in the following sections.

5.3. Constructing an ontology of skills

To incorporate the observed interrelationships that exist between skills and experience within the context of job search, we have developed a domain specific ontology of skills. An *ontology* provides a formal definition of objects, properties, and interrelationships between objects for a the given domain. Thus creating a taxonomy, which compartmentalizes variables needed for computing similarities between skills, thereby establishing formal relationships between them. Taking advantage of the taxonomy provided through DBpedia, we combine this structure with a novel similarity measure to establish a formal relationship between skills Fig. 5.

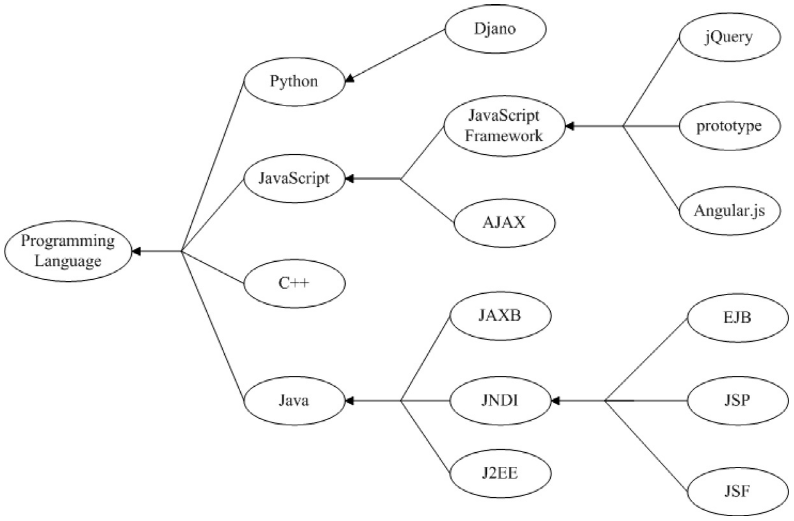


Fig. 5. Programming language ontology.

5.4. Computing similarity measure

The Skills ontology framework provides a real-world constraint on relationships between skills. Within this framework, a skill a is said to be similar to another skill b if at least one of the following conditions are met:

1. Skill A is a hypernym of skill B.
2. Skill A is a hyponym of skill B.
3. Skill A and Skill B share the same hypernym.

Given two skills, a and b, both having the same direct hypernym or one is the hypernym of the other, we define similarity $sim(a, b)$ between a and b as

$$sim(a,b) = \frac{jaccard(A,B)}{\log_2\{\frac{1}{N_{a\cap b}}\sum_{k=1}^{N_{a\cap b}} \arg\min_{i,j} \|d^k_{a_i} - d^k_{b_j}\|\}} \tag{1}$$

Where $jaccard(A, B)$ is defined as: $jaccard(A,B) = \frac{N_{a\cap b}}{N_{a\cup b}}$.

$N_{a\cap b}$ is the number of documents $d_{a\cap b}$ where skills a and b are both present.

$N_{a\cup b}$ is the number of documents $d_{a\cup b}$ where either skill a or skill b is present.

$d^k_{a_i}$ is the term index for the ith occurrence of skill a in the kth $d_{a\cap b}$ document.

$d^k_{b_j}$ is the term index for the jth occurrence of skill b in the kth $d_{a\cap b}$ document.

6. Document model similarity

The document model describes the model of a job description document (*job objects*) or a résumé document (*résumé object*). Both objects are constituent of similar fields: job title, college major, academic degree, and technical skills. Once we obtain an object representation of both job description (*job objects*) and résumé documents (*résumé objects*), we are able to compare both objects by computing a weighted similarity measure. Formally, we define the similarity between a *job objects* d_j and a *résumé objects* d_r as the weighted sum of the distance between corresponding fields in both documents

$$sim(d^i_j, d^i_r) = \sum_{i=1}^{n} \lambda_i(d^i_j, d^i_r) \times \omega_i \tag{2}$$

Where $\lambda_i(d_{j_i}, d_{r_i})$ is a distance function for the ith field between both objects, and ω_i is an empirically determined weight term for the ith field. In the following sections, we provide a detailed description of λ for each field.

6.1. College major similarity

In the simplest case, if the majors in the résumé model and job model are the same, the similarity value is 1. If they are different, we can check whether the major in the résumé model is in the list of related majors for the major listed in the job model. If this is the case, the similarity value is 0.5; otherwise the similarity value is 0. The computation for major similarity λ_m is shown below

$$\lambda_m(m_j, m_r) = \left\{\begin{array}{ll} 1, & m_j = m_r \\ 0.5, & m_j \in hypernym(m_r) \| m_j \in hyponym(m_r) \| \\ & \quad hypernym(m_j) = hypernym(m_r) \\ 0, & otherwise \end{array}\right\} \tag{3}$$

Where m_j and m_r represent the college major listed in the job description and the résumé respectively.

6.2. Academic degree similarity

The design of the academic degree similarity metric is to: (1) exclude all jobs for which the candidate is under-qualified, i.e. zero weight to jobs for which the academic qualification requirement is greater than the candidate's highest academic qualification (2) give higher weight to jobs for which the highest academic qualification requirement is at most two degrees below the candidate's highest academic qualification, (3) assign a lower weight to jobs for which the highest academic qualification requirement is greater than two degrees below the candidate's highest academic qualification. To achieve this, we assign a unique integer value between 1 and 5 to the five major types of academic degrees (High School Diploma, Associates, Bachelor's, Master's, and Doctoral degrees). This method assigns a higher numerical value to more advanced degrees (e.g. A doctoral degree is assigned a value of 5, while a High School diploma is assigned a value of 1). Based on the numerical values assigned to each degree, we calculate the academic degree similarity λ_d using the following equation:

$$\lambda_d(d_j, d_r) = \left\{\begin{array}{ll} 0, if & d_r < d_j \\ 1, if & d_r - d_j \leqslant 2 \\ 0.5, if & d_r - d_j > 2 \end{array}\right\} \tag{4}$$

Where d_j and d_r represent the academic degree listed in the job description and the résumé respectively.

Table 6
Pairwise similarity measures by 12 experts.

Skill	Javascript	C#	PHP	CSS	Java	Python	Ruby	Bash	SQL	C++
Javascript	1									
C#	.6	1								
PHP	.63	.58	1							
CSS	.58	.3	.48	1						
Java	.62	.8	.58	.3	1					
Python	.64	.58	.56	.36	.70	1				
Ruby	.7	.48	.63	.49	.58	.64	1			
Bash	.51	.42	.54	.35	.44	.49	.5	1		
SQL	.38	.3	.4	.42	.33	.4	.4	.3	1	
C++	.5	.82	.53	.29	.84	.64	.56	.42	.31	1

6.3. Job title similarity

The title of a job description contains information that can be separated into unique segments including functional role (developer, tester, architect), hierarchical level (junior, senior), platform (web, mobile, network, cloud), programming language, and domain. There are typically multiple titles in a single résumé, reflecting a job candidate's prior experience. This results in a one to many comparison between job object title and résumé object titles. We determine the similarity between the job title and the résumé titles by calculating the distance between all résumé titles and selecting the title with the highest numerical value. We calculate the similarity between job titles using the following equation:

$$\lambda_t(t_j, t_r) = \arg\max_k \left\{ \frac{1}{n} \sum_{i=1}^{n} sim_k(t_{j_i}, t_{r_i}) \right\} \tag{5}$$

Where $sim_k(t_{j_i}, t_{r_i}) = \begin{Bmatrix} 1, & t_{j_i} = t_{r_i} \\ 0, & otherwise \end{Bmatrix}$

k is the number of unique job titles contained in a résumé object, n is the number of unique segments within a résumé job title, t_{r_i} and t_{j_i} represent the ith term in the title of the résumé object and job object respectively.

6.4. Technical skills similarity

As described previously, we developed an ontology of skills with which we determine the similarity between skills. Using on this framework, we compute the similarity λ_s between skills listed in job object s_j and skills listed in résumé object s_r as follows:

$$\lambda_s(s_j, s_r) = \frac{1}{n} \sum_{i=1}^{n} \arg\max_m \{sim(s_{j_n}, s_{r_m})\} \tag{6}$$

Where the similarity $sim(s_{j_n}, s_{r_m})$ between skills s_{j_n} and s_{r_m} is computed as defined in Eq. 1, n is the number of skills listed in the job description, and m is the number of skills listed in the résumé.

7. Experimental evaluation

7.1. Overview

In this section, we describe methodology for evaluating each system component. We also evaluate the overall performance of the RésuMatcher system by comparing the quality of results obtained during a job search with the results of a similar search using a search engine from a popular job search website.

First, we to populated the RésuMatcher system database. As illustrated in Fig. 3, and later described in Section 3.2.1, this task is accomplished by the *jobs web crawler*. The jobs web crawler was configured to retrieve a thousand (1000) computing related job postings from www.indeed.com. For evaluative purposes, a computing related job is defined as a job posting that requires one or more software programming skills. A list of software programming skills utilized by the jobs web crawler to retrieve jobs for our experimental evaluation is provided in Table 7.

After populating the jobs database, we performed component-level and system-level tests, and report on the computational efficiency and storage implications of the system. Finally, we evaluated the performance of the proposed information extraction methods and similarity indexes with previously existing methods.

Table 7
Computing related skills for jobs web crawler retrieval from www.indeed.com

No.	Programming skill
1	Java
2	Object-C
3	C++
4	Visual Basic
5	C#
6	PHP
7	Python
8	Javascript
9	Perl
10	Visual Basic.NET
11	Ruby
12	F#
13	Transact-SQL
14	Delphi
15	Object Pascal
16	LISP
17	Pascal
18	MATLAB
19	Actionscript
20	SQL
21	PL/SQL
22	postscript
23	Cobol
24	SAS
25	Fortran
26	Lua
27	Scala
28	Prolog
29	Erlang
30	Groovy
31	Foxpro
32	JCL

7.2. Performance benchmark tests

We evaluated the performance of the RésuMatcher system by testing the computational costs of the four major components of the system: (1) jobs web crawler , (2) jobs model builder (3) résumé matcher, and (4) résumé processor. In addition, we performed an end-to-end test of the RésuMatcher system (see Fig. 3).

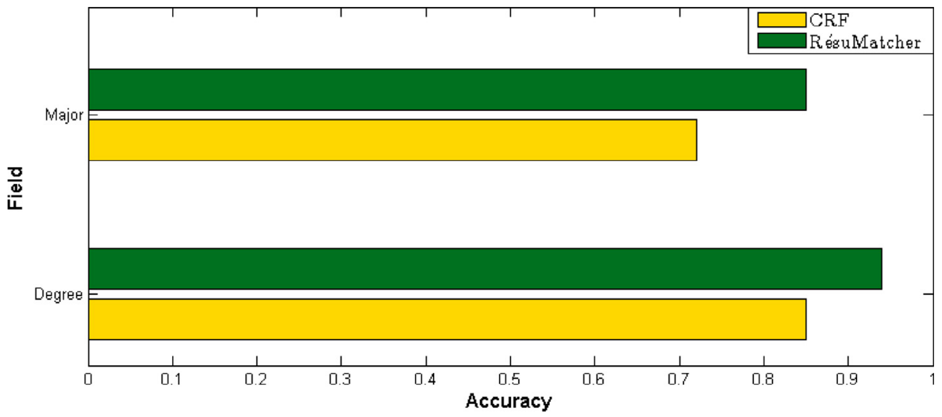


Fig. 6. Information extraction performance comparison between RésuMatcher and conditional random fields.

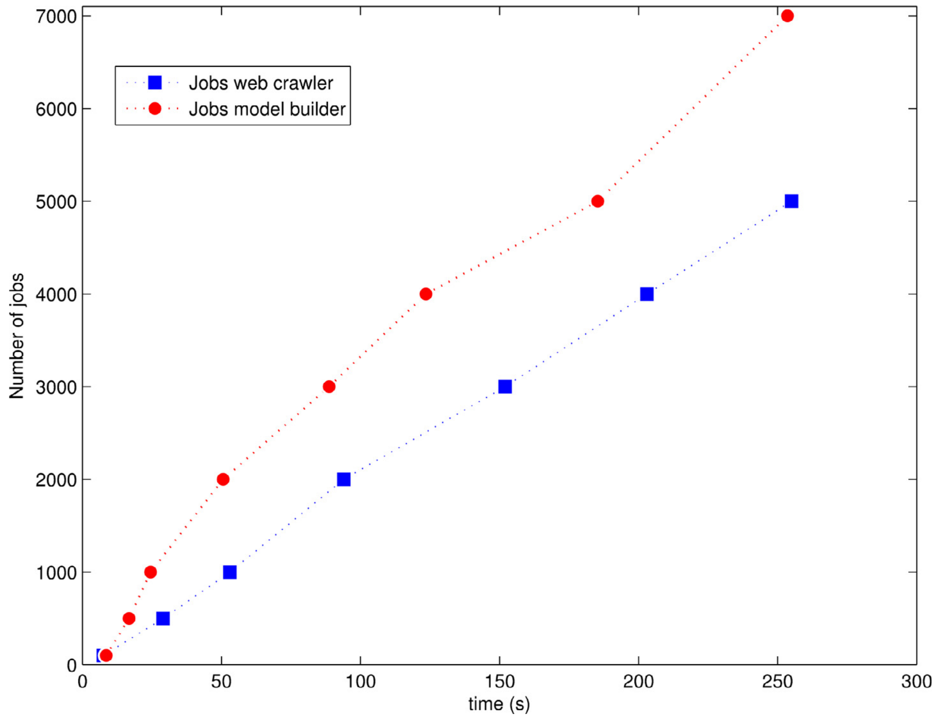


Fig. 7. Jobs data processor components performance test.

All tests were performed on a desktop PC running Microsoft Windows 10 operating system (additional specifications: intel i5-3210 quad-core CPU at 2.5 GHz, 8 GB RAM, and 256 GB SSD).

7.2.1. Jobs web crawler

The jobs web crawler was designed to run as a background process during periods of low network traffic (approximately 2:00am), returning an average of 7389 job postings. We conducted a scalability test to determine how the jobs web crawler's performance changes with an increase in the number of jobs returned. The results from these tests, illustrated in Fig. 7, show that the jobs web crawler performance increases linearly with the number of jobs retrieved.

7.2.2. Jobs model builder

The jobs model builder processes job postings to extract information about the location, skills, academic qualification, experience, etc. required for the job in question (see Section 3.2.1). We conducted a similar scalability test on the jobs model processor to determine its performance changes with increments in the number of jobs queued for processing. Fig. 7 illustrates the performance of the job model builder as a function of the number of jobs being processed. The jobs model builder computational time, similar to the jobs web crawler, approximately linearly in the number jobs.

7.2.3. Resume model builder

The résumé model builder processes résumés received via the query interface. This component extracts candidate related information (such as academic or professional training, skill, work experience, etc.) from each résumé. The résumé model builder outputs a résumé model object, which is used for similarity computations in the later stages of the RésuMatcher system. To test the scalability of the résumé model builder, we ran an automated résumés query script, which repeatedly uploads a specified number of résumés at random and records the computation time. We tested the résumé model builder component with 10, 20, 50, and 100, repeated uploads and reported an average processing time of 118 ms per resume.

Table 8
Résumatcher overall performance results.

Highest degree	Word count	No. of jobs	Experience (yrs)	No. skills	Time (ms)
ASSOCIATE	687	2	15	20	284
BS-LEVEL	875	6	4	12	270
BS-LEVEL	700	7	9	7	284
BS-LEVEL	444	4	11	14	241
MS-LEVEL	544	4	10	9	170
MS-LEVEL	547	3	9	15	306
PHD-LEVEL	1050	1	2	5	107
PHD-LEVEL	643	1	1	7	156

7.2.4. Resume matcher

The résumé matcher is the core component of the RésuMatcher system. It takes as input the résumé model, interacts with the jobs database using similarity indexes discussed in Section 3.2.3, and retrieves the most relevant jobs for the resume in question. The performance of the résumé matcher component is largely dependent on the size of the jobs database, since it performs a similarity computation on job models contained in the database. To this end, we recorded the average computation time of the résumé matcher component on multiple randomly ordered queries from eight distinct résumés while increasing the size of jobs database (100, 500, 1000). The computation time over multiple trials averaged (250 ms) on each résumé.

7.2.5. Overall system

We evaluated the performance of the overall system by running end to end tests, which starts with a user uploading a resume and ends when query results are presented to the user. The overall system test was conducted using eight (8) résumés with varied academic qualifications, skills, and years of experience. We recorded the time duration for job queries on each résumé while increasing the size of the jobs database (100, 500, 1000). In Table 8, we report the averaged performance (duration of query) for the RésuMatcher system on each of the eight résumés. Finally, we report the storage costs of the RésuMatcher system (illustrated in Fig. 8) as a function of the size of the database.

7.3. Information extraction

To evaluate the performance of the information extraction module, we measure the accuracy of sentence filters in identifying sentences whose content pertain to the applicant's college degree. For this experiment, we selected 100 sentences from an existing collection of job descriptions. The content of each sentence defines candidate degree and college major requirements for the respective job posting. Values for "degree" and "major" were manually labeled. Patterns were identified from each sentence to match and extract degree information. We compare our method with the use of the Conditional Random Fields (CRF) model, a state of art in machine learning model for sequence labeling (Lafferty, McCallum, & Pereira, 2001)]. Fig. 6 shows results obtained using patterns from our pattern matching library discussed in Section 4.4 (candidate degree and candidate major use 6 and 10 unique patterns respectively). The RésuMatcher content matching method outperforms (94% accuracy) the conditional random fields (CRF) model (85% accuracy) when predicting type of "degree".

7.4. Ontology-based similarity

We tested the accuracy of our ontology-based similarity measures by comparing the ranking of ontology similarity scores on 8 skills from a collection of 500 job descriptions with the average pairwise similarity score between skills by domain experts ($n = 12$). Table 6 and Table 9 give pairwise similarity measures for 10 skills from a collection of 500 job descriptions using average expert scores and ontology-based similarity measures respectively. A higher value corresponds to a greater degree of similarity between two skills. We use the *Normalized Discounted Cumulative Gain* (NDCG) as a measure of ranking quality (Manning, Raghavan, & Schütze, 2008). NDCG is used in the field of information retrieval to measure effectiveness of web search engine algorithms or related applications. DCG measures the usefulness of a document by comparing its position in the result list against the graded relevance scale of documents returned in a result set. Using measures from Table 6 as ground truth estimates, we computed NDCG for the similarity measures in Table 9. Our results, provided in Table 10, show that the RésuMatcher system consistently returns the most relevant job search results $(\mu = 0.96, \sigma = 0.04)$.

7.5. Comparison with alternative retrieval models

We created a data set of 100 job descriptions, for a diverse set of jobs such as web developer, server back-end developer, mobile developer, etc. Using 5 candidate résumés, we queried the corpus of 100 jobs using the RésuMatcher system to retrieve the top 20 jobs. We used the average ranking provided by domain experts as ground-truth measure of relevance of each job description to a given résumé.

To evaluate the quality of the similarity ranking of search results returned by the RésuMatcher system in comparison with other retrieval methods, we use two measures: *NDCG* (described previously) and *Precision@k*. *Precision@k* measures the proportion of relevant documents in the first K positions but disregards the individual position of each document. Using these two measures, we compare the performance of the RésuMatcher system with three established retrieval models: Kullback–Leibler divergence algorithm (*KL*), Term Frequency - Inverse Document Frequency ($tf - idf$) and Okapi BM25 (*BM25*) (Zhai, 2008; Manning et al., 2008; Robertson et al., 1995). Kullback–Leibler divergence provides a non-symmetric measure of the difference between two documents. The term frequency-inverse document frequency ($tf - idf$) is a numerical statistic, which measures the importance of a word to a document within a collection and can be used as a model for comparing the similarity between documents. Okapi BM25 is a bag-of-words retrieval model, which ranks documents based on similar query terms appearing in each document.

In Fig. 9 we illustrate the quality of job ranking results obtained using the RésuMatcher system in comparison with alternative models using *Precision@k* and *NDCG*. Both the RésuMatcher system and $tf - idf$ yield better results in comparison with other methods. However, the RésuMatcher system consistently outperforms ($\mu = 37.44\%$) existing information retrieval methods in the quality of jobs matches returned.

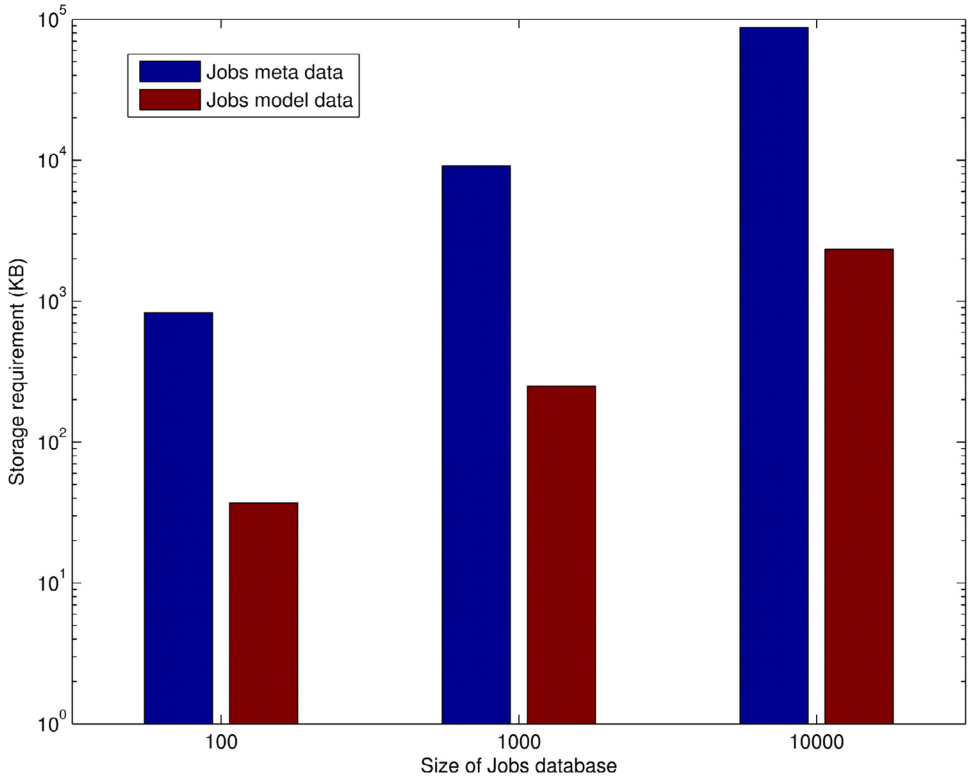


Fig. 8. Log scale illustration of Résumatcher system storage costs.

Table 9
RésuMatcher Pairwise ontology-based similarity scores.

Skill	Javascript	C#	PHP	CSS	Java	Python	Ruby	Bash	SQL	C++
Javascript	1									
C#	.04	1								
PHP	.06	.01	1							
CSS	.24	.04	.06	1						
Java	.07	.02	.04	.05	1					
Python	.02	.01	.06	.02	.05	1				
Ruby	.02	.02	.06	.02	.03	.13	1			
Bash	0	0	0	0	0	0	.02	1		
SQL	.08	.11	.04	.07	.05	.02	.03	0	1	
C++	0	.03	0	0	.01	.02	.04	0	.01	1

Table 10
Pairwise skill Similarity.

Skill	JavaScript	C#	PHP	CSS	Java	Python	Ruby	Bash	SQL	C++
NDCG	0.960	0.905	0.988	0.993	0.898	0.985	0.981	0.990	0.987	0.951

7.6. Comparison with keyword search

Finally, we compared the quality of search results obtained using the RésuMatcher system and a keyword-based search, which is a commonly used search method on a majority of job search websites. In this experiment, we selected a listed on a given résumé (e.g. "Java") as query term on www.indeed.com. After recording the first 20 jobs (used later for comparison), the first 500 results returned by www.indeed.com were stored in a jobs database. Using the same résumé as query term, we used the RésuMatcher system to retrieve the top 20 jobs from the jobs database created in the previous step. Using *Precision@k* and *DCG*, we compared the top 20 keyword search results with query results from the RésuMatcher system. Table 11 shows the average results of five résumés and five corresponding skills (Java, Python, Javascript, Ruby, and HTML). These results show that the ReśuMatcher system returns, on average, more accurate and more consistent results than keyword search.

Table 11
Comparison with www.indeed.com keyword search.

	Precision@k		DCG	
k	Indeed	RésuMatcher	Indeed	RésuMatcher
5	0.84	0.87	23.87	32.97
10	0.72	0.86	37.02	45.57
20	0.645	0.768	58.70	66.70

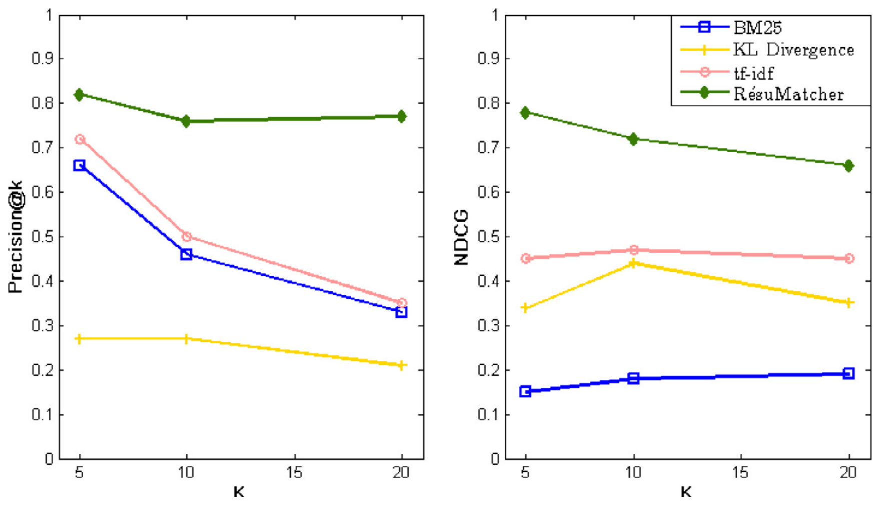


Fig. 9. Comparison with various job retrieval and ranking methods.

8. Conclusion and future work

In this paper we presented RésuMatcher, a personalized job-résumé matching system that offers a novel change in the online job search process. Using the content of a job seeker's résumé in contradistinction to traditional keyword search, we are able to offer more accurate and relevant search results. The main contribution of the RésuMatcher system is a novel statistical similarity measure for calculating and ranking the similarity between résumés and job postings.

We compared the performance of the RésuMatcher system's résumé-job matching algorithm with other information retrieval models (KL, $tf-idf$ and $OKpaiBM25$) using two measures: *Precision@k* and *NDCG*. We also performed a second set of evaluations comparing the RésuMatcher system with a commercial job search engine (www.indeed.com) on 10 keyword based search result rankings. Both results indicate that the RésuMatcher system produced more accurate and consistent results than other information retrieval methods, and provided an improvement over existing search methods used by www.indeed.com and other similar job search websites.

In addition, we examined scalability and reported on computational performance of the overall system, and its constituent components, as a function of query load and the size of the job search space (number of jobs available in the jobs database). Our results show that the RésuMatcher system scales linearly, and can be optimized for large scale implementation.

The results from performance tests presented in this paper are orchestrated on a desktop PC with a small dataset (10,000 jobs) and may therefore differ slightly from actual performance in a deployed server environment. From these results, we anticipate bottlenecks in the computational complexity of the résumé matching component, since it scales linearly with the size of the jobs database. However, distributed computing solutions such as Apache Hadoop, and database interaction solutions such as SPARQL, are designed specifically for large scale data manipulation as is required by a large scale implementation of the RésuMatcher system.

The current implementation of the RésuMatcher system is designed to handle a single domain as a proof of concept. A future implementation, expanding by including additional domains, will require the creation of domain-specific ontology mapping to accommodate jobs descriptions and résumés from other domains. This creates an additional layer of computational complexity in disambiguating between skills, job descriptors, and other terminology, which have domain-dependent meaning.

An intelligent job search tool opens the possibility for leveraging machine intelligence to make additional useful inference from jobs postings and job seeker résumés. Our current implementation focuses on finding appropriate jobs based on academic qualifications, skills, and work experience. However, human beings have additional constraints when seeking jobs, and these constraints are often subjective, i.e. highly dependent on job seeker in question. Factors such as location, location flexibility (openness to relocation), job seeker's age, types of companies the job seeker has worked with in the past (small, medium, or large size), industry, etc. Using information from both job models and resume models, we can train predictive models capable of making reliable inference on career progression and suggest potential jobs based on other more experienced resume models available within the system.

The job search process is a very complex and time consuming task, which is dependent on both explicit and implicit factors. Our work establishes the validity of using information extraction techniques in machine intelligence to create a more personalized job matching system, with ample potential for improvements in the future.

Acknowledgments

The authors thank members of the Sketch Recognition Lab for their support and comments on this work.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.eswa.2016.04.013

References

Appelt, D. E., & Onyshkevych, B. (1998). The common pattern specification language. In *Proceedings of a workshop on held at Baltimore, Maryland: October 13-15, 1998* (pp. 23–30). Association for Computational Linguistics.
Bird, S. (2006). Nltk: The natural language toolkit. In *Proceedings of the COLING/ACL on interactive presentation sessions* (pp. 69–72). Association for Computational Linguistics.
Brusilovsky, P., Kobsa, A., & Nejdl, W. (2007). *The adaptive web: Methods and strategies of web personalization*: vol. 4321. Springer-Verlag Berlin Heidelberg.
Burke, R. (1999). The wasabi personal shopper: A case-based recommender system. In *AAAI/IAAI* (pp. 844–849).

Çelik, D., & Elçi, A. (2013). Pervasive Computing and the Networked World: Joint International Conference, ICPCA/SWS 2012, Istanbul, Turkey, November 28–30, 2012, revised selected papers. *An Ontology-Based Information Extraction Approach for Résumés* (pp. 165–179). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-37015-1_14.

Chang, A. X., & Manning, C. D. (2014). TOKENSREGEX: Defining cascaded regular expressions over tokens. *Technical Report CSTR 2014-02*. Department of Computer Science, Stanford University.

Cunningham, H., Maynard, D., Bontcheva, K., & Tablan, V. (2002). A framework and graphical development environment for robust nlp tools and applications. In *ACL* (pp. 168–175).

Daramola, J. O., Oladipupo, O. O., & Musa, A. (2010). A fuzzy expert system (fes) tool for online personnel recruitments. *International Journal of Business Information Systems, 6*(4), 444–462.

Drigas, A., Kouremenos, S., Vrettos, S., Vrettaros, J., & Kouremenos, D. (2004). An expert system for job matching of the unemployed. *Expert Systems with Applications, 26*(2), 217–224.

Färber, F., Weitzel, T., & Keim, T. (2003). An automated recommendation approach to selection in personnel recruitment. In *AMCIS 2003 proceedings* (p. 302).

Gonzalez, T., Santos, P., Orozco, F., Alcaraz, M., Zaldivar, V., Obeso, A. D., et al. (2012). Adaptive employee profile classification for resource planning tool. In *SRII global conference (SRII), 2012 annual* (pp. 544–553). IEEE.

Gross, M. (1989). Electronic Dictionaries and Automata in Computational Linguistics: LITP Spring School on Theoretical Computer Science Saint-Pierre d'Oléron, France, May 25–29, 1987 Proceedings. *The Use of Finite Automata in the Lexical Representation of Natural Language* (pp. 34–50). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/3-540-51465-1_3.

Gueutal, D., Stone, L., & Salas, E. (2005). *The brave new world of e-HR: Human Resources in the digital age*. Pfeiffer, San Francisco.

Hobbs, J. R., Appelt, D., Bear, J., Israel, D., Kameyama, M., Stickel, M., et al. (1997). FASTUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text. *Finite-state language processing* p. 383.

Jiang, J. J., & Conrath, D. W. (1997). Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of international conference research on computational linguistics (ROCLING X)*.

Lafferty, J. D., McCallum, A., & Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the eighteenth international conference on machine learning*. In *ICML '01* (pp. 282–289). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.URL: http://dl.acm.org/citation.cfm?id=645530.655813.

Leacock, C., & Chodorow, M. (1998). Combining local context and wordnet similarity for word sense identification. *WordNet: An Electronic Lexical Database, 49*(2), 265–283.

Lee, D. H., & Brusilovsky, P. (2007). Fighting information overflow with personalized comprehensive information access: a proactive job recommender. In *Autonomic and autonomous systems, 2007. ICAS07. Third international conference on*. IEEE.21–21.

Lehnert, W., Cardie, C., Fisher, D., Riloff, E., & Williams, R. (1991). University of Massachusetts: Description of the circus system as used for muc-3. In *Proceedings of the 3rd conference on message understanding* (pp. 223–233). Association for Computational Linguistics.

Lin, D. (1998). An information-theoretic definition of similarity. In *ICML: 98* (pp. 296–304).

Lu, Y., El Helou, S., & Gillet, D. (2013). A recommender system for job seeking and recruiting website. In *Proceedings of the 22nd international conference on world wide web companion* (pp. 963–966). International World Wide Web Conferences Steering Committee.

Malinowski, J., Keim, T., Wendt, O., & Weitzel, T. (2006). Matching people and jobs: A bilateral recommendation approach. In *System sciences, 2006. HICSSŠ06. proceedings of the 39th annual Hawaii international conference on: vol. 6* (p. 137c). IEEE.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*: (vol. 1. Cambridge: Cambridge university press.

Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM, 38*(11), 39–41.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The Pagerank Citation Ranking: Bringing Order to the Web. *Technical report 1999-66 Stanford InfoLab*, http://ilpubs.stanford.edu:8090/422/.

Rada, R., Mili, H., Bicknell, E., & Blettner, M. (1989). Development and application of a metric on semantic nets. *Systems, Man and Cybernetics, IEEE Transactions on, 19*(1), 17–30.

Rafter, R., Bradley, K., & Smyth, B. (2000). Personalised retrieval for online recruitment services. In *Proceedings of the 22nd annual colloquium on information retrieval*.

Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th international joint conference on artificial intelligence-volume 1* (pp. 448–453). Morgan Kaufmann Publishers Inc.

Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M. M., Gatford, M., et al. (1995). Okapi at trec-3. *Overview of the Third Text Retrieval Conference (TREC-3)* (pp. 109–126). National Institute of Standards & Technology (NIST).

Roche, E., & Schabes, Y. (1997). *Finite-state language processing*. Cambridge, Massachusetts: MIT Press.

Rodríguez, M. A., & Egenhofer, M. J. (2003). Determining semantic similarity among entity classes from different ontologies. *Knowledge and Data Engineering, IEEE Transactions on, 15*(2), 442–456.

Sánchez, D., Batet, M., Isern, D., & Valls, A. (2012). Ontology-based semantic similarity: A new feature-based approach. *Expert Systems with Applications, 39*(9), 7718–7728.

Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. (2007). The Adaptive Web: Methods and Strategies of Web Personalization. In *Collaborative Filtering Recommender Systems* (pp. 291–324). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-540-72079-9_9.

Singh, A., Rose, C., Visweswariah, K., Chenthamarakshan, V., & Kambhatla, N. (2010). Prospect: a system for screening candidates for recruitment. In *Proceedings of the 19th ACM international conference on information and knowledge management* (pp. 659–668). ACM.

Trewin, S. (2000). Knowledge-based recommender systems. *Encyclopedia of Library and Information Science: Volume 69-Supplement 32* p. 180. Marcel Dekker, Inc New York.

Tversky, A. (1977). Features of similarity. *Psychological review, 84*(4), 327.

Wei, K., Huang, J., & Fu, S. (2007). A survey of e-commerce recommender systems. In *Service systems and service management, 2007 international conference on* (pp. 1–5). IEEE.

Wu, Z., & Palmer, M. (1994). Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on association for computational linguistics* (pp. 133–138). Association for Computational Linguistics.

Yi, X., Allan, J., & Croft, W. B. (2007). 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* (pp. 809–810). ACM.

Yu, K., Guan, G., & Zhou, M. (2005). Resume information extraction with cascaded hybrid model. In *Proceedings of the 43rd annual meeting on association for computational linguistics* (pp. 499–506). Association for Computational Linguistics.

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

Zobel, J., & Moffat, A. (2006). Inverted files for text search engines. *ACM computing surveys (CSUR), 38*(2), 6.