RÉSUMATCHER: A PERSONALIZED RÉSUMÉ-JOB MATCHING SYSTEM

A Thesis

by

SHIQIANG GUO

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Tracy Hammond
Committee Members,	Anxiao Jiang
	Daniel W. Goldberg
Head of Department,	Dilma Da Silva

May 2015

Major Subject: Computer Science

ABSTRACT

Today, online recruiting web sites such as Monster and Indeed.com have become one of the main channels for people to find jobs. These web platforms have provided their services for more than ten years, and have saved a lot of time and money for both job seekers and organizations who want to hire people. However, traditional information retrieval techniques may not be appropriate for users. The reason is because the number of results returned to a job seeker may be huge, so job seekers are required to spend a significant amount of time reading and reviewing their options. One popular approach to resolve this difficulty for users are recommender systems, which is a technology that has been studied for a long time.

In this thesis we have made an effort to propose a personalized job-résumé matching system, which could help job seekers to find appropriate jobs more easily. We create a finite state transducer based information extraction library to extract models from résumés and job descriptions. We devised a new statistical-based ontology similarity measure to compare the résumé models and the job models. Since the most appropriate jobs will be returned first, the users of the system may get a better result than current job finding web sites. To evaluate the system, we computed Normalized Discounted Cumulative Gain (NDCG) and precision@k of our system, and compared to three other existing models as well as the live result from Indeed.com.

ACKNOWLEDGEMENTS

My greatest thanks to the members of the Sketch Recognition Lab for their continued support and help in the research work covered in this thesis. This thesis would not have been possible without their support. In addition, I would like to give extra thanks to my advisor Dr. Tracy Hammond, as well as to my committee members Dr. Anxiao Jiang and Dr. Daniel W. Goldberg for their valuable sage advice.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

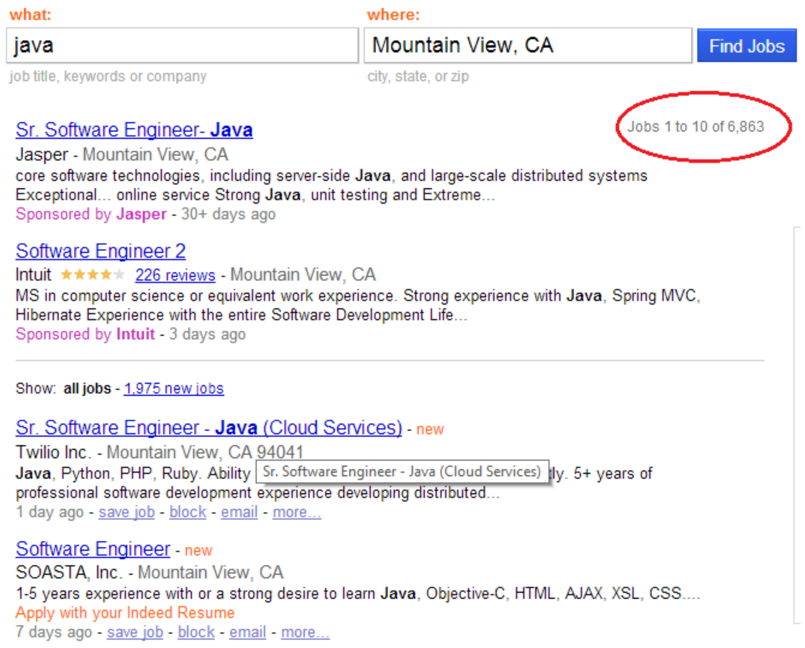
1. INTRODUCTION

1.1 Motivation

Currently one of the main channels for job seekers is online job finding web sites, like Indeed or Monster etc, that make the job finding process easier and decrease the recruitment time. But most such web sites only allow users to use keywords to search the jobs, which makes job searching tedious and blind task. For example, I used keyword “Java” to search jobs with location restriction Mountain View, CA on the job searching site indeed.com, the web site returned about 7,000 jobs (Figure 1.1). The number of results of job searching is huge but not well ranked, so the job seeker has to review every job description. Since no one has enough time to read all the jobs in the searching result, the actual quality of job searching service is low. This is a classic problem of information overflow.

The reason for such a result is because current job searching web sites use the same information retrieval technology like “Inverted index” [53] as the common search engines, which just use keywords to map all the stored documents. Modern search engines all have some ranking algorithms to sort the search result, like page rank [33], so the top results always be the most related ones. But such algorithms are unavailable to the job search systems, because the criteria of how to rank the job searching result is very personalized. A great job opening for one job seeker maybe looks not good to the other, because the goodness of a job to a specular job seeker is heavily depend on his personal background, like his education or professional experience etc.

Since the people’s résumés contain the most important background information, we believe the content of the résumé could be used to rank the job openings. We give


what: java
job title, keywords or company
where: Mountain View, CA
city, state, or zip
Find Jobs

Jobs 1 to 10 of 6,863

Sr. Software Engineer- **Java**
Jasper - Mountain View, CA
core software technologies, including server-side **Java**, and large-scale distributed systems Exceptional... online service Strong **Java**, unit testing and Extreme...
Sponsored by **Jasper** - 30+ days ago

Software Engineer 2
Intuit ★★★★☆ 226 reviews - Mountain View, CA
MS in computer science or equivalent work experience. Strong experience with **Java**, Spring MVC, Hibernate Experience with the entire Software Development Life...
Sponsored by **Intuit** - 3 days ago

Show: **all jobs** - 1,975 new jobs

Sr. Software Engineer - **Java** (Cloud Services) - new
Twilio Inc. - Mountain View, CA 94041
Java, Python, PHP, Ruby. Ability Sr. Software Engineer - Java (Cloud Services) dly. 5+ years of professional software development experience developing distributed...
1 day ago - save job - block - email - more...

Software Engineer - new
SOASTA, Inc. - Mountain View, CA
1-5 years experience with or a strong desire to learn **Java**, Objective-C, HTML, AJAX, XSL, CSS....
Apply with your Indeed Resume
7 days ago - save job - block - email - more...

Figure 1.1: Search Result of Indeed.com

an example of résumés in Table 1.1. In this thesis, we created a web system which uses the résumés of job seekers to find the jobs that match their profiles best. The main idea is to calculate the similarity between the résumé model and job models, which should be generated from résumés and job descriptions. We want to transfer the job searching task from key word searching to candidate model matching. The matching result should be sorted by the matching score, higher matching score means a better matching. The matching algorithm does not only help job seekers to find the appropriate jobs, but also offers priority to them [18]. The job with higher matching score means the job is more appropriate to the job seeker, and if he applies to the job, the chance of getting the interview will be higher as well. Figure 1.2 shows how this approach works.

Table 1.1: Résumé Example

Ryan Richman

WORK EXPERIENCE

Web Developer
Fabuso/Advanced Brain Technologies - Ogden, UT - February 2012 to Present
Created dynamic custom web applications for e-commerce and B2B clients.
Designed and edited audio-visual content for many different online applications.
Spearheaded migration of largest client's website from Joomla platform into Java code base.
Built dynamic event pages, document viewers, training course applications, shopping carts, and more.
Utilized advanced e-mail standards and best practices, SQL database queries, and Google Analytics.

IT Representative
Advanced Brain Technologies - Ogden, UT - April 2011 to February 2012
Provided internal software/hardware support for 20 employees both in-house and remote.
Designed using wireframes, tested, and debugged web pages.
Constructed dynamic projects and graphic designs in coordination with senior developers.
Created HTML-optimized emails for hundreds of campaigns.
Maintained and upgraded hardware for 20+ workstations company-wide.

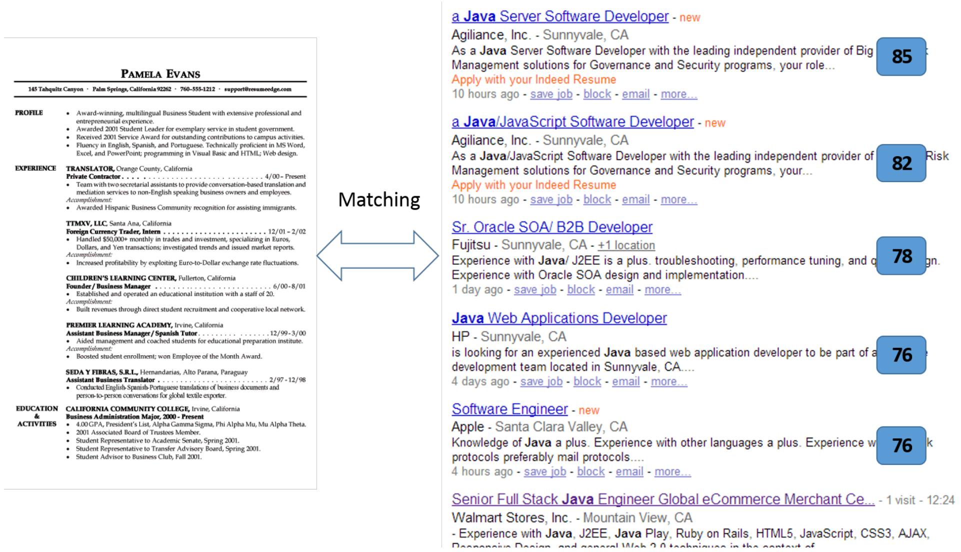
Founder/Head Technician
Teton Media Services, LLC - Ogden, UT - October 2008 to January 2011
Created and developed websites for personal and small business customers
Sold high-speed cable and satellite internet access on the phone, online and in person.
Installed and serviced high-speed internet access hardware in residential and commercial properties.
Designed and implemented networking solutions for homes and businesses.

EDUCATION

Computer Science
Weber State University - Ogden, UT 2010 to 2013

ADDITIONAL INFORMATION

Technical Skills
Adept in the use of HTML, CSS, jQuery, Javascript, SQL, PHP, JSON, Windows, Windows Server, Mac, and Photoshop.



Pamela Evans

145 Tahquitz Canyon · Palm Springs, California 92262 · 760-555-1212 · support@resumeedge.com

PROFILE
- Award-winning, multilingual Business Student with extensive professional and entrepreneurial experience.
- Awarded 2001 Student Leader for exemplary service in student government.
- Received 2001 Service Award for outstanding contributions to campus activities.
- Fluency in English, Spanish, and Portuguese. Technically proficient in MS Word, Excel, and PowerPoint; programming in Visual Basic and HTML; Web design.

EXPERIENCE

TRANSLATOR, Orange County, California
Private Contractor . . . . 4/00 – Present
- Team with two secretarial assistants to provide conversation-based translation and mediation services to non-English speaking business owners and employees.

Accomplishment:
- Awarded Hispanic Business Community recognition for assisting immigrants.

TTMXV, LLC, Santa Ana, California
Foreign Currency Trader, Intern . . . . 12/01 – 2/02
- Handled $50,000+ monthly in trades and investment, specializing in Euros, Dollars, and Yen transactions; investigated trends and issued market reports.

Accomplishment:
- Increased profitability by exploiting Euro-to-Dollar exchange rate fluctuations.

CHILDREN'S LEARNING CENTER, Fullerton, California
Founder / Business Manager . . . . 6/00 - 8/01
- Established and operated an educational institution with a staff of 20.

Accomplishment:
- Built revenues through direct student recruitment and cooperative local network.

PREMIER LEARNING ACADEMY, Irvine, California
Assistant Business Manager / Spanish Tutor . . . . 12/99 - 3/00
- Aided management and coached students for educational preparation institute.

Accomplishment:
- Boosted student enrollment; won Employee of the Month Award.

SEDA Y FIBRAS, S.R.L., Hernandarias, Alto Parana, Paraguay
Assistant Business Translator . . . . 2/97 - 12/98
- Conducted English-Spanish-Portuguese translations of business documents and person-to-person conversations for global textile exporter.

EDUCATION & ACTIVITIES

CALIFORNIA COMMUNITY COLLEGE, Irvine, California
Business Administration Major, 2000 - Present
- 4.00 GPA, President's List, Alpha Gamma Sigma, Phi Alpha Mu, Mu Alpha Theta.
- 2001 Associated Board of Trustees Member.
- Student Representative to Academic Senate, Spring 2001.
- Student Representative to Transfer Advisory Board, Spring 2001.
- Student Advisor to Business Club, Fall 2001.

a **Java** Server Software Developer - new
Agiliance, Inc. - Sunnyvale, CA
As a **Java** Server Software Developer with the leading independent provider of Big
Management solutions for Governance and Security programs, your role...
Apply with your Indeed Resume
10 hours ago - save job - block - email - more...

a **Java**/JavaScript Software Developer - new
Agiliance, Inc. - Sunnyvale, CA
As a **Java**/JavaScript Software Developer with the leading independent provider of Risk
Management solutions for Governance and Security programs, your...
Apply with your Indeed Resume
10 hours ago - save job - block - email - more...

Sr. Oracle SOA/ B2B Developer
Fujitsu - Sunnyvale, CA - +1 location
Experience with **Java**/ J2EE is a plus. troubleshooting, performance tuning, and q gn.
Experience with Oracle SOA design and implementation....
1 day ago - save job - block - email - more...

Java Web Applications Developer
HP - Sunnyvale, CA
is looking for an experienced **Java** based web application developer to be part of a
development team located in Sunnyvale, CA....
4 days ago - save job - block - email - more...

Software Engineer - new
Apple - Santa Clara Valley, CA
Knowledge of **Java** a plus. Experience with other languages a plus. Experience w k
protocols preferably mail protocols....
4 hours ago - save job - block - email - more...

Senior Full Stack **Java** Engineer Global eCommerce Merchant Ce... - 1 visit - 12:24
Walmart Stores, Inc. - Mountain View, CA
- Experience with **Java**, J2EE, **Java** Play, Ruby on Rails, HTML5, JavaScript, CSS3, AJAX,

Figure 1.2: Matching the Jobs with Résumé

1.2 Contribution

We make the following contributions in this work:

1. We proposed a résumé - job matching system.

2. We proposed a finite state transducer based matching tool to extract information from unstructured data source, which is a lightweight and flexible library, and can be extended in very easy ways.

3. We proposed a semi-automatic approach, which can collect technical terms from hr data sources, and by which we created a domain specific ontology for recruitment.

4. We proposed statistical-based ontology similarity measure, which can measure the similarities between technical terms .

1.3 Organization

The subsequent chapters are organized as follows: we first describe what has been done in terms of prior work. We introduce some basic conception of recommender systems, and how to apply recommender technologies into Job Recommender Systems. Some previous Job Recommender Systems will be introduced, their advantages and limits will be discussed as well. Two import problems of content-based Job Recommender Systems, Information Extraction and Similarity Calculation, will be fully explained.

Then we introduce our work, RésuMatcher, a the Personalized Résumé-Job Matching System. First, we give an overview of the system, which includes the architecture and the interfaces. Then we explained details of how we resolve the problems of information extraction and model similarity calculation. We propose a finite state transducer library which can match patterns in sentence, and extracts related information. Ontology plays an important role in this system. We will present how to construct the domain specific ontology for recruitment. We also give a brief review of different ontology similarity measures, and explain the statistical-based ontology similarity measure we used in this system.

Finally, we evaluate the accuracy of our information extraction approach. We used NDCG to evaluate the accuracy of statistical-based ontology similarity measure. To evaluate the performance of the system, we compared our algorithm to some classical information retrieval approaches by precision@k and NDCG. We created a data set of job descriptions as documents, and use résumés as query to retrieval documents. The result shows the ranking performance is better than other information retrieval approaches.

2. BACKGROUND

Some scholars found that current boolean search and filtering techniques cannot satisfy the complexity of candidate-job matching requirement [29]. They hope the system can understand the job requirement, determine which requirements are mandatory and which are optional, but preferable. So they moved to use recommender systems technique to address the problem of information overflow. Recommender systems are broadly accepted in various areas to suggest products, services, and information items to latent customers.

2.1 Recommender System

Job searching, which has been the focus of some commercial job finding web sites and research papers is not a new topic in information retrieval. Usually scholars called them Job Recommender Systems (JRS), because most of them used technologies from recommender systems. Wei et al. classified Recommender Systems into four categories [48]: Collaborative Filtering, Content-based filtering, Knowledge-based and Hybrid approaches. Some of these techniques had been applied into JRS; Zheng et al. [44] and AlOtaibi et al. [3] summarized the categories of existing online recruiting platforms and listed the advantages and disadvantages of technical approaches in different JRSs. The categories include:

1. Content-based Recommendation (CBR). The principle of a content-based recommendation is to suggest items that have similar content information to the corresponding users, like Prospect [43].

2. Collaborative Filtering Recommendation (CFR). Collaborative filtering recommendation finds similar users who have the same taste with the target user and

recommends items based on what the similar users, like CASPER [35].

3. Knowledge-based Recommendation (KBR). In the knowledge-based recommendation, rules and patterns obtained from the functional knowledge of how a specific item meets the requirement of a particular user, are used for recommending items, like Proactive [24].

4. Hybrid recommender systems combine two or more recommendation techniques to gain better performance, and overcome the drawbacks of any individual one. Usually, collaborative filtering is combined with some other technique in an attempt to avoid the ramp-up problem.

2.2 Job Recommender Systems

Rafter et al. began to use Automated Collaborative Filtering (ACF) in their Job Recommender System, "CASPER" [35]. In the system user profiles are gotten from server logs, that includs: revisit data, read time data, and activity data. All these factors are viewed as measure of relevance among users. The system recommend jobs in two steps: First, the system finds a set of users related to the target user; second, the jobs that related users liked will be recommend to the target user. The system use cluster-based collaborative filtering strategy. The similarity between users are based on how many jobs they both reviewed, or applied.

CASPER also allows users to search jobs by a query which is a combination of some fields: like location, salary, skill and so on. The system uses such query to find jobs, and the returned jobs are ranked with the collaborative filtering algorithm. In their paper, the authors do not give a detailed description on how to detect the related fields they need and how to the transfer semi-structured job description to the structured data.

The shortages of collaborative filtering: First, since the number of search results is huge, and the results are sorted randomly, the probability of two similar users reviewing the same jobs is low, which causes the sparsity problem of collaborative filtering. The authors also noticed the sparseness problem caused by few in users profile, so they try to user cluster-based solution to resolve this problem. Second, because recommended jobs are from others users' search results, since the quality of current searching result are low, the quality of recommendation cannot be high.

Färber et al. [14] presented a recommender system built on a hybrid approach. The system integrate two methods, content-based filtering and collaborative filtering, which tries to overcome the problem of rating data sparsity by leveraging a combined model. In the system, the data source is synthetic resumes. The latent aspect model is shown in Figure 2.1.

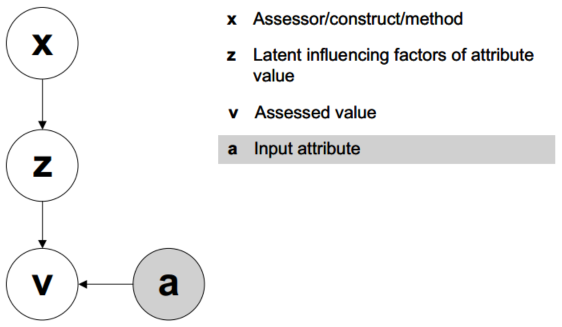

x
z
v
a
x Assessor/construct/method
z Latent influencing factors of attribute value
v Assessed value
a Input attribute

Figure 2.1: Latent Aspect Model

In Malinowski et al. [29], they classified the job recommender systems into t-wo categories, CV-recommender, which recommends CVs to recruiter, and the job-recommender, which recommends jobs to job seekers. The system collects the users' profile data by asking input their profiles to the web form based interface field by field. The input data collected are:

1. Demographic data (e.g. date of birth, contact information)

2. Educational data (e.g. school courses, grades, university, type of degree, intermediate and final university examinations, postgraduate studies)

3. Job experience (e.g. name of the company, type of employment, industry group, occupational field)

4. Language skills (e.g. language, level of knowledge)

5. IT skills (e.g. type of skill, level of knowledge)

6. Awards, scholarships, publications, others

The system also asked the users to upload their resumes, but the resumes were only for facilitating the human judgment. In Malinowski's study, the latent aspect model is a statistical model, which needs to be trained before applied to recommendation. The system uses the users' search results as the training data to train the model for recommendation, so the system needs a a long time training for each user.

2.3 Information Extraction in Job Recommender System

Big IT companies met the similar problem of information overflow when they received many resumes for one job opening. The recruiter had to screen the all the applications manually, but this task was also tedious and time consuming. For this reason these companies tried to build systems to help screen the resumes.

Amit et al. in IBM presented a system, "PROSPECT" [43], to aid shortlisting candidates for jobs. The system uses a résumé miner to extract the information from résumés, which use a conditional random field (CRF) model to segment and label the résumés. The CRF model used three kinds of features, they are: Lexicon, Visual, and Named Entity. The paper compared some algorithms to ranked the candidates applicants, such methods include: Okapi BM25, KL, Lucene Scoring, and Lucene Scoring + SkillBoost.

HP also built a system to solve the similar problem, which is introduced in Gonzalez et al.'s paper [17]. The system also pays a lot of attention to information extraction.

The dictionaries which are used to tag entities need to be updated often since there always new terms appears. So an adaptive learning module is used to achieve two objectives: use semantic data to enhance the information extraction and to discover new terms.

A domain-oriented ontology is used to represent knowledge, inference rules are defined based on the ontology knowledge base. When a detected term found, the system will search in external knowledge base, like DBpedia. The résumés are also classified to different categories like "Web Technology" and "No Web Technology" by naive Bayes classifier. The company can allocate appropriate employees to required positions with the system.

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.

Yu et al. [51] used a cascaded IE framework to get the detailed information from the résumés. In the first stage, the Hidden Markov Modeling (HMM) model is used to segment the résumé into consecutive blocks. Based on the result, a SVM model is used to obtain the detailed information in the certain block, the information include: name, address, education etc.

Celik Duygua and Elci Atilla proposed a Ontology-based Rsum Parser (ORP) [7], which uses ontology to assistant the information extraction process. The system processes a résumé in following steps: converting the résumé file into plain text, separating the text into some segments, using the ontology knowledge base to find the concepts in the sentences, normalizing all the terms, and classifing the sentences to get the wanted terms.

But the personal information the system retrieved like name and addresses is not the information that the recruiters care about. The recruiters want some information that relate to the job opening, and can help them to judge the competence of job applicants.

2.4 Matching Algorithms in Job Recommender Systems

Lu et al. [28] used latent semantic analysis (LSA) to calculate similarities between jobs and candidates, but they only selected two factors "interest" and "education" to compare candidates. Xing et al. [50] used structured relevance models (SRM) to match résumés and jobs.

Drigas et al.[13] presented a expert system to match jobs and job seekers, and to recommend unemployed to the positions. The expert system used Neuro-Fuzzy rules to evaluate the matching between user profiles and job openings. The system uses a relation matrix to represent the fuzzy relation between these specialities. The system needs the training data to train that Neuro-Fuzzy network. Both résumé data and job opening data were manually input into the system.

Daramola et al.[10] also proposed a fuzzy logic based expert system(FES) tool for online personnel recruitment. In the paper, the authors assumed that the information already be collected. The system uses a fuzzy distance metric to rank candidates' profiles in the order of their eligibility for the job.

Yao et al. [28] also presented a hybrid recommender system which integrated content-based and interaction-based relation. In content-based part, relations between job-job, job-job seeker, and job seeker-job seeker can be identified by their similarity of profiles. There two approaches are used to calculate the similarities: For the structured data, like age and gender, the weight sum values will be returned; for the unstructured data, like similarity between job and user profile, the latent semantic analysis is applied in the system.

In summary, there are some problems exist in previous Job Recommender Systems:

1. Most systems can only process the structured data of résumés and job descriptions, but in reality both them are in unstructured formats, such as text files or HTML web pages.

2. The systems that have information extraction modules are designed for recruiters to select applicants, not for job seekers to select jobs.

3. In the systems the information fields to match résumés and job descriptions are coarse-grained. To improve the quality of recommendation, we need to improve the granularity of the information fields.

3. PROBLEM

3.1 Problem Definition

The basic problem in this thesis is how to find appropriate job descriptions by user's résumé, which means we need calculate the similarity between the users rsum and the job. If we take the résumé as a query and the job descriptions as documents, we need to build an information retrieval model to get the most relevance documents. The RésuMatcher will parse the job descriptions to the job models, and store them in the database. When a user searches the jobs by their résumé in the system, the system will compare the similarity values between the résumé and the job models, and return the jobs sorted by their similarity values.

The core idea of our algorithm is calculate similarity between the résumé model and job model. We give a formal definition of our problem. All of the notations will be used frequently throughout the thesis.

We use r to denote the user's résumé model, which has some features r_i like their academic degree, their major, their skills and so on. The symbol J is the set of job models stored in the database, and j is a job model in the set J. The similarity function $sim(r, j)$ gives the similarity values between résumé r and job j. The return list of search function $search(r, J)$ will calculate all the similarity value in the database, and the result of the function will be the job description list ranked by their similarity values. The equation of how to calculate similarity value is given below:

$$sim(r, j) = \sum_{i=1}^{n} simfun_i(r_i, j_i) \times w_i$$

The value of $sim(r, j)$ is the summation of the similarity values of different fields times their corresponding weights. Different fields like major and skills, may have different functions to calculate their similarity values. We will describe the similarity functions of individual fields in later parts.

3.2 Challenges

There are two challenges exist in our system. The first one is how to extract models of jobs and résumés. To calculate the similarity between a job and a resume, the RésuMatcher system needs structured digital models of each document. To get the structured data, some JRSs ask the job seekers input their profiles in forms field by field, and the recruiter input their job descriptions in the same way. However, as we discussed in Chapter 2, the users are reluctant to take the tedious process [43]. Job seekers prefer upload their resumes directly, and recruiters prefer to post the whole job descriptions to web sites. So we need extract the structured information from un-structured data source, like résumés and job descriptions.

The other challenge is how to compute the similarity between rsum and job models. 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 3.1 shows portions of a résumé and a job description:

If just looking at the text, we can find that the résumé has very few common words with the job description. But from the view of an experienced engineer, the candidate is closely matches the job: the two relational databases Oracle and Mysql are very similar, OOA/OOD is the same meaning of many years of Java and C++ experience, and Tomcat and JBOSS are both Java web applications servers. If we

Table 3.1: Portions of Resume and Job Description

Resume Portion	**Job Description Portion**
B.S. degree in computer science	BS degree above
5+ years Java	4+ years Java
2+ year C++	Some experience of Python
Some experience in Oracle database	Mysql, MS-SQL
Other experience like:	Java web application Server
Hibernate, JBOSS, JUnit, Tomcat etc.	OOA/OOD

use keyword matching, the system does not provide a strong matching result in very common cases such as this. So we need a better approach to calculate the similarity between different technical concepts.

4. SYSTEM OVERVIEW

4.1 System Overview

The system uses information extraction technique to parse job descriptions and résumés, and it gets information such as skills, job titles and education background. The information is used to create the models of job openings and job seekers. A domain specific ontology is used to construct the knowledge base, which includes the taxonomies that support résumé-job matching.

The models of résumé includes job seekers' specialties, working experience and education background, and all the fields are extracted from their résumés. The job models are extracted from job descriptions, and they have the same information fields as the résumé models. When a job seeker searches the jobs by their résumé, the system calculates the similarity between the résumé model and job models, then gives every job model a similarity value.

4.2 System Architecture

Figure 4.2 shows the architecture of the whole system, which includes such modules:

1. The Web Crawler can access and download all new IT job opening web pages from indeed.com everyday.

2. The Job Parser can parse the job opening web pages, extract the information and create the job models.

3. The Resume Parser is much like the Job Parser; it parses the résumés and creates the résumé models.

4. All the job descriptions and job models are stored in the database.

5. When a user searches the jobs with their résumé, the Ontology Matcher calculates the similarity values of jobs in the database and returns the jobs ranked by their similarity values.

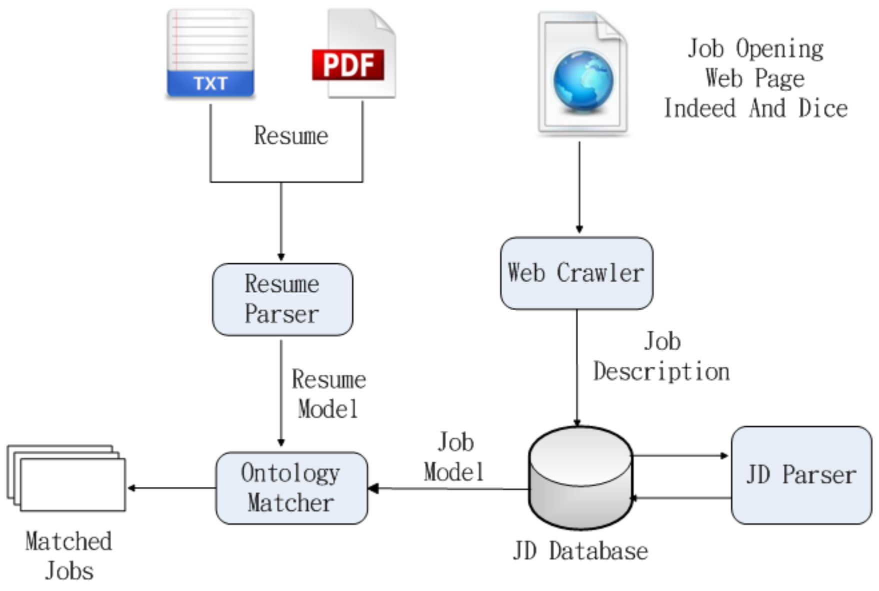


Figure 4.1: System Architecture

4.3 Text Processing Stages

Information Extraction is the task of automatically extracting structured information such as entities, relationships between entities, and attributes describing entities from unstructured sources [42]. The IE framework in our system uses six stages in order to extract the information from job descriptions: HTML parsing, segmenting, preprocessing, tokenizing, labeling and pattern matching, which is show in Figure 4.2.

1) The **HTML Parsing** will parse the web pages that contain job descriptions, which are obtained from web crawler. The parser uses HTML tag template to extract attributes of the jobs, like job title, location, company name, content and so on. A job will be saved as a record with these attributes in the database. In the record, the content field contains the text part of the job description, which will be processed in later stages.

2) In the **segmenting stage**, the content field of the job description is be separated into paragraphs according HTML tags. Then paragraphs are separated into sentences by either HTML tags or punctuation, and after this step, all HTML tags will be removed.

3) Web pages of job description are created in different character sets, (e.g. UTF8 and ISO 8859-1), and almost always contain some unreadable characters. In the **prepossessing stage**, characters in the sentences are converted to ASCII characters, unreadable characters will be deleted, and some punctuation will be replaced by spaces (e.g. / and -).

4) In the **tokenizing stage**, the sentences will be tokenized into arrays of tokens by NLTK [5].

5) In the **labeling stage**, the sentences will be given two layers of labels by a dictionary matching approach. The labels in the first layer are the semantic value of the text, and the labels in the second layer are the ontology hypernym of the labels in the first layers.

6) In the **pattern matching** stage, the FST library is used to matching the labels of the labeled sentences. If a layered sentence match any pre-defined pattern, the information will be extracted and added to the job model. After every sentence of a job description has be processed, a job model will be created and saved in the database. More details about matching will follow in Section C.

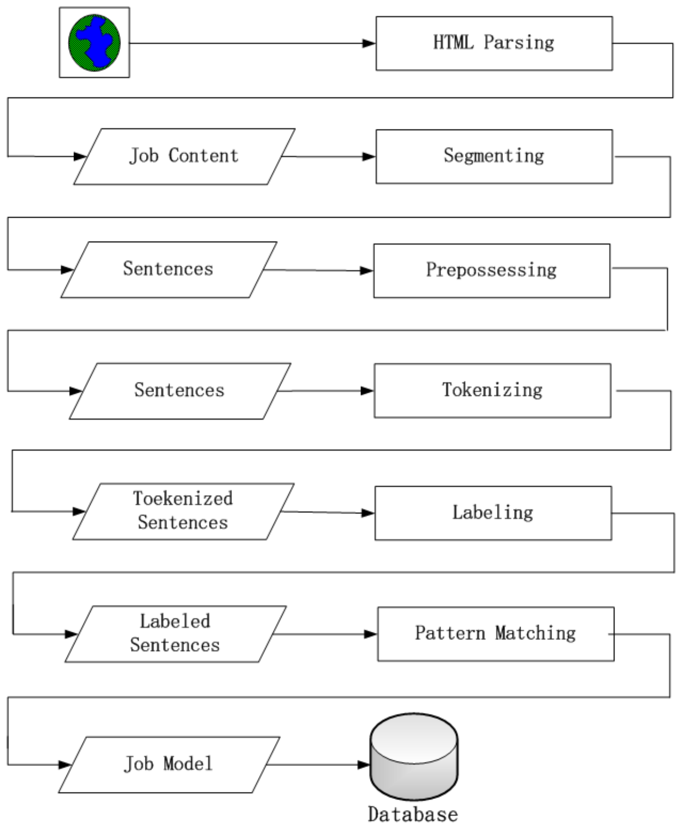


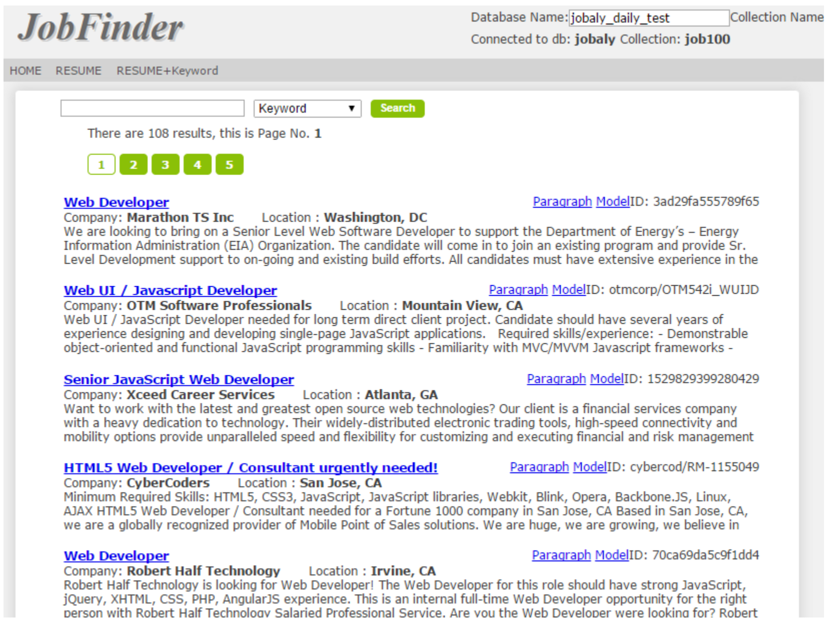
Figure 4.2: Job Description Process Pipeline

4.4 System Implementation

We will describe some implementation details here. The whole system is implemented in Python and uses some third party libraries and frameworks. We used Flask, a lightweight web framework, to build the web application. We used Rdflib as the Web Ontology Language (OWL) file parser, Python Lex-Yacc (PLY) as the token regular expression compiler, whoosh as the inverted index builder and Beautiful Soup as the HTML parser. All the jobs retrieved by the Web Crawler are stored in the MongoDB NoSQL database. For the natural language processing procedure, we used Natural Language Toolkit (NLTK), a natural language processing library, to extract and tokenize the sentences.

4.5 System Interface

The system provides some interfaces to end users. The most important interfaces are the web pages like: reviewing all the jobs in the database, searching the jobs by keyword (Figure 4.3), uploading users' résumés (Figure 4.4) and matching the jobs with a résumé (Figure 4.5).



Database Name: jobaly_daily_test Collection Name
Connected to db: **jobaly** Collection: **job100**

There are 108 results, this is Page No. **1**

Web Developer Paragraph ModelID: 3ad29fa555789f65
Company: **Marathon TS Inc** Location : **Washington, DC**
We are looking to bring on a Senior Level Web Software Developer to support the Department of Energy's – Energy Information Administration (EIA) Organization. The candidate will come in to join an existing program and provide Sr. Level Development support to on-going and existing build efforts. All candidates must have extensive experience in the

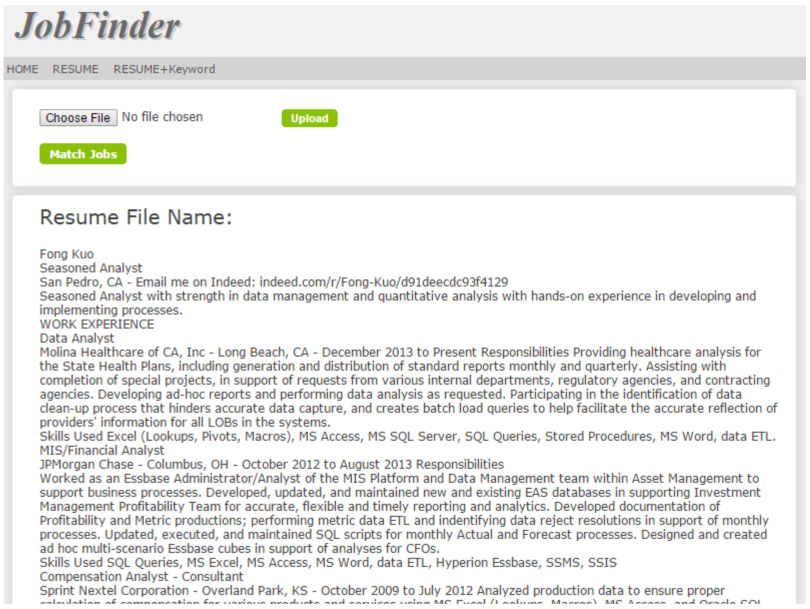
Web UI / Javascript Developer Paragraph ModelID: otmcorp/OTM542i_WUIJD
Company: **OTM Software Professionals** Location : **Mountain View, CA**
Web UI / JavaScript Developer needed for long term direct client project. Candidate should have several years of experience designing and developing single-page JavaScript applications. Required skills/experience: - Demonstrable object-oriented and functional JavaScript programming skills - Familiarity with MVC/MVVM Javascript frameworks -

Senior JavaScript Web Developer Paragraph ModelID: 1529829399280429
Company: **Xceed Career Services** Location : **Atlanta, GA**
Want to work with the latest and greatest open source web technologies? Our client is a financial services company with a heavy dedication to technology. Their widely-distributed electronic trading tools, high-speed connectivity and mobility options provide unparalleled speed and flexibility for customizing and executing financial and risk management

HTML5 Web Developer / Consultant urgently needed! Paragraph ModelID: cybercod/RM-1155049
Company: **CyberCoders** Location : **San Jose, CA**
Minimum Required Skills: HTML5, CSS3, JavaScript, JavaScript libraries, Webkit, Blink, Opera, Backbone.JS, Linux, AJAX HTML5 Web Developer / Consultant needed for a Fortune 1000 company in San Jose, CA Based in San Jose, CA, we are a globally recognized provider of Mobile Point of Sales solutions. We are huge, we are growing, we believe in

Web Developer Paragraph ModelID: 70ca69da5c9f1dd4
Company: **Robert Half Technology** Location : **Irvine, CA**
Robert Half Technology is looking for Web Developer! The Web Developer for this role should have strong JavaScript, jQuery, XHTML, CSS, PHP, AngularJS experience. This is an internal full-time Web Developer opportunity for the right person with Robert Half Technology Salaried Professional Service. Are you the Web Developer were looking for? Robert

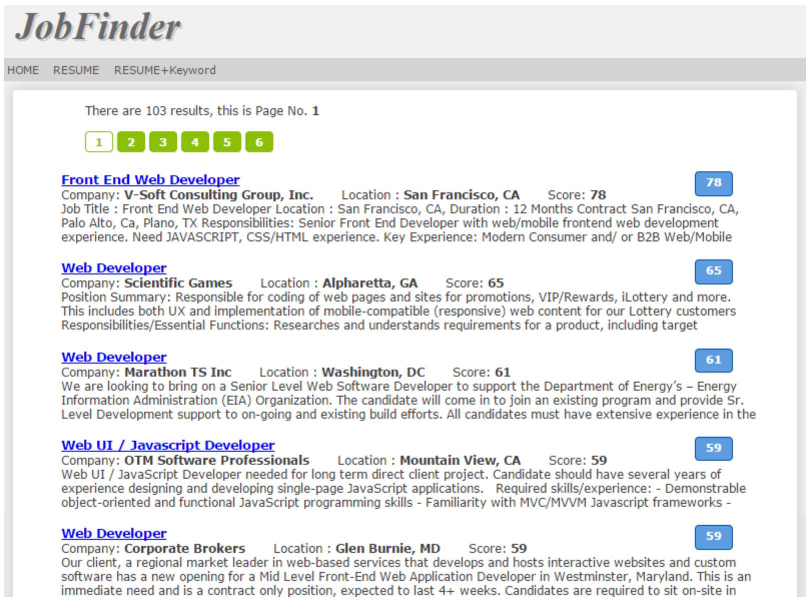
Figure 4.3: Job Description List



Resume File Name:

Fong Kuo
Seasoned Analyst
San Pedro, CA - Email me on Indeed: indeed.com/r/Fong-Kuo/d91deecdc93f4129
Seasoned Analyst with strength in data management and quantitative analysis with hands-on experience in developing and implementing processes.
WORK EXPERIENCE
Data Analyst
Molina Healthcare of CA, Inc - Long Beach, CA - December 2013 to Present Responsibilities Providing healthcare analysis for the State Health Plans, including generation and distribution of standard reports monthly and quarterly. Assisting with completion of special projects, in support of requests from various internal departments, regulatory agencies, and contracting agencies. Developing ad-hoc reports and performing data analysis as requested. Participating in the identification of data clean-up process that hinders accurate data capture, and creates batch load queries to help facilitate the accurate reflection of providers' information for all LOBs in the systems.
Skills Used Excel (Lookups, Pivots, Macros), MS Access, MS SQL Server, SQL Queries, Stored Procedures, MS Word, data ETL.
MIS/Financial Analyst
JPMorgan Chase - Columbus, OH - October 2012 to August 2013 Responsibilities
Worked as an Essbase Administrator/Analyst of the MIS Platform and Data Management team within Asset Management to support business processes. Developed, updated, and maintained new and existing EAS databases in supporting Investment Management Profitability Team for accurate, flexible and timely reporting and analytics. Developed documentation of Profitability and Metric productions; performing metric data ETL and indentifying data reject resolutions in support of monthly processes. Updated, executed, and maintained SQL scripts for monthly Actual and Forecast processes. Designed and created ad hoc multi-scenario Essbase cubes in support of analyses for CFOs.
Skills Used SQL Queries, MS Excel, MS Access, MS Word, data ETL, Hyperion Essbase, SSMS, SSIS
Compensation Analyst - Consultant
Sprint Nextel Corporation - Overland Park, KS - October 2009 to July 2012 Analyzed production data to ensure proper

Figure 4.4: Upload Resume



There are 103 results, this is Page No. **1**

[Front End Web Developer]
Company: **V-Soft Consulting Group, Inc.** Location : **San Francisco, CA** Score: **78**
Job Title : Front End Web Developer Location : San Francisco, CA, Duration : 12 Months Contract San Francisco, CA, Palo Alto, Ca, Plano, TX Responsibilities: Senior Front End Developer with web/mobile frontend web development experience. Need JAVASCRIPT, CSS/HTML experience. Key Experience: Modern Consumer and/ or B2B Web/Mobile

[Web Developer]
Company: **Scientific Games** Location : **Alpharetta, GA** Score: **65**
Position Summary: Responsible for coding of web pages and sites for promotions, VIP/Rewards, iLottery and more. This includes both UX and implementation of mobile-compatible (responsive) web content for our Lottery customers Responsibilities/Essential Functions: Researches and understands requirements for a product, including target

[Web Developer]
Company: **Marathon TS Inc** Location : **Washington, DC** Score: **61**
We are looking to bring on a Senior Level Web Software Developer to support the Department of Energy's – Energy Information Administration (EIA) Organization. The candidate will come in to join an existing program and provide Sr. Level Development support to on-going and existing build efforts. All candidates must have extensive experience in the

[Web UI / Javascript Developer]
Company: **OTM Software Professionals** Location : **Mountain View, CA** Score: **59**
Web UI / JavaScript Developer needed for long term direct client project. Candidate should have several years of experience designing and developing single-page JavaScript applications. Required skills/experience: - Demonstrable object-oriented and functional JavaScript programming skills - Familiarity with MVC/MVVM Javascript frameworks -

[Web Developer]
Company: **Corporate Brokers** Location : **Glen Burnie, MD** Score: **59**
Our client, a regional market leader in web-based services that develops and hosts interactive websites and custom software has a new opening for a Mid Level Front-End Web Application Developer in Westminster, Maryland. This is an immediate need and is a contract only position, expected to last 4+ weeks. Candidates are required to sit on-site in

Figure 4.5: Résumé Job Matching

5. INFORMATION EXTRACTION

In this chapter we will explain how the Information Extraction (IE) module of our system extracts information from these unstructured data source. An example of job description is shown in Table 5.1. The IE framework will be introduced by example of processing the job descriptions. The Finite-State Transducer(FST) library, which is used as pattern matching tools, will be introduced as well.

5.1 Semantic Labeling

In this section, we will introduce why and how we add two layers of labels to the tokenized sentences. In natural language, a single concept often has multiple expressions to represent it. For example, the simple concept bachelor's degree, can be expressed in many ways in job descriptions, e.g. B.S., BA/BS, 4-years-degree, and so on. Table 5.2 shows the words that if followed with word "degree" have the semantic value of "bachelor's degree".

To add labels to a sentence, we use regular expression over tokens. A regular expression over token transfer a patten to a Finite-State Transducer (FST), and every token of that will be transferred to an edge of FST. If we use all the expressions of a semantic value to create a pattern, the pattern will be very large, and there are too many states in the FST. For example, if we use some words in Table 5.2 to create the pattern of semantic value "bachelor's degree", the pattern will like below:

$$(\ Baccalaureate\ \mid\ bachelors\ \mid\ bachelor\ \ \mid\ B.S\ \mid\ BS\ \mid\ BA\ )\ \ degree$$

If all words in Table 5.2 are added to the pattern, the FST will have too many edges, and the matching process will be very slow because of the problem of combinatorial

Table 5.1: Example of Job Description

Senior/Principal Software Engineer
RichRelevance - San Francisco, CA

RichRelevance powers personalized shopping experiences for the worlds largest and most innovative retail brands, including Target, Sears, Marks & Spencer, John Lewis and others. Founded and led by the e-commerce expert who helped pioneer personalization at Amazon.com, RichRelevance helps retailers increase sales and customer engagement by recommending the most relevant content to consumers regardless of the channel they are shopping. RichRelevance has delivered more than $5.5 billion in attributable sales for its retail clients to date, and is accelerating these results with the introduction of a new form of digital advertising called Shopping Media which allows manufacturers to engage shoppers where it matters most – in the digital aisles on the largest retail sites in world. RichRelevance is headquartered in San Francisco, with offices in New York, Seattle, Boston, Reading and Malmho, and has been twice recognized as one of the Best Places to Work in the Bay Area.
RichRelevance is looking for a Senior/Principal Software Engineer to join our growing team!
Primary responsibilities:
- Working with large scale distributed systems
- Work with Hadoop ecosystem (technologies like Hive, Impala, HBase)
- Algorithmic development with primary focus Machine Learning
- Working with rapid and innovative development methodologies like: Kanban, Continuous Integration and Daily deployments
- Unit testing with JUnit, Performance testing and tuning

Minimum requirements:
- BS/MS in CS, Electrical Engineering or foreign equivalent plus relevant software development experience
- At least 5+ years of software development experience
- Expert in Java, Scala or any other object oriented language
- Proficient in SQL concepts (HiveQL or Postgres a plus)
- Additional language skills for scripting and rapid application development

Desired skills and experience:
- Working with large data sets in the PBs
- Familiarity with UNIX (systems skills a plus)
- Working in a distributed environment and has dealt with challenges around scaling and performance
- Mobile development for Android or iOS.

RichRelevance is an Equal Opportunity Employer and does not discriminate against any applicant on the basis of race, color, religion, national origin, gender, marital status, age, disability, sexual orientation, military/veteran status, or any other status protected by Federal or State law or local ordinance.

Table 5.2: All Words Mean Bachelors

”Baccalaureate”,”bachelors”, ”bachelor” ,”B.S.”, ”B.S”,”BS”,”BA”,”BA/BS”, ”BABS”, ”BSBA”, ”B.A.” ,”4-year”,”4-year”, ”4 year”, ”four year”,”college”,”Undergraduate” , ”University”

explosion.

To resolve this problem, we proposed an approach to use the patterns to match the *labels* of the tokens, not the the original text. In the system, we don’t care what words the sentences really use, but want to extract the semantic value of the tokens which match the pattern. The details of the approach is described below.

At first, we created two dictionaries, which are used to label the tokens. In the first dictionary, the keys are the tokens, like words in Table 5.2, and the values are the symbols for semantic values, like “BS-LEVEL” for “bachelor’s degree”, or “MS-LEVEL” for “master’s degree”. The values of the the second dictionary are the ontology hypernym of their keys, like keys “BS-LEVEL” and “MS-LEVEL” both have value “DE-LEVEL”, which means that bachelor’s degree and master’s degree are both one kind of degree level. We show the dictionaries for degree information in Table 5.3. There are also some words in the dictionaries that have the same first layer and second layer labels, which is shown in Table 5.4.

With the two dictionaries, we can label the tokens with two layers. Table 5.5 shows how the sentence “Bachelors degree in computer science or information systems.” is labeled.

The pattern “DE-LEVEL DEGREE IN MAJOR OR MAJOR ” can match the sentence above, and the output of the matching process is “BS-LEVEL” for bachelor’s degree, “MAJOR-CS” and “MAJOR-INFO” for two majors mentioned in sentence. In our system, most patterns match the labels in second layer. With this approach,

Table 5.3: Semantic Labeling

Original Text	Layer 1	Layer 2
bachelors	BS-LEVEL	DE-LEVEL
bachelor		
B.S.		
Baccalaureate		
Master	MS-LEVEL	
MS		
M.S.		
PhD	PHD-LEVEL	
Ph.D		
Doctorate		

Table 5.4: More Labels

”Be”, ”be”, ”is”, ”are”, ”am”	BE	BE
”a”, ”A”, ”an”, ”An”, ”The”, ”the”	DE	DE
”MBA”, ”BSCS”, ”BSEE”, ”MSCS” ”MSEE”, ”MSCE”,”MPH”	MAJOR-DEGREE	MAJOR-DEGREE
”practical experience”, ”work experience” ”professional experience”, ”experience	EXPERIENCE	EXPERIENCE
”preferred”, ”required”, ”desired”	PREFER-VBD	PREFER-VBD
”a plus”, ”mandatory”, ”desirable”	PREFER-JJ	PREFER-JJ
”similar”, ”related”, ”Relevant” ”equivalent”, ”based”	DEGREE-JJ	DEGREE-JJ

the size of the FST for the pattern will be minimized, so speed of matching process can be improved.

5.2 Patterns for Matching

As we explained in section B, we mentioned matching tokens in the second layer to patterns we defined. To match the labels in sentences to our patterns, we proposed a library that support matching pattern over tokens. The difference between this library and traditional regular expression is that the basic unit to be matched is token, not character. Some patterns used to match degree phrases are in Table 7.2.

Table 5.5: Labeled Sentence

layer 2	DE-LEVEL	DEGREE	IN	MAJOR	OR	MAJOR	.
layer 1	BS-LEVEL	DEGREE	IN	MAJOR-CS	OR	MAJOR-INFO	.
words	bachelors	degree	in	computer science	or	information systems	.

The patterns looks like regular expression, but they use tokens as the basic units.

Table 5.6: Patterns to Match Degree Sentences

DE-LEVEL, DE-LEVEL, OR DE-LEVEL DEGREE
DE-LEVEL DEGREE (IN | OF) DT MAJOR
MAJOR-DEGREE , MAJOR-DEGREE OR MAJOR
DE-LEVEL (, DE-LEVEL)* (OR DE-LEVEL)? BE? PERFER-VBD

5.3 Pattern Matching Library

In section C we introduced how we use the library of pattern matching over tokens to match the sentences. In this section we will introduce more details of this library, including its advantages and implementation details.

5.3.1 Finite-State Transducer

Finite-State Transducers [39] have been used as a tool to match patterns and extract information for more than 20 years. This approach has been demonstrated to be very effective in extracting information from text like CIRCUS [25] and FASTUS [19]. In the widely used NLP toolkit GATE [9], the semantic tagger JAPE (Java Annotations Pattern Engine) could describe patterns that are used to match and annotate tokens. JAPE adopts a version of CPSL (Common Pattern Specification Language) [4], which provides FST over annotations. Chang et al. presented cascaded regular expressions over tokens [8], which proposed a cascaded pattern matching tool over token sequences.

After studying these tools, we found most of them to be powerful and complex, but not very flexible. One reason is that developers need to learn some Domain specific Languages (DSLs) like CPSL. The other reason is the extra effort and time required to integrate the pattern matching tool into the system. So here we proposed a more flexible and lightweight FST framework, which can do regular expression matching over labeled tokens. We give the definition of Finite-State Transducer here. A Finite-State Transducer is a 6-tuple $(\Sigma_1, \Sigma_2, Q, i, F, E)$ where:

- Σ_1 is a finite alphabet, called the input alphabet.
- Σ_2 is a finite alphabet, called the output alphabet.
- Q is a finite set of states.
- $i \in Q$ is the initial state.
- $F \subset Q$ is the set of final states.
- $E \subset Q \times \Sigma_1^* \times \Sigma_2^* \times Q$ is the set of edges.

For example, the FST $T_{d3} = (\{0,1\}, \{0,1\}, \{0,1,2\}, E_{d3})$ where $E_{d3} = \{$ (0, 0, 0, 0), (0, 1, 0, 1), (1, 0, 0, 2), (1, 1, 1, 0), (2, 1, 1, 2), (2, 0, 1, 1) $\}$ is shown in Figure 5.1.

Regular expressions can be converted to automata [2], and FST is also an automata. To convert a regular expression over token to a FST we need two steps: The first is parsing the expression to a tree of matchers, the second is transfer the tree of matchers to the FST. We will introduce these two steps in next.

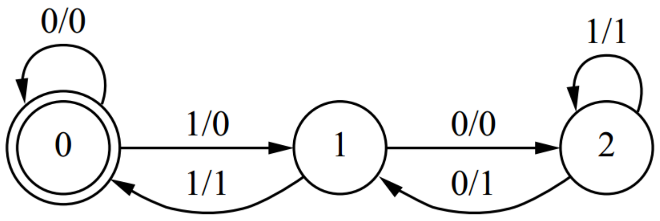


Figure 5.1: Zero or One NFA

5.3.2 Matchers in the Pattern Matching Library

In our library, a "matcher" could be a token to be matched, or a composition of other matchers. Our library supports syntax used in traditional regular expressions over strings. We list the syntax that the library supports in Table 5.7. The first column is the names of the matchers, the second column is the explanation of the function of the matchers, and third column is the their counterpart syntaxes of traditional regular expression. The RegexMatcher in our library is constructed with a regular expression, and the matcher matches any string that matches the regular expression in the matcher. We give examples of the syntax of these matchers in Table 5.8.

Table 5.7: Matchers of Our Library

Matcher Name	Function	Counter Part of regex
UnitMatcher	token is matches the it	character in regex
SequenceMatcher	A list of Matcher	sequence of characters
QuestionMatcher	One or more of the preceding token	?
StarMatcher	Zero or more of the preceding token	*
PlusMatcher	Zero or one of the preceding token	+
DotMatcher	Any token	.
RegexMatcher	Any token matches the regular expression	N/A

Table 5.8: Examples of Matcher

Matcher Name	Example
UnitMatcher	DEGREE
SequenceMatcher	DE-LEVEL DEGREE
QuestionMatcher	DE-LEVEL (OR DE-LEVEL)? DEGREE
StarMatcher	DE-LEVEL (, DE-LEVEL)* DEGREE
PlusMatcher	DEGREE IN MAJOR +
DotMatcher	HAS . DEGREE
RegexMatcher	r"d-d" years

The framework supports three styles of creating patterns: regular expression style, operator style and object 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.

The pattern is: DE-LEVEL DEGREE (IN | OF) DT? MAJOR

The code is:

```
seqMatcher =parser.parse("DE-LEVEL DEGREE ( IN | OF ) DT? MAJOR")
```

The second style is using algebraic operators to connect matchers, which can help developer reuse previous patterns when the new patterns include old ones. It is shown in follows:

The pattern is: "DE-LEVEL DEGREE (IN | OF) MAJOR"

The code is:

```
seqMatcher = UnitMatcher("DE-LEVEL") + UnitMatcher("DEGREE") +
( UnitMatcher("IN") | UnitMatcher("OF" ) ) + UnitMatcher("MAJOR")
```

We also could create a complex matcher in object-oriented programming style.

The pattern is: ”DE-LEVEL DEGREE (IN | OF) MAJOR”

The code is:

```
matcher1 = UnitMatcher("DE-LEVEL")
matcher2 = UnitMatcher("DEGREE")
matcher3 = UnitMatcher("IN")
matcher4 = UnitMatcher("OF")
matcher5 = UnitMatcher("MAJOR")
matcher6 = AlternateMatcher([matcher3,matcher4])
seqMatcher = SeqMatcher([matcher1, matcher2, matcher6, matcher5])
```

The flexibility of the tool also comes from the fact that developers can determine which layer of the array should be matched, the original text, or labels in the first layer or labels in the second layer. Developers can assign a lambda expression to the matcher's catching function, which defines how to get the matching input strings from the sentences, as well as an out function, which defines what should be outputed. For example, the labeled sentence is a sequence of arrays, each array includes the original text token and its labels in the other two layers, which is shown in table 5.5. To match the labeled sentence, we set the lambda expression for catching function to “lambda x:x[2]”, and the out function to “lambda x:x[1]”, which make the matcher match the label in second layer, and output the the value of semantic value in the first layer.

5.3.3 Implementation of the Pattern Matching Library

We use PLY(Python Lex-Yacc) as the grammar parser, which is a pure-Python implementation of the popular compiler construction tools lex and yacc. We defined syntaxes of regular expression over tokens in the parser, which can parse the token regular expression to the tree structure of matchers.

We use the algorithm proposed by Thompson and Ken [45] to construct the FST from the tree of matchers, which is shown in Algorithm 1. The algorithm accesses the inner structure of a matcher recursively, and create state for the matcher. Each state is a partial Nondeterministic Finite Automaton (NFA), which has one or more dangling arrows. The algorithm builds the whole NFA by connecting the arrows of the partial NFAs. Different operator will have different structures of NFA, which is shown below:

The NFAs for matching single token is shown in Figure 5.2.

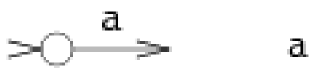


Figure 5.2: Single Token NFA

The NFA for the concatenation e_1e_2 connects the final arrow of the e_1 machine to the start of the e_2 machine, as shown in Figure 5.3.

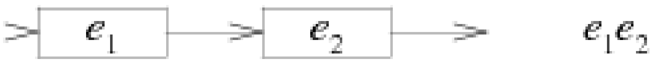


Figure 5.3: Concatenation NFA

The NFA for the alternation $e_1 \mid e_2$ adds a new start state with a choice of either the e_1 machine or the e_2 machine, which is shown in Figure 5.4.

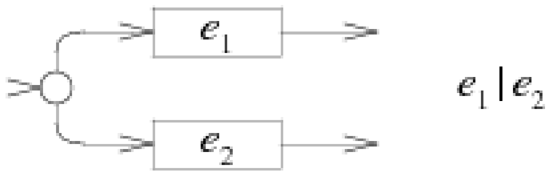


Figure 5.4: Alternation NFA

The NFA for e? alternates the e machine with an empty path, which is shown in Figure 5.5.

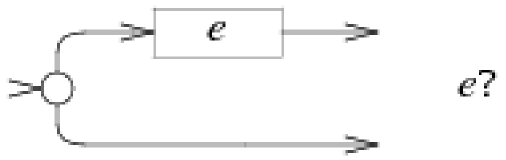


Figure 5.5: Zero or One NFA

The NFA for e* uses the same alternation but loops a matching e machine back to the start, which is shown in Figure 5.6.

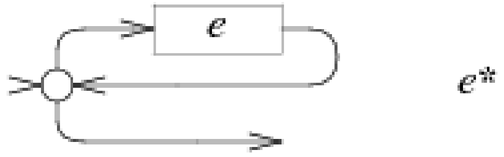


Figure 5.6: Zero or More NFA

The NFA for e+ also creates a loop, but one that requires passing through e at least once, which is shown in Figure 5.7.

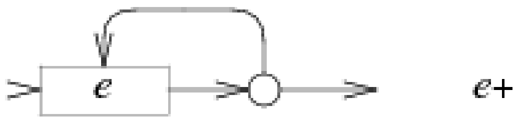


Figure 5.7: One or More NFA

With the above rules, we can convert the expression "DL (, DL)* (or DL)? DEGREE" to an FST in Figure 5.8.

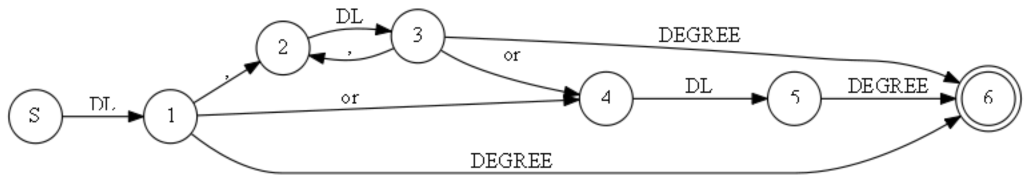


Figure 5.8: Finite Automata Transducers

Comparing our method to other state of art machine learning-based information extraction algorithms, our method has such advantages:

1. Easy to implement. According to the pseudo code in 1, even some undergraduate students without strong machine learning background can implement our algorithm in a few days.

2. There is no need for data labeling. Supervised machine learning algorithms always need a large labeled training data set, but data labeling is tedious work

```
Input: $matcher, nfa$
Output: $NFA$
switch $matcher$ do
    case $UnitMatcher$
        state = compileToNFA(matcher]) ;
        connect( nfa.last,  [state] ) ;
        nfa.last = [state] ;
        break;
    case $SequenceMatcher$
        for $(i = 0; i < len(matcher.list); i++)$ do
            state = compileToNFA(matcher.list[i]) ;
            connect( nfa.last,  [state] ) ;
            nfa.last = [state] ;
        end
        break;
    case $AlternateMatcher$
        state1 = compileToNFA(matcher.list[0]) ;
        state2 = compileToNFA(matcher.list[1]) ;
        connect( nfa.last,  [state1,  state2] ) ;
        nfa.last = [state1,   state2] ;
        break;
    case $QuestionMatcher$
        state1 = compileToNFA(matcher.list[0]) ;
        connect( nfa.last,  [state1,  nfa.last] ) ;
        nfa.last = [state1,   nfa.last] ;
        break;
    case $PlusMatcher$
        state1 = compileToNFA(matcher.list[0]) ;
        connect( state1,  state1 ) ;
        connect( nfa.last,  state1 ) ;
        nfa.last = [state1 ] ;
        break;
    case $StarMatcher$
        state1 = compileToNFA(matcher.list[0]) ;
        connect( state1,  state1 ) ;
        connect( nfa.last,  state1 ) ;
        nfa.last = [state1,  nfa.last] ;
        break;
endsw
```

Algorithm 1: CompileToNFA

for users. With our pattern matching algorithm, we avoid the work manual data labeling.

3. The accuracy of information extraction can increase monotonously as the number of patterns increase. So with enough patterns, the accuracy becomes quite high.

4. High speed for labeling data. The time complexity of pattern matching is O(n) [45], which is smaller than some complex machine learning based approaches. One example is Conditional Random Fields(CRFs), which uses Viterbi algorithm [47] to label the sequence, the time complexity of it is $O(n^2t)$.

In this chapter we have introduced how we extracted the information from the resumes and job descriptions, and the implementation details of the pattern matching library, regular expression over tokens. We can get the models of resumes and job descriptions through the procedure described in this chapter. In the next chapter, we will discuss how our system searches and ranks job models by resume models.

6. MODEL SIMILARITY

The similarity value between a job model and a résumé model is the summation of weighted similarity values of different fields. The equation is given below:

$$sim(r, j) = \sum_{i=1}^{n} simfun_i(r_i, j_i) \times w_i$$

The value of $sim(r, j)$ is the summation of similarity values of different fields times their corresponding weights. $simfun_i(r_i, j_i)$ is the similarity function of the ith field of the model. In our system, the résumé model and job model both have four fields: job title, major, academic degree and skills. The similarity value between a résumé model and job model is the sum of the productions of similarity values of all the fields pairs and their weights. We will introduce how to calculate the similarity value for each field in this chapter.

6.1 Similarity of Major and Academic Degree

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 in the job model. If it is, the similarity value is 0.5; otherwise the similarity value is 0. The equation is shown below:

$$MajorSim(r, j) = \left\{ \begin{array}{lc} 1, & r_{major} = j_{major} \\ 0.5, & r_{major} \in related(j_{major}) \\ 0, & otherwise \end{array} \right\}$$

There are five kinds of academic degrees in the system: high school, associate, bachelor, master, and Ph.D., which are mapped to the integer values form 1 to 5. If the degree value in the résumé model is less than that in the job model, which means that the job seeker's education background cannot satisfy the requirement of the job, the similarity value in this case is 0. If the degree value in the résumé model is equal to the job model and no more than 2 above, the similarity value is 1. In some cases, the degree value in the résumé model is greater than that of the job model, and the difference is greater than 2, which means that the job seeker's degree is much higher than the requirement of the job. The situation is also a kind of relative matching, so the similarity value here is 0.5. The equation is shown below:

$$DegreeSim(r, j) = \left\{ \begin{array}{ll} 0, & r_{degree} < j_{degree} \\ 1, & 0 < r_{degree} - j_{degree} \leqslant 2 \\ 0.5, & r_{degree} - j_{degree} > 2 \end{array} \right\}$$

6.2 Similarity of Job Title

Another field of needs similarity calculation is the job titles in the models. A job title can be parsed into some sub fields: job role, level, platform, programming language. The value of job roles includes: developer, manager, administrator and so on. There are levels values in such roles: such as junior, senior and architect. The platforms: web, mobile and cloud are used by the very roles. The similarity value between two titles is the sum of all the similarity values of these fields. The similarity value of each sub filed ranges from 0 to 1, and we also normalized the similarity summation value to 1 by dividing the number of sub fields. If the job seeker has some working experience, there may be some job titles in their résumé. When calculating the similarity value between a résumé model and a job model, the system calculates the similarity values of the title of job model to all the titles in the résumé model and returns the maximum one.

6.3 Similarity of Skills

The job model usually has requirements of some skills, and the résumé model lists the skills the job seeker has as well. The similarity value of skills field is the normalized summation of all the similarity values of skills in the job model.

$$SkillSetSim(SJ, SR) = \frac{\sum_{sj_i \in SJ} SkillSim(sj_i, SR)}{|SJ|}$$

For every skill in the job model, the similarity value is the maximum value it can get from the skills in the résumé model. The equation is shown below:

$$SkillSim(sj_i, SR) = \left\{\begin{matrix} 1, & sj_i \in SR \\ max(sim(sj_i, rj_k)), & sj_i \notin SR \end{matrix}\right\}$$

In the equation, sj_i is the ith skill in the job model, and SR is the skill set of the résumé model. If there is the skill sj_i in the skill set SR, the similarity value for sj_i is 1, otherwise the system chooses the maximum similarity value from all the similarity values between skill sj_i and the skills in the the résumé model.

We introduced how to calculate similarity values for three fields in résumé and job models. In the next chapter, we will introduce how to use a domain specific ontology to calculate similarity values between the skills fields of the two models.

7. ONTOLOGY CONSTRUCTION AND SIMILARITY

7.1 Semantic Similarity in JRSs

After getting job models by the information extraction module, users can search for jobs in the system. In previous studies of JRSs, ontology is used as a knowledge base to store knowledge and rules, which could help compare the similarity between different concepts. Liu and Dew [27] used Resource Description Framework (RDF) to represent and store the expertise of experts, and they used a RDF-based expertise matcher to retrieve the experts whose expertise included the required concept.

Proactive [24] used two kinds of ontology, job category and company information. The system used an ontology checker to classify the job information, stored the domain knowledge and calculated the weight value in recommendations.

Fazel [15] used a hybrid approach to match job seekers and job postings, which takes advantage of the benefits of both logic-based and ontology-based matching. In his paper the description logics (DL) are used to represent the candidate and job opening, and the ontology is used to organize the skills in a taxonomy. The paper provides an equation to calculate the matching degree:

$$sim\,(P, j) = \sum x_{ij} \times u(ds_i)$$

where x_{ji} is the Boolean variable indicating whether desire i is satisfied by applicant A_j in the set of all qualified applications.

Kumaran et al. [21] also used an ontology to calculate the similarity between the job criteria and candidates' résumé in their system [21]. The similarity equation they used is:

$$M\left(i_1, i_2\right) = \frac{\sum_{k=1}^{n} Sim\left(p_k^{i1}, p_k^{i2}\right) * W_k^{i2}}{\sum_{k=1}^{n} W_k^{i2}}$$

The similarity function $Sim(p_1, p_2)$ is defined as follows:

$$Sim(p1, p2) = \begin{Bmatrix} 1, & if\ similarity\ of\ p1\ and\ p2 \geqslant t \\ 0, & otherwise \end{Bmatrix}$$

7.2 Ontology Construction

Before calculating the similarity between concepts, we need to construct the ontology first. Semantic web has been a popular research topic in previous years, and at the same time thousands of domain ontologies have been created [11]. A paradigmatic example is WordNet [16], which is a general purpose thesaurus, and contains more than 100,000 general English concepts. ACM has created a poly-hierarchical ontology that can be utilized in semantic web applications [1], but it is mostly used in academic areas. DBpedia [6] provides structured information from Wikipedia and make this information available on the Web, but its coverage is huge, and most of them is not related to job finding. Currently, there is no domain specific technology ontology built for recruiting purpose.

The domain specific technology ontology for recruiting should include a lot of technical terms, like programming language, programming library, commercial products and so on. Furthermore, there are new techniques invented everyday, so new IT terms will appear continuously. Ding et al. [12] gave a survey of current ontology generation approaches such as manual, semi-automatic, and automatic. Some aspects of the approaches were discussed in the paper, like the source data, concept extraction methods, ontology representation, and construction tools. Inspired by this paper, we propose a semi-automatic approach to construct the IT skill ontology, which use a pattern matching approach to collect possible technical terms, and use DBpedia to verify the them.

From the observation, we found that sentences with skill requirements in job descriptions always list several skills in the sentence, which is shown in Table 7.1. Based on this character, we propose a bootstrap approach to collect IT terms in job descriptions. First, we manually collect about fifty terms from job descriptions, and add them to the term list. Then we use our pattern match library to find the sentences that matching the pattern in Table 7.2 from a set of job descriptions. An example of a sentence which matches the pattern is shown in Table 7.3. We extract the tokens which match the star symbol from the sentences; these tokens have high probability to be technical terms. Then we could check the tokens in Dbpedia to see whether they are under the categories like software, programming language or any other technical related ones. If they are, we could classify them as terms, and add them to the terms list. After scanning all the sentences in the job description set, the term list will be larger, and we can use the larger term list to start a new iteration of scaning. This process stops when the number of found new terms is below a threshold. The process is shown Figure 7.1.

Table 7.1: Example Sentences in Job Descriptions

1. A high-level language such as Java, Groovy, 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

Table 7.2: Patterns to Extract Terms

term , * , *, term
term , * , *, and term

Table 7.3: An Example Sentence Matches the Pattern

Experience	with	TERM	,	*	,	*	,	TERM	,	and	*
Experience	with	AJAX	,	XML	,	XSL	,	XSLT	,	and	CSS

For example, we extract the token ”XSL”, which currently is not in the terms list. We check the word on DBpedia by accessing the URL:http://dbpedia.org/page/XSL. If we can get the XML formatted description of XSL, and any element in “dcterms:subject” section has the value which is a technical category, like “Programming languages”, “Markup languages” and so on, we can indicate that the word is a technical term, and add it to the term list.

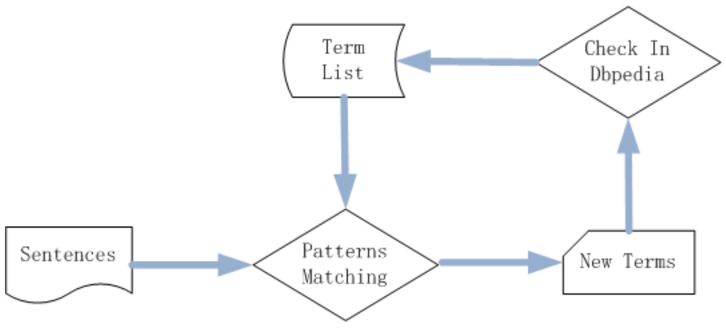


Figure 7.1: Procedure of Finding Technical Terms

But not all the extracted terms can be verified in DBpedia, because some terms have multiple meanings in English, and the URLs of their DBpedia pages are unpredictable. For example, the word “Python” could be an animal name or a programming language. The meaning of the programming language has the DBpedia URL http://dbpedia.org/page/Python_(programming_language), which is difficult to predict. In this case, we have to check the term manually. After getting all terms, we use Protege [32], an open source ontology editor, to edit the domain specific ontology, and saved it in RDF format. The interface of Protege is shown in Figure 7.2. Part of the technical ontology is shown in Figure 7.3.

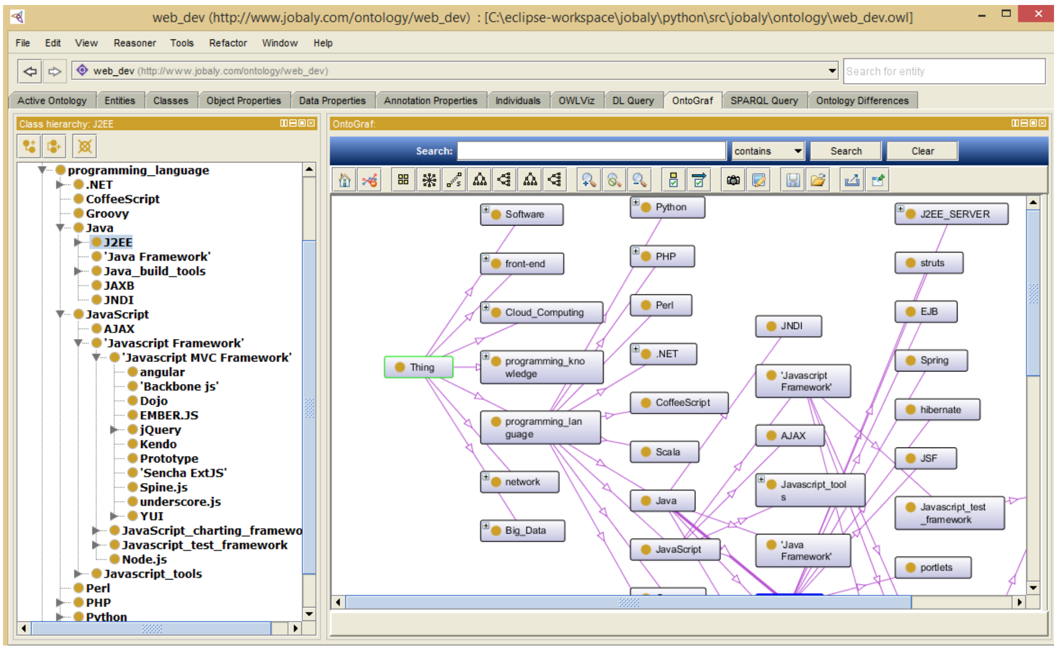


Figure 7.2: Interface of Protege

7.3 Ontology-Based Semantic Similarity

Sánchez et al. [41] summarized ontology-based similarity assessment into three kinds and gave both advantages and disadvantages of each approach. The three kinds

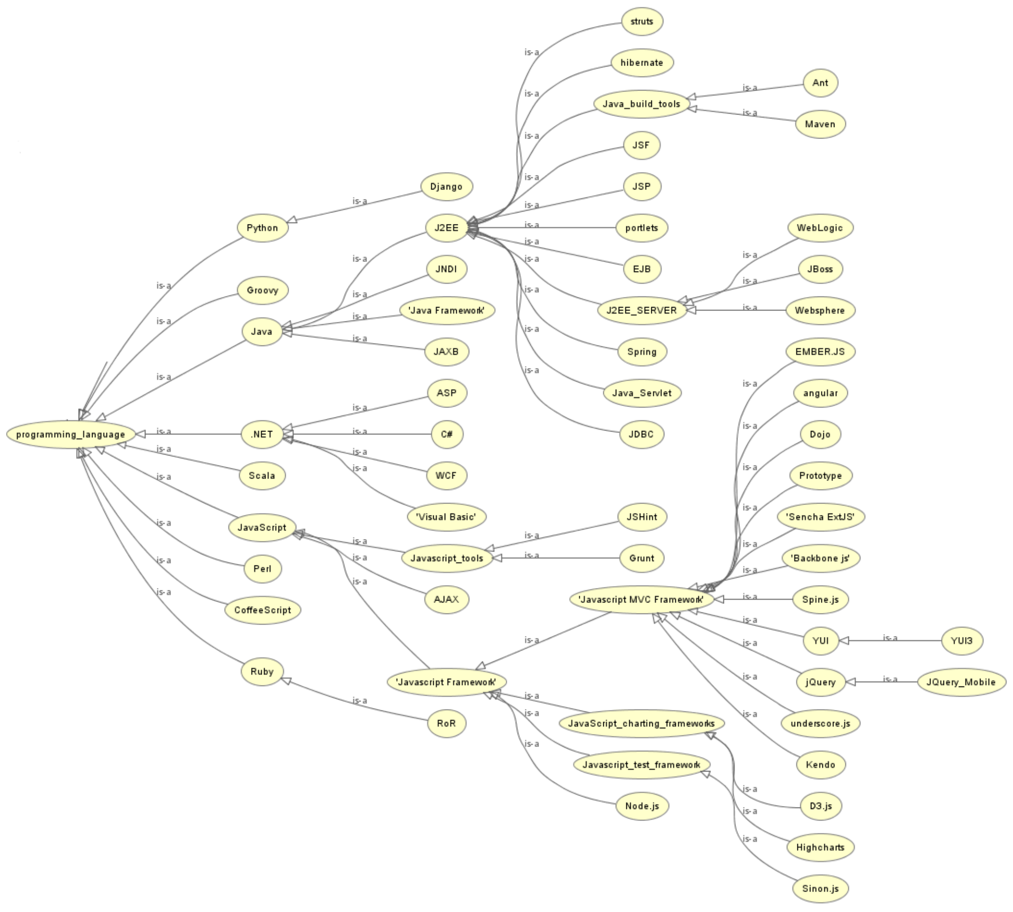


Figure 7.3: Part of Ontology

of categories are: Edge-counting approaches, Feature-based measures, and Measures based on Information Content.

7.3.1 Path-Based Approaches

In path-based approaches, the ontology is viewed as a directed graph, in which the nodes are the concepts, and the edges are taxonomic relation (e.g. is-a). Rada, et al. [34] measure the similarity by the distance of two nodes in the graph. Therefore, the semantic distance of two concepts a and b will be:

$$dis_{rad}(a,b) = min|path_i(a,b)|$$

Wu and Palmer [49] realized that the depth in the taxonomy will impact the similarity measure of two nodes, because the deeper of the nodes are in the tree, the semantic distance is smaller. Therefore they gave a new measure of ontology:

$$sim_{w\&p}(a,b) = \frac{2 \times N_3}{N_1 + N_2 + 2 \times N_3}$$

N_1 and N_2 is the numbers of is-a links from each term to their Least Common Subsumer(LCS), N_3 is the number of is-a links of the LCS to the root of the ontology.

Based on the same idea, Leacock and Chodorow [23] also proposed a similarity measure that combined distance Np between terms a and b and the depth D of the taxonomy.

$$sim_{l\&c}(a,b) = -\log(Np/2D)$$

There are some limitations of path-based approaches. First, it only considers the shortest path between concept pairs. When they meet a complex situation like multiple taxonomical inheritance, the accuracy of them will be low. Another problem of the path-based approaches is that they assume that all links in the taxonomy have uniform distance.

7.3.2 Feature-Based Measures

Feature based approaches assess the similarity between concepts as a function of their properties. They consider the degree of overlapping between sets of ontological features, like Tversky's model [46], which subtracts the non-common features from common features of two concepts.

$$sim_{tve}(a,b) = \alpha \cdot F\left(\Psi(a) \cap \Psi(b)\right) - \beta \cdot F\left(\Psi(a) \setminus \Psi(b)\right) - \gamma \cdot F\left(\Psi(b) \setminus \Psi(a)\right)$$

Where F is salience of a set features, and α, β and γ are weights of the contribution of each component.

Rodríguez and Egenhofer [40] computed similarity by summing the weighted sum of similarities between synsets, features, and neighbour concepts.

$$sim_{re}(a,b) = w \cdot S_{synsets}(a,b) + u \cdot S_{features}(a,b) + v \cdot S_{neighborhoods}(a,b)$$

The feature-based methods consider more semantic knowledge than path-based methods. But only big ontologies/thesauri like Wordnet [31] have this kind of information. Ding et al. [11] revealed that domain ontologies very occasionally model any semantic feature apart from taxonomical relationship.

7.3.3 Content-Based Measures

Other approaches want to overcome the limitations of edge-counting approaches are Content-based measures. Resnik [36] proposed a similarity measure, which depends on the amount of shared information between two terms:

$$sim_{res}(a,b) = IC(LCS(a,b))$$

LCS is the Least Common Subsumer of terms in a ontology, and IC is Information Content, which is the negative log of its probability of occurrence, $p(a)$. Lin [26] and Jiang and Conrath [20] extended Resnik's work. They also considered the IC 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.

$$sim_{lin}(a,b) = \frac{2 \times sim_{res}(a,b)}{(IC(a) + IC(b))}$$

The are also two disadvantages of the content-based measures. First, the approaches cannot compute the concepts of leave nodes, because they don't have subsumers. Second, if the concepts do not have enough common subsumers, their similarities are hard to be calculated.

7.4 Statistical-Based Ontology Similarity Measure

In this thesis, we proposed a new statistical-based ontology similarity measure. In most job descriptions, they list many skills the positions required. From observation, we found that related skills always exist in the job description simultaneously, and the positions of them are always close, e.g. HTML and CSS are always required together, and appear in the same sentence. We could see this phenomenon in Table 7.4, which include some skill requirement sentences from some job descriptions.

We can see from the Table 7.4, the closely related concepts are always have short distance. Based on such observation, we give a new statistical-based ontology

Table 7.4: Some sentences of Job Descriptions

1. A high-level language such as Java, Groovy, 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

similarity measure. If two concepts a and b have the same direct hypernym or one is the hypernym of the other, the similarity between them is given:

$$S(a, b) = \frac{N_{a \cap b}/N_{a \cup b}}{avg(\log_2(mindis(d_i, a, b) + 1))}$$

The numerator is the ratio of the number of documents in which the two terms exist together ($N_{a \cap b}$) and the number of documents have a least one of them ($N_{a \cup b}$). The denominator is the average log value of minimum distance $mindis(doc, a, b)$ of the two terms in documents that have them both.

We set the restriction on the position of the two concepts in the ontology, because the position of the concepts in the ontology are based on their technical similarity to others. Similar techniques will be assigned into the same category, so they should share the same hypernym, and one could be an alternative to the other. For example, we put EJB and Hibernate in the same category, because they are both J2EE persistence layer technologies, and both have the O/R mapping concept. If the applicant is familiar one of them, they can master the other very quickly. Another example is Grail and Django, they are both web frameworks and share same web design philosophies, but one of them is designed for Java web application and the other is created for Python web application. If a developer has some some experience with one of them, he/she still need to spend a lot of time to learn the other to overcome the gap between programming languages. The algorithm to calculate the similarity of two concepts is in Algorithm 2.

```
**Input**: $Docs\ term1,\ term2$
**Output**: $similarity$
$total = 0;\ hastwo = 0;\ dislist = [\ ];$
for $i = 1;\ i \le len(Docs);\ i++$ do
    if $Docs_i$ has at least one term then
        $total\ += 1$ ;
        if $Docs_i$ has both terms then
            $hastwo\ += 1$ ;
            $mindis = minimium\_distance\ (Docs_i, term1, term2)$ ;
            $dislist.\ add\ (log_2(mindis + 1))$ ;
        end
    end
end
$factor1 = hastwo\ /\ total$ ;
$factor2 = avg(dislist)$ ;
return $factor1\ /\ factor2$;
```

Algorithm 2: Calculating Statistical-based Similarity

The matrix in Table 7.5 show the similarity values among of some skills, which is gotten from 500 job descriptions. For example the skill HTML, the most relevant skills in order are CSS, Javascript, and jQuery, which is the same from the perspective of experienced developers. The other example is Java, the most relevant skill in the matrix is JSP, which is also agree with the general technical knowledge.

Table 7.5: Similarities of Skills List 1

Term	Java	JDBC	Spring	Hibernate	MySql	Oracle
Java	1	0.0523	0.091	0.0458	0.0339	0.0608
JDBC	0.0523	1	0.0525	0.0799	0.006	0.0616
Spring	0.091	0.0525	1	0.2008	0.0194	0.0878
Hibernate	0.0458	0.0799	0.2008	1	0.0073	0.115
MySql	0.0339	0.006	0.0194	0.0073	1	0.049
Oracle	0.0608	0.0616	0.0878	0.115	0.049	1

Table 7.6: Similarities of Skills List 2

Term	Javascript	jQuery	HTML	CSS	Java	Python	Ruby	JSP
Javascript	1	0.1981	0.2087	0.2439	0.0665	0.0189	0.023	0.0253
jQuery	0.1981	1	0.0979	0.1328	0.0439	0.0142	0.0266	0.0232
HTML	0.2087	0.0979	1	0.3569	0.0473	0.0175	0.023	0.0103
CSS	0.2439	0.1328	0.3569	1	0.0537	0.0153	0.0181	0.015
Java	0.0665	0.0439	0.0473	0.0537	1	0.0498	0.0287	0.075
Python	0.0189	0.0142	0.0175	0.0153	0.0498	1	0.1333	0.0025
Ruby	0.023	0.0266	0.023	0.0181	0.0287	0.1333	1	0.012
JSP	0.0253	0.0232	0.0103	0.015	0.075	0.0025	0.012	1

8. EVALUATION

In this section we will evaluate the performance of the key components of the system, the Information Extraction module, the Ontology Similarity module, and the overall system resulting from the combination of the components.

8.1 Experiments of Information Extraction

To evaluate the performance of the information extraction module, we extract sentence types through the use of sentence filters. To explain the process of our experiment, we use the sentences whose content pertains to the applicant's college degree information.

In the experiment, we selected 100 sentences from existing job descriptions, and the content of these sentences were requirements of candidate degree and major. We labeled the values for "degree" and "major" manually. We use some content patterns that we can identify from these sentences to match and extract the degree information. When we used 6 patterns, the accuracy of "degree" became 94%. We also compared our pattern matching method to the conditional random fields (CRFs) model [22], which is a state of art machine learning model for sequence labeling. We used 200 labelled sentences to train the CRF model, and the features of the CRF model are words in the sentences and part of speech tags of the words. The accuracies of information extraction of the three fields with our two methods, pattern matching, and the application of the CRF model are shown in Table 8.1.

8.2 Experiments of Ontology Similarity

We selected some common skills from 500 job descriptions; table 8.2 shows similarity values between these skills. Higher values correspond to greater similarities, so

Table 8.1: Information Extraction

Field	Pattern Num	Accuracy of Pattern Matching	Accuracy of CRF
Degree	6	0.94	0.85
Major	10	0.85	0.72

Our pattern matching approach can get higher accuracy.

Table 8.2: Similarities of Skill List 1

Term	Java	JDBC	Spring	Hibernate	MySql	Oracle
Java	1	0.0523	0.091	0.0458	0.0339	0.0608
JDBC	0.0523	1	0.0525	0.0799	0.006	0.0616
Spring	0.091	0.0525	1	0.2008	0.0194	0.0878
Hibernate	0.0458	0.0799	0.2008	1	0.0073	0.115
MySql	0.0339	0.006	0.0194	0.0073	1	0.049
Oracle	0.0608	0.0616	0.0878	0.115	0.049	1

the similarity between one skill and itself is 1. We selected one concept and ranked the other concepts by their similarity values to this concept. Human judges helped rank these concepts by assigning them ”relevance scores” so that we can use NDCG to evaluate the effectiveness of our approach.

We use the $Normalized\ Discounted\ Cumulative\ Gain(NDCG)$ [30] to evaluate the statistical-based similarity. NDCG is an important measure to evaluate the ranked retrieval results, which is the ratio of Discounted Cumulative Gain (DCG) to Ideal Discounted Cumulative Gain (IDCG).

$$NDCG = \frac{DCG}{IDCG}$$

DCG is the measure of how documents are ranked according to their relevance scores, and IDCG is the DCG value that the documents are strictly sorted by their

Table 8.3: Javascript Similarity Evaluation

Term	Similarity Value	Position	Relevance
jQuery	0.1981	4	8
HTML	0.2087	3	4
CSS	0.2439	2	3
Java	0.0665	5	1
Python	0.0189	8	1
Ruby	0.023	7	1
JSP	0.0253	6	2

Table 8.4: HTML Similarity Evaluation

Term	Similarity Value	Position	Relevance
Javascript	0.2087	2	3
jQuery	0.0979	3	3
CSS	0.3569	1	5
Java	0.0473	4	1
Python	0.0175	6	1
Ruby	0.023	5	1
JSP	0.0103	7	3

relevance values.

$$DCG = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

Table 8.3 shows how we evaluate the similarity between the concept "Javascript" and other concepts. The first column is a skill name, the second column is its similarity value to "Javascript", the third column is its position ranked by the similarity value, and the fourth column is its relevance value given by the judges. The NDCG value for concept Javascript is 0.94, and in Table 8.4, NDCG value for concept HTML is 0.97.

8.3 Evaluation of the System

In a traditional information retrieval systems, precision and NDCG are widely used measures [30]. Precision (P) is the fraction of retrieved documents that are relevant.

$$Precision = \frac{\#(releveant\ items\ retrieved)}{\#(retrieved\ items)}$$

We first used Precision@K to compare the performance of our approach to the classical information retrieval models: Okapi BM25 [37], Kullback-Leibler divergence, and the TF-IDF. Precision@K is the proportion of relevant documents in the first K positions and is given below:

$$P@k = \frac{1}{k}\sum_{i=1}^{m} l_i 1\left(r(i) \leq k\right)$$

Where 1 is the indicator function: $1(A) = 1$ if A is true, 0 otherwise.

To evaluate a job finding, we compare the results of the system with three classical information retrieval models: Kullback-Leibler divergence [52], TF-IDF [30] and Okapi BM25 [38]. We give the definition of these measures below.

Kullback-Leibler divergence is a non-symmetric measure of the difference between two probability distributions P and Q. The score of a document D with respect to query Q is given by:

$$\begin{aligned} s(D,Q) &= -D(\theta_Q \parallel \theta_D) \\ &= -\textstyle\sum_{\omega \in V} p(\omega \mid \theta_Q) \log \frac{p(\omega|\theta_Q)}{p(\omega|\theta_D)} \\ &= \textstyle\sum_{\omega \in V} p(\omega \mid \theta_Q) \log p(\omega \mid \theta_D) - \sum_{\omega \in V} p(\omega \mid \theta_Q) \log p(\omega \mid \theta_Q) \end{aligned} \tag{8.1}$$

In the equation θ_Q is the language model for a query, and θ_D is the language model for a document.

TF-IDF is a Vector Space Model, which calculates the Cosine Similarity between the vectors of the query q and the document d. In the equation, tf is the term frequency, and *idf* is the inverse document frequency. The TF-IDF weighting scheme assigned to term t a weight in document d given by:

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t$$

The similarity between the query q and the document d given by:

$$score(q,d) = \sum_{t \in q} tf\text{-}idf_{t,d}$$

Okapi BM25 is a bag-of-words retrieval model that ranks a set of documents based on the query terms appearing in each document. Given a query q, the BM25 score of a document d is:

$$score(q,d) = \sum_{t \in q} idf_t \frac{tf_{t,d} \cdot (k_1 + 1)}{tf_{t,d} + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

In the formula, idf_t is IDF value of term t, tf is the term frequency; $|D|$ is the length of the document D; *avgdl* is the average length of all the documents, and k_1 and b are the free parameters.

In the evaluation phase, we created a data set of 100 job descriptions that includes several kinds of jobs such as web developers, server back-end developers, mobile developers and so on. We used 5 candidate résumés and retrieved the top 20 jobs. The relevance value of the job descriptions to each résumé were set manually by ten human judge. We created a query q from the résumé, treated the text of the job descriptions as documents d, and applied standard ad-hoc retrieval techniques to rank the jobs. We intended to return jobs that better matched the candidates' résumés at the top. The result of Precision@k is in table 8.5.

Table 8.5: Precision of Job Ranking

k	Okapi BM25	KL	TF-IDF	RésuMatcher
5	0.66	0.27	0.72	0.82
10	0.46	0.27	0.50	0.76
20	0.33	0.21	0.35	0.77

RésuMatcher get the highest precision.

The other measure we used is NDCG, which is explained in last section. Table 8.6 shows the NDCG value get from different information retrieval models. The result shows that Ontology Similarity performs the best.

Table 8.6: NDCG of Job Ranking

k	Okapi BM25	KL	TF-IDF	RésuMatcher
5	0.15	0.34	0.45	0.78
10	0.18	0.44	0.47	0.72
20	0.19	0.35	0.45	0.66

Our sytem get the highest NDCG value.

8.4 Comparing with the Keyword Searching

We also made experiments to compare the quality of search results quality between our system and the keyword searching approach. In one experiment, we first selected a résumé, and picked a related keyword e.g. “Java” to search jobs in Indeed.com. We used this process to simulate how a job seeker searches jobs on Indeed.com. The top 100 jobs returned by Indeed.com were saved in our system, then we used the résumé to match the 100 jobs. We compared the quality of the approaches by calculating the measures Precision@K and DCG. We use DCG not NDCG because we compare the two approaches on the same data set, and the IDCG is the same. Five judges gave the relevance values to the résumés and jobs, and we used average values. Table 8.7 shows comparison between the two approaches by using keyword "Java" and the résumé of a Java developer. The comparison for keyword "Python" and the résumé of a Python developer is shown in Table 8.8, and the average comparison for five keywords and five résumé is shown in Table 8.9. We can see from the result that our system can get much better results than keyword searching approach.

Table 8.7: Comparison of the Two Approaches - Java Developer

k	Precision@K		DCG	
	Indeed	RésuMatcher	Indeed	RésuMatcher
5	0.73	1.00	10.86	27.80
10	0.65	0.95	24.85	47.06
20	0.72	0.83	51.97	73.33

RésuMatcher can get better result.

Table 8.8: Comparison of the Two Approaches - Python Developer

k	Precision@K		DCG	
	Indeed	RésuMatcher	Indeed	RésuMatcher
5	0.45	0.63	11.27	11.98
10	0.32	0.62	13.43	15.79
20	0.22	0.42	16.44	20.21

RésuMatcher can get better result.

Table 8.9: Comparison of the Two Approaches - Average

k	Precision@K		DCG	
	Indeed	RésuMatcher	Indeed	RésuMatcher
5	0.84	0.87	23.87	32.97
10	0.72	0.86	37.02	45.57
20	0.65	0.76	58.70	66.70

RésuMatcher can get better result.

9. CONCLUSION AND FUTURE WORK

9.1 Conclusion

In this thesis we presented RésuMatcher, a personalized job-résumé matching system that can help job seekers find appropriate jobs faster and more accurately by using their résumé contents. The key components of the system are the information extraction procedure and the ontology matching module.

In the system, job descriptions and résumés are parsed into job models and résumé models by the information extraction module. When searching the jobs by a résumé, similarity values between the résumé model and job description models are calculated in the ontology matching module. The result is sorted by the ontology similarity scores, which are the sum of similarities of different fields multiplied by their weights. We made such contributions in our works:

1. We developed a résumé - job matching system.
2. We developed a finite state transducer based matching tool to extract information from unstructured data source, which is a lightweight and flexible library, and can be extended in very easy ways.
3. We developed a semi-automatic approach, which can collect technical terms from hr data sources, and by which we created a domain specific ontology for recruitment.
4. We developed statistical-based ontology similarity measure, which can measure the similarities between technical terms .

In the experiment phase, we evaluated the accuracy of information extraction. We calculated the ontology similarity with the NDCG. Finally, we also tested the performance of résumé job matching algorithm via precision@k and NDCG, which showed that our algorithm can achieve a better searching result than other information retrieval models like TF-IDF and OKpai BM25. We also compared our system with the commercial job search engine www.indeed.com, and the results showed that our system can return jobs with a better ranking.

9.2 Future Work

Finding a job is a complex process, affected by both explicit and implicit factors. Our work establishes the validity of using information extraction techniques to create a more personalized job matching system, with ample potential for improvement in the future.

First we can introduce a more complex job and rsum model to improve performance of the system. In the résumé model, we can consider hiring history and project experience of the job seekers. To improve the job description model, job responsibilities and company characteristics (size, dress code, etc.) should be considered as well.

Second, to improve searching speed of our system, we can reduce the the number of comparison by filtering out jobs that are clearly not related to résumés. The system can classify the jobs into some different subsets, when searching jobs, the system only need to calculate the similarity between the résumé and according subset of jobs.

RésuMatcher is a content based recommendation system that is mostly focused on comparing the similarities between the résumé and a relevant job description. In future work, we could introduce a hybrid recommendation system that would take advantage of other recommendation algorithms such as Collaborative Filtering. Future work on this system would place greater consideration on job seeker's personal preference like job location, career development plan, and company background.

REFERENCES

[1] ACM. Acm computing classification system, 2012.

[2] Alfred V Aho and Jeffrey D Ullman. *Foundations of computer science*, volume 2. Computer Science Press New York, 1992.

[3] Shaha T Al-Otaibi and Mourad Ykhlef. A survey of job recommender systems. *International Journal of the Physical Sciences*, 7(29):5127–5142, 2012.

[4] Douglas E Appelt and Boyan Onyshkevych. The common pattern specification language. In *Proceedings of a workshop on held at Baltimore, Maryland: October 13-15, 1998*, pages 23–30. Association for Computational Linguistics, 1998.

[5] Steven Bird. Nltk: The natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics, 2006.

[6] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia-a crystallization point for the web of data. *Web Semantics: Science, services and agents on the world wide web*, 7(3):154–165, 2009.

[7] Duygu Çelik and Atilla Elçi. An ontology-based information extraction approach for résumés. In *Pervasive Computing and the Networked World*, pages 165–179. Springer, 2013.

[8] Angel X Chang and Christopher D Manning. Tokensregex: Defining cascaded regular expressions over tokens. Technical report, Technical Report CSTR 2014-02, Department of Computer Science, Stanford University, 2014.

[9] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. A framework and graphical development environment for robust nlp tools and applications. In *ACL*, pages 168–175, 2002.

[10] J Olawande Daramola, Olufunke O Oladipupo, and AG Musa. A fuzzy expert system (fes) tool for online personnel recruitments. *International Journal of Business Information Systems*, 6(4):444–462, 2010.

[11] Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: A search and metadata engine for the semantic web. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659. ACM, 2004.

[12] Ying Ding and Schubert Foo. Ontology research and development. part 1-a review of ontology generation. *Journal of Information Science*, 28(2):123–136, 2002.

[13] Athanasios Drigas, Stelios Kouremenos, Spyros Vrettos, John Vrettaros, and Dimitris Kouremenos. An expert system for job matching of the unemployed. *Expert Systems with Applications*, 26(2):217–224, 2004.

[14] Frank Färber, Tim Weitzel, and Tobias Keim. An automated recommendation approach to selection in personnel recruitment. 2003.

[15] 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.

[16] Christiane Fellbaum. *WordNet*. Wiley Online Library, 1998.

[17] Tere Gonzalez, Pano Santos, Fernando Orozco, Mildreth Alcaraz, Victor Zaldivar, Alberto De Obeso, and Alan Garcia. Adaptive employee profile classification for resource planning tool. In *SRII Global Conference (SRII), 2012 Annual*, pages 544–553. IEEE, 2012.

[18] Hal G Gueutal and Dianna L Stone. *The brave new world of eHR*. John Wiley & Sons, 2006.

[19] Jerry R Hobbs, Douglas Appelt, John Bear, David Israel, Megumi Kameyama, Mark Stickel, and Mabry Tyson. 13 fastus: A cascaded finite-state transducer for extracting information from natural-language text. *Finite-State Language Processing*, page 383, 1997.

[20] Jay J Jiang and David W Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *ArXiv Preprint Cmp-lg/9709008*, 1997.

[21] V Senthil Kumaran and A Sankar. Towards an automated system for intelligent screening of candidates for recruitment using ontology mapping (expert). *International Journal of Metadata, Semantics and Ontologies*, 8(1):56–64, 2013.

[22] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[23] Claudia Leacock and Martin Chodorow. Combining local context and wordnet similarity for word sense identification. *WordNet: An Electronic Lexical Database*, 49(2):265–283, 1998.

[24] Danielle H Lee and Peter Brusilovsky. Fighting information overflow with personalized comprehensive information access: A proactive job recommender. In

Autonomic and Autonomous Systems, 2007. ICAS07. Third International Conference on, pages 21–21. IEEE, 2007.

[25] Wendy Lehnert, Claire Cardie, David Fisher, Ellen Riloff, and Robert Williams. University of massachusetts: Description of the circus system as used for muc-3. In *Proceedings of the 3rd conference on Message understanding*, pages 223–233. Association for Computational Linguistics, 1991.

[26] Dekang Lin. An information-theoretic definition of similarity. In *ICML*, volume 98, pages 296–304, 1998.

[27] Ping Liu and Peter Dew. Using semantic web technologies to improve expertise matching within academia. *Proceedings of I-Know (Graz, Austria*, pages 370–378, 2004.

[28] Yao Lu, Sandy El Helou, and Denis Gillet. A recommender system for job seeking and recruiting website. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 963–966. International World Wide Web Conferences Steering Committee, 2013.

[29] 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.

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

[31] George A Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[32] Natalya F Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W Fergerson, and Mark A Musen. Creating semantic web contents with protege-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.

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

[34] Roy Rada, Hafedh Mili, Ellen Bicknell, and Maria Blettner. Development and application of a metric on semantic nets. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(1):17–30, 1989.

[35] Rachael Rafter, Keith Bradley, and Barry Smyth. Personalised retrieval for online recruitment services. In *Proceedings of the 22nd Annual Colloquium on Information Retrieval. 2000.*, 2000.

[36] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. *ArXiv Preprint Cmp-lg/9511007*, 1995.

[37] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Information Retrieval*, 3(4):333–389, 2009.

[38] 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.

[39] Emmanuel Roche and Yves Schabes. *Finite-state language processing.* MIT press, 1997.

[40] M Andrea Rodríguez and Max J. Egenhofer. Determining semantic similarity among entity classes from different ontologies. *Knowledge and Data Engineering, IEEE Transactions on*, 15(2):442–456, 2003.

[41] David Sánchez, Montserrat Batet, David Isern, and Aida Valls. Ontology-based semantic similarity: A new feature-based approach. *Expert Systems with Applications*, 39(9):7718–7728, 2012.

[42] Sunita Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008.

[43] 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.

[44] Zheng Siting, Hong Wenxing, Zhang Ning, and Yang Fan. Job recommender systems: A survey. In *Computer Science & Education (ICCSE), 2012 7th International Conference on*, pages 920–924. IEEE, 2012.

[45] Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

[46] Amos Tversky. Features of similarity. 1977.

[47] Andrew J Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, 1967.

[48] Kangning Wei, Jinghua Huang, and Shaohong Fu. A survey of e-commerce recommender systems. In *Service Systems and Service Management, 2007 International Conference on*, pages 1–5. IEEE, 2007.

[49] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics, 1994.

[50] 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.

[51] Kun Yu, Gang Guan, and Ming Zhou. Resume information extraction with cascaded hybrid model. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 499–506. Association for Computational Linguistics, 2005.

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

[53] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Computing Surveys (CSUR)*, 38(2):6, 2006.