Crowdsourced Query Augmentation through Semantic Discovery of Domain-specific Jargon

Khalifeh AlJadda
Department of Computer Science
University of Georgia
Athens, Georgia
aljadda@uga.edu

Mohammed Korayem
School of Informatics & Computing
Indiana University
Bloomington, Indiana
mkorayem@indiana.edu

Trey Grainger
Director of Engineering
CareerBuilder Search Group
Norcross, Georgia
trey.grainger@careerbuilder.com

Chris Russell
Engineering Lead, Relevancy
CareerBuilder Search Group
Norcross, Georgia
chris.russell@careerbuilder.com

***Abstract*—Most work in semantic search has thus far focused upon either manually building language-specific taxonomies/ontologies or upon automatic techniques such as clustering or dimensionality reduction to discover latent semantic links within the content that is being searched. The former is very labor intensive and is hard to maintain, while the latter is prone to noise and may be hard for a human to understand or to interact with directly. We believe that the links between similar user's queries represent a largely untapped source for discovering latent semantic relationships between search terms. The proposed system is capable of mining user search logs to discover semantic relationships between key phrases in a manner that is language agnostic, human understandable, and virtually noise-free.**

I. Introduction

The most commonly used technique for implementing a text search engine is the creation of an inverted index of all of the unique terms mapped to the documents in which each term can be found [1]. This technique enables direct lookups of a set of documents matching any term, and it also makes it easy to perform complex Boolean queries by performing set operations on the document sets for multiple terms. For example, the query $java\ developer$ would become a Boolean query of $java\ AND\ developer$, which would return the set of all documents which match the word $java$ intersected with the set of all documents which match the word $developer$. This is a naive approach to searching text, however, as many keywords have alternate meanings when combined with other keywords and can thus match documents that are unrelated to the real intent of the person submitting the query.

Others have attempted to solve this problem through the manual creation of taxonomies/ontologies per language or the automatic dimensionality reduction of the data source being searched into blurred latent semantic representations. The former approach requires custom natural language processing algorithms per language, and the latter can result in a loss of meaning and a representation that is not necessarily indicative of its common human expression. Our goal is to utilize the wisdom of the crowd to demonstrate a language-agnostic technique for discovering the intent of user searches by revealing the latent semantic relationships between the terms and phrases within search queries. Our hope is to express these relationships in common human language so that we can automatically augment a user's queries using terminology that the user can easily understand and adjust.

Fig. 1. Abstract view of the proposed model which utilizes the search behavior from both sides of a two-sided market and eliminates noise by utilizing content-based filtering. The shaded area represents the final result set which is the most accurate list of related terms for any given term

The specific implementation of our technique will be presented in the context of an employment search engine in English, but the technique is both domain- and language-agnostic. To demonstrate the problem, consider the different meanings of the word developer in jobs for land developers versus software developers. In aggregate, when users search for the keyword $developer$, they are usually searching for a software developer, though the search engine does not natively know this and will return documents for both software developers and land developers. Take another example of software architects versus building architects. Even if a user searches for a specific phrase like $software\ architect$, the search engine does not natively know that these two words form a meaningful phrase within the domain of employment search, so the search engine is likely to execute a search for $software\ AND\ architect$ as independent terms, which may yield results for building architect jobs that require the use of some kind of software. There is clearly room for improvement here in identifying key phrases and relating them to their intended meaning.

It is difficult to determine the semantics of the keyword "developer" or "architect" based solely upon these single searches. However, we have access to the search history of millions of users. When considered in aggregate this history allows us to discover the relationship between search terms and the most common meaning of each term. Once we know the semantic relationships between terms according to our users, we can use those relationships to expand the search keywords

978-1-4799-5666-1/14/$31.00 ©2014 IEEE

in order to more accurately express the intent of the user.

Our solution to this challenge makes use of the wisdom of the crowd to discover domain-specific relationships. Using the query logs of over a billion user searches we can discover the family of related keyword phrases for any particular term or phrase for which a search has been conducted. We are not attempting to fit these terms into an artificial taxonomy; instead we are discovering the existing relationships between terms according to our users. Since these logs are all within the domain of employment search, we avoid the ambiguity of a wider context and are able to automatically identify domain-specific jargon. In addition, our solution is language-independent since it is discovering relationships between terms and phrases based upon the behavior of real users rather than an understanding of their language. We can also determine the strength of the relationship between terms by examining the frequency of co-occurrence and statistical independence of co-occurring terms.

In order to eliminate noise (false positives) in our algorithm, we make use of three datasets: the query logs of job seekers, the query logs of recruiters, and the content of the job postings which are provided by recruiters and searched by job seekers. Each of these datasets could be used independently to derive semantic relationships, but each dataset is also prone to its own, unique, kind of noise. For example, the job seeker search behavior represents our largest source of data. With these searches coming from a publicly available search engine that any person or computer system could send queries to, however, we have found that many terms present are from automated processes and not users searching for related phrases. With regard to the recruiter search behavior, we have found that many recruiters search for overly-specific terms such as people's names or demographic information, which we would not want to add to our semantic relationship mappings. Finally, when considering the content of actual job postings, it would be possible to use natural language processing and statistical analysis of term and phrase independence, but the output would draw semantic links based upon entire documents instead of queries, so the semantic relationships may not be in a form that would actually be natural for expression as a user query. Analyzing the content would also require independent parsing and analysis for each supported language, which would thus violate our goal of creating a language-independent algorithm.

Our approach seeks to combine the best from each approach - finding latent semantic links between job seeker search phrases, intersecting them with latent semantic links discovered between recruiter search phrases, and ultimately validating that each of the phrases exists with a high enough threshold to be meaningful within the job postings being searched. This ensures that only semantic links which are universally agreed upon by both sides of this two-sided market appear in our final list of relationships, while also preventing us from augmenting queries with terms that would yield no search results. The final result is a high-precision data set that reveals the latent semantic relationships between search terms and phrases within our domain.

The main contribution of this paper is thus a language-agnostic methodology for discovering the latent semantic relationships between domain-specific terminology such that those links can be used to better express the intent of a user-entered search in a way that the user could understand and adjust.

II. Related Work

Utilizing relationships between similar phrases is commonplace in most search and natural language processing systems. For a general purpose search engine, a publicly-available list of synonym relationships can often be utilized, such as the WordNet lexical database [2]. For a domain-specific taxonomy, a human expert will generally be used to compile a list of important synonym relationships. In both cases, these approaches require substantial manual work which must be repeated on a language-by-language basis for each written language supported. Other approaches, such as Latent Semantic Indexing (LSI) , attempt to automate the process of discovering semantic relationships through performing Singular Value Decomposition upon all terms within a search index to reduce related terms to a common semantic form. This can be useful for inferring relationships between multiple documents containing similar but not identical terms, though the forms represented in the dimensionally-reduced matrix may be expressed in a way that is nonsensical for human understanding [3], [4]. Likewise, LSI may tend to blur relationships beyond what a human searcher would expect with no easy ability to undo the dimensionality reduction. A third approach, typically utilized within the context of recommender systems, is that of Collaborative Filtering (CF) [5]. In CF, the textual content of documents is completely unused for purposes of deriving relationships between the documents. Instead, links between documents are formed when a human acts upon two or more documents. The assumption is that the person executing a query is looking for related documents, so the fact that a user takes action upon two documents within a given context implies that the person searching believes the documents are related. After aggregating all of these links between users and documents, other similar documents can be recommended by first finding all users who acted upon the document in question and then returning the most common other documents those users also acted upon. CF is effectively a human-machine algorithm which has been shown to consistently beat content-based recommendation approaches due to its ability to crowdsource human knowledge of relationships between two or more items [1].

III. Methodology

A. Problem Description

We would like to create a language-independent algorithm for modeling semantic relationships between search phrases that provides output in a human-understandable format. It is important that the person searching can be assisted by an augmented query without us creating a black-box system in which that person is unable to understand and adjust the query augmentation. CareerBuilder[1] operates job boards in many countries and receives tens of millions of search queries every day. Given the tremendous volume of search data in our logs, we would like to discover the latent semantic relationships between search terms and phrases for different region-specific websites using a novel technique that avoids the need to use

[1] http://www.careerbuilder.com/

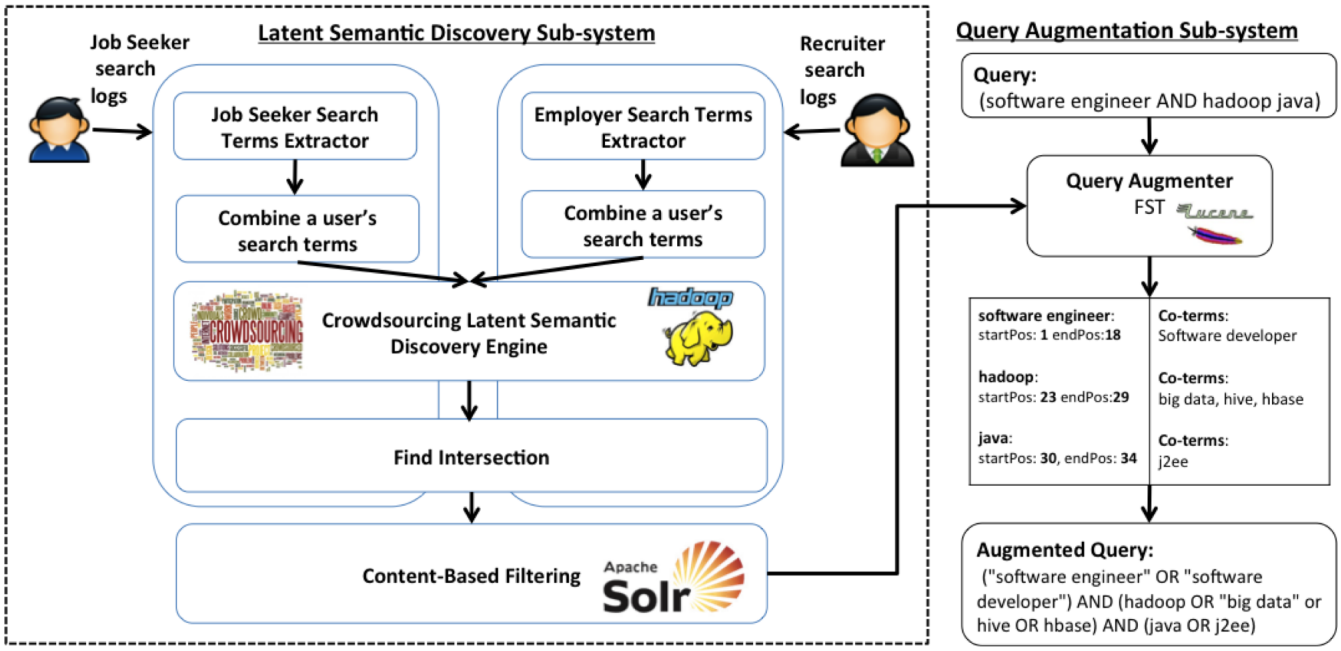

Fig. 2. System Architecture. The system collects the terms used on both sides of a two-sided market (job seekers and recruiters) and sends them to the crowdsourcing latent semantic discovery engine. The engine utilizes a combined co-occurrence and pointwise mutual information similarity measure to perform collaborative filtering and uncover the latent links between semantically-related terms. The results from both sides are independently scored and intersected to remove noise unique to each side of the market. The refined list of semantically related search terms is finally sent to a content-based filtering phase, which checks the search engine (based upon Apache Lucene/Solr) to ensure that the pair of related search terms is actually found together commonly within real documents within the domain. The resulting related terms are then stored in a Finite State Transducer, which is used to identify and expand known phrases on future, real-time user queries.

natural language processing (NLP). We wish to avoid NLP in order to make it possible to apply the same technique to different websites supporting many languages without having to change the algorithms or the libraries per-language.

It is tempting to suggest using a synonym dictionary since the problem sounds like finding synonyms, but the problem here is more complicated than finding synonyms since the search terms or phrases on our site are often job titles, skills, and company names which are not, in most cases, regular words from any dictionary. For example if a user searched for "$java\ developer$", we wouldnt find any synonyms for this phrase in a dictionary. Another user may search for "hadoop" which is also not a word that would be found in a typical English dictionary.

B. Proposed System

This challenging problem in the domain of employment search exists in many other domains, such as social network analysis [6], [7] and ecommerce websites like Amazon.com [8]. To tackle this problem we propose a hybrid system that employs crowdsourcing latent semantic discovery with content-based filtering to discover the latent relationships between the terms used by the users in a given domain (jargon). Figure 2 shows the system architecture. The proposed system starts by collecting the terms searched for by each user from a query log file. Our hypothesis is that there is often a relationship among the various search terms used by each individual user. As such, we combine all of the terms that a specific user searched for in one list as potentially related terms. This list is not sufficient to conclude that two terms are related to each other; we need some kind of validation for each relationship. Crowdsourcing is a powerful technique which is applicable in this case to validate the hypothesis that two terms are related. For any particular pair of search terms we can examine the query logs to determine how many users searched for both of those terms. The higher this number is, the more evidence we have that the two terms are related. To improve the crowdsourcing accuracy, we utilized the classification of the users who performed the searches (Figure 3). This is a very important step to eliminate noise and increase our confidence in the discovered related terms. If term x and term y appeared together in the searches of users who belong to the same class, then the confidence that the two terms are related increases more than if the two terms appeared together in the searches performed by users belonging to different classes. For example, if two users from the "Java Developer" class searched for "$java$" and "$j2ee$" while a user from the "Java Developer" class and another from the "Barista" class searched for "$java$" and "$coffee$" then the former two terms "java" and "j2ee" are likely to be more related within our domain than "java" and "coffee".

Since the focus of our semantic discovery is on employment searches, we decided to classify the users based upon their individual behavior in applying to jobs. For each user we examine all of his/her job applications and the classification assigned to each job to which the user applied. Finally, we apply a majority voting technique to assign a class for each user (Algorithm 1).

Data: $List < User >$
Result: Classified Users
foreach *user in* $List < User >$ **do**
  $List < JobApp > \leftarrow getJobApplications(user)$
  foreach *jobApp in* $List < JobApp >$ **do**
    jobClass $\leftarrow getClass(jobApp)$
    count $\leftarrow getCount(jobClass, user) + 1$
    $setCount(jobClass, count, user)$
  end
  max = 0
  foreach *jobClass in* $List < JobClass >$ **do**
    if getCount(jobClass, user) $>$ max **then**
    max $\leftarrow getCount(jobClass, user)$
    userClass $\leftarrow jobClass$
  end
end
classifiedUsers.$add(user, userClass)$

Algorithm 1: Classification Algorithm Using Users' Job Apps

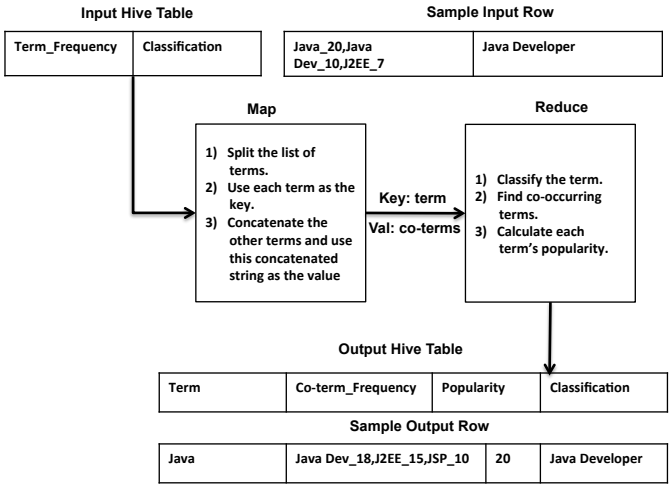

Fig. 3. The crowdsourcing latent semantic discovery engine implementation over Hadoop using map/reduce jobs

C. Scoring

One of the most important and critical aspects of our work is the scoring of the related terms in order to rank them in a way that represents how relevant they are to a given term. We utilize two scoring techniques for our work:

1) Co-occurrence score
2) Pointwise mutual information (PMI) Score

1) Co-occurrence Score: The co-occurrence score is calculated during the discovery of the latent semantic relationship between the search terms. This score basically represents how many distinct users searched for $term_x$ and its related term $term_y$. Since this score is based on the number of users who searched for both terms, this score is biased by the popularity of the term. To explain this problem let's assume that $term_x$ was searched by 100 distinct users, that its co-term $term_y$ was searched by 50 of those distinct users, and that its co-term $term_z$ was searched by only 20 of those distinct users. Based on the co-occurrence score, $term_y$ will be ranked before $term_z$. But, what if we know that the total number of distinct users who searched for $term_y$ was 200 while the total number of users who searched for $term_z$ was 20? This means that all of the users who searched for $term_z$ used it with $term_x$. Thus, the relationship between $term_z$ and $term_x$ is stronger than the relationship between $term_y$ and $term_x$. Unfortunately, because $term_z$ is less popular than $term_y$ it is penalized here.

2) Pointwise Mutual Information (PMI) Score: To overcome the problem with the co-occurrence score, we applied the well known scoring technique Pointwise Mutual Information [9], [10]. PMI normalizes the co-occurrence score and provides a chance for the less popular terms to appear at the top of the related terms list if they are strongly related to the given term [11]. We calculated PMI^2 using the following equation:

$$PMI^2(x, y) = log \frac{P(x, y)^2}{P(x) \times P(y)} \quad (2)$$

The results we observed after applying PMI were better than the results of co-occurrence scoring. However, we noticed that some popular terms were penalized when they appeared with each other because of their popularity, while rarely used terms were pushed up in the rankings. By looking at the results of both scoring techniques, we concluded that neither single scoring technique was sufficient to get the best ranked result set and we needed to find a way to combine both scores to get the best results.

3) Final Ranking: To combine both scores in order to get the best results, we decided to use the rank of each co-term in the list of co-terms as the key to combine the results of the two different scores. The proposed ranking algorithm aims to take the best and avoid the worst of each scoring approach. In this ranking technique we sort the co-terms list by co-occurrence score and name this $list_a$. We then sort the initial list again by the PMI score and name that list $list_b$. We take the rank of $item_x$ in $list_a$ as r_a and we take the rank of that term in the $list_b$ as r_b. The new combined score (we call it Composite Ranking Score, or CRS) is calculated by the following equation:

$$CRS(item_x) = \frac{\frac{r_a + r_b}{r_a \times r_b}}{2} \quad (3)$$

If $term_x$ doesn't appear in both lists we use α as a static rank in the list that is missing $term_x$ where $\alpha = length(list) + 1$.

D. Filtering

In order to eliminate noise and further improve the accuracy of the proposed technique, we applied three types of filtering:

1) Collaborative user intersection pre-filtering which applies before the calculation of the co-occurrence and the PMI scores.
2) Two-sided market agreement post-filtering which occurs after the Composite Ranking Score has been calculated for both job seeker related search terms and recruiter related search terms.
3) Content intersection post-filtering which applies after the intersected list of job seeker and recruiter searches is found.

For the two-sided market agreement pre-filtering phase, we decided to apply another collaborative filtering model to the logs of resume searches performed by recruiters. The reason for this is to reduce noise caused by web crawlers (bots) which indiscriminately execute queries to harvest content from CareerBuilder's websites. These bots will execute common, but unrelated, queries thousands of times resulting in incorrect and noisy links between unrelated search terms.

Taking advantage of the fact that CareerBuilder is a two-sided market (allowing companies to post jobs to be searched by job seekers and for job seekers to post resumes to be searched by recruiters), we decided to run through a similar workflow with our resume search logs as previously described for our job search logs in order to create a list of related terms according to our corporate users. Because the noise created by bots would not be represented in recruiter searches (which require authentication and paid access), and because recruiters sometimes search for very specific terms (such as people's names), we intersect the lists of recruiter search terms and job seeker search terms, using a reasonable threshold for the number of distinct users executing searches on both sides of the market. This removes virtually all noise created by either side of the market. Any terms not used by a large enough number of job seekers and recruiters are filtered out from the main list prior to looking for related search terms through the co-occurrence or PMI models.

In the next prefiltering phase, we look at collaborative user-intersection by using a threshold β that represents the minimum number of distinct users who needed to search for two terms, $term_x$ and $term_y$, in order for the relationship between the terms to be considered more than coincidental. If $term_x$ and $term_y$ were searched for by a number of distinct users $n < \beta$, we exclude them from the results as related terms.

In our content intersection post-filtering phase (Figure 4), we applied the search engine itself as part of our workflow for post-filtering, since our ultimate goal is to find the most related search terms that have actually been used within the search engine. For each term we examine all of its relationships. For each relationship we run a query using the original term and another query using the related term. We compare the two resulting document sets and look for intersection. If there are too few intersections we consider the relationship to be invalid and remove it.

E. Query Augmentation

With a high-precision list of semantically-related search terms defined, we need some way to utilize this data to automatically enhance incoming queries. This requires having a fast, real-time ability to perform entity extraction for multi-word phrases on incoming content (queries or documents) in a way that preserves the most specific meaning of each term. For example, if someone were to submit a query to the search engine for $machine\ learning\ research\ and\ development\ Portland,\ OR\ software\ engineer\ AND\ hadoop\ java$, a traditional Boolean query parser may interpret this as $(machine\ AND\ learning\ AND\ research\ AND\ development\ AND\ portland)\ OR\ (software\ AND\ engineer\ AND\ hadoop\ AND\ java)$, which would yield results very different than what the user is expecting. In reality, the user's intent is something much more like the following query: "$machine\ learning$" AND "$research\ and\ development$" AND "$Portland,\ OR$" AND "$software\ engineer$" $AND\ hadoop\ AND\ java$.

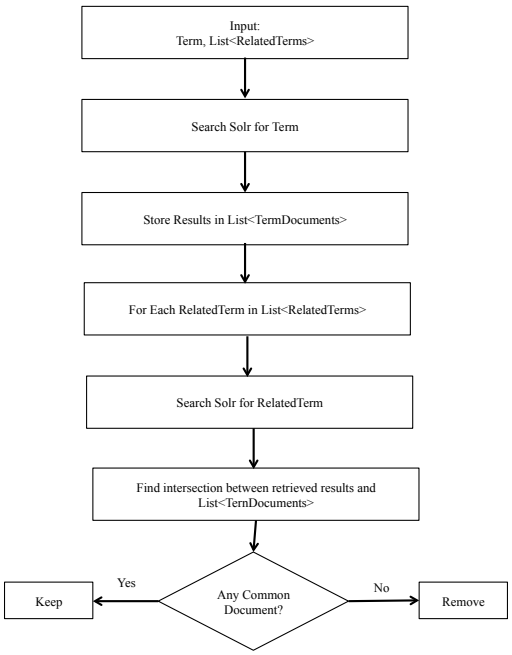

Fig. 4. Content based filtering using Solr. Solr is searched for the term, then the retrieved set of documents is used to find the intersection with the document set retrieved by searching each candidate related term. Candidate related terms without intersection with the given term are removed from the related search terms list for that term.

Because our proposed system both identifies key phrases within a domain and also maps those to semantically-related phrases, if we can extract the key phrases within incoming queries then we will actually be able to expand our example query to something more like this: $(\text{"}machine\ learning\text{"}\ OR\ \text{"}computer\ vision\text{"}\ OR\ \text{"}data\ mining\text{"}\ OR\ matlab)\ AND\ (\text{"}research\ and\ development\text{"}\ OR\ \text{"r\&d"})\ AND\ (\text{"}Portland,\ OR\text{"}\ OR\ \text{"}Portland,\ Oregon\text{"})\ AND\ (\text{"}software\ engineer\text{"}\ OR\ \text{"}software\ developer\text{"})\ AND\ (hadoop\ OR\ \text{"}big\ data\text{"}\ OR\ hbase\ OR\ hive)\ AND\ (java\ OR\ j2ee)$. There are many ways to manipulate this query with different Boolean logic to achieve different levels of precision and recall in the search results, but the key value our system adds is the ability to better understand the semantics of the terms within the query and expand the original query to highly-related terms within the domain.

In order to achieve this objective, we insert an entity tagging stage at the beginning of our query processing pipeline. This entity tagging makes use of a minimal, deterministic, Finite State Transducer (FST), which is a very fast, memory efficient data structure useful for representing natural language dictionaries [12]. Specifically, we harness the Lucene FST implementation as exposed through the SolrTextTagger open

source project[2] [13]. We establish a minimal Solr text analysis pipeline suitable for splitting tokens from inputted documents and queries, and then we proceed to build an FST containing every phrase in our domain-specific list of phrases.

Once the FST is built, we can then send our user-entered queries into the SolrTextTagger, which will tokenize the query and iterate through all the tokens (from left to right) to see if any of our domain-specific phrases in the FST begin with that token. If the token is found, the process will combine it with the next token to determine whether both tokens exist together as a phrase. This process continues until the longest possible phrase is identified, and then all known terms and phrases are tagged for later processing. As shown in the query augmentation sub-system in Figure 2, this FST process effectively tags all of the known phrases within the incoming query so that each phrase can be augmented in-place with a more helpful expression of the semantic intent of that phrase.

One benefit of this approach is that if one term is a subset of a longer phrase, the longer phrase is the one tagged. For example, "learning" is a sub-term of "machine learning", but it has a very different meaning when considered within the longer phrase. Any unidentified terms are left as-is in the query to be treated as independent keywords. Once all of the known phrases have been tagged with their starting and ending positions within the incoming query, we can easily substitute the original phrase with our semantic understandings of what that phrase means based upon the previously-described Boolean expansion of each phrase to its related phrases. The fully-expanded query can then be sent to a Boolean parser, and the final query can be executed and scored using the pre-existing methods that would otherwise be applied for an unmodified traditional keyword search. It is worth noting that we actually weight the relevancy of each expanded term based upon the calculated CRS Score in order to ensure the known strength of the semantic relationship influences the final relevancy scores of the matching documents. As such, when stronger semantically-linked phrases are matched, the documents containing those stronger links score higher than documents containing weaker semantic relationships to the terms in the users original query.

IV. Experiment and Results

In this section we describe how we successfully applied the proposed system to discover the latent semantic relationships between search keywords for the largest job board in the US.

A. Use Case

CareerBuilder operates the largest job board in the U.S. and has an extensive and growing global presence, with millions of job postings, more than 60 million actively-searchable resumes, over one billion searchable documents, and more than a million searches per hour. Historically the primary approach to relevancy has been a keyword matching based approach that combines a Boolean filtering model, a term frequency / inverse document frequency-based scoring model (tf-idf) [14], as well as common text-normalization and linguistic analysis techniques such as per-language lemmatization. Although this approach is often quite effective, there are many instances where a strictly keyword based approach is insufficient to capture the intent of the user. In these instances of ambiguity, it would be best if we could understand the intent of the user's query.

[2]http://www.opensextant.org/

TABLE I. Accuracy for samples

Sample	Precision Before CBF using Slor	Precision After CBF using Solr
Most Popular 100 terms	87.9%	98.3%
Least Popular 100 terms	83.1%	96.8%
Random 100 terms	87.3%	98.6%

B. Experiment Setup

We ran our experiments using 1.8 billion search records from CareerBuilder's search engine query logs. We only considered sets of terms to be potentially related (and subject to further processing) if at least 10 distinct users searched for both of the terms. This eliminated considerable noise from our results.

We used a Hadoop[3] cluster with 63 data nodes each having a 2.6 GHZ AMD Opteron Processor with 12 to 32 cores and 32 to 128 GB RAM. The total execution time for the proposed system on the given dataset was 1.5 hours. The technologies used to implement the proposed system for CareerBuilder's data set are: HDFS [15], Map/Reduce jobs [16], Hive [17], and Solr[4].

C. Results and Evaluation

The size of the final high-precision table after all filtering stages is 11,304 unique terms (out of more than 50,000 total terms identified prior to aggressive precision-driven filtering). To evaluate the results we had to assess the quality of 3000 pairs of discovered term relationships and count how many of those pairs were actually related to each other. Then, by taking the fraction of the related pairs to the total number of selected pairs, we calculated the accuracy. We repeated the same evaluation process three times: in the first round we selected 1000 pairs among the most popular terms, in the second round we selected 1000 pairs among the least popular terms, and in the last round we randomly selected 1000 pairs of terms. Table I shows the accuracy of the different test sets, while table III shows how content-based filtering (CBF) discovered the outliers and cleaned up the list of discovered related terms to improve the accuracy.

By examining table II we can see that neither the original job seeker list nor recruiter list is accurate in all cases. However, the intersection list is much more accurate. It is also obvious that in most cases, especially for the terms related to emerging technologies like "data scientist", the recruiters' background and knowledge was not sufficient to search for related skills or titles. As a result, the recruiters often searched for terms that are totally irrelevant but contain the term "data" like "data management assistant" or "data assistant". Meanwhile, the job seeker list for the same term shows very clean and related co-terms, since the job seekers usually work in this field and know the set of skills and other titles that

[3]http://hadoop.apache.org
[4]http://lucene.apache.org/solr

TABLE II. ORDERED RESULTS OF JOB SEEKERS LIST, RECRUITERS LIST, AND THE FINAL INTERSECTED LIST

Term	Job Seekers	Recruiters	Intersection
Cashier	retail cashier, customer service, cashiers, receptionist, jobs, retail, ...	host, retail, floater, sales, trainee, waiter, customer service, ...	retail, retail cashier, customer service, cashiers, receptionist, cashier jobs, teller, ...
Food Service	constr, restaurant food service, customer service, food service worker, food,warehouse, ...	food, foodservice, foodservices, restaurant, retail sysco, grocery, ...	food, constr, foodservice, restaurant, restaurant food service, ...
Chief	director, president, vice president, vp, ...	director, head, vice president, officer, vp, coo, manager, ...	vice president, director, vp, president, coo, ...
Scrum	agile, scrum master, project manager, scrummaster, csm, ...	scrum master, waterfall, scrummaster, team foundation services, windows forms,winforms, ...	scrum master, agile, scrummaster, project manager
Agile	scrum, project manager, agile coach, pmiacp, scrum master, ...	scrum, waterfall, ajax, xml, javascript, java, embedded, ...	scrum, project manager, agile coach, pmiacp, scrum master, ...
Plumbing	plumber, plumbing apprentice, plumbing maintenance, plumbing sales, maintenance, ...	fire protection, process systems, uninterruptible power systems, sheetmetal, ups, ...	plumber, plumbing apprentice, plumbing maintenance
Sales Rep	electrican and jr mechanic, sales representative, sales, customer experience manager and nps, member relationship manager, ...	sales, uniform sales rep, sales representative, territory manager, territory rep, ...	sales, sales representative, electrican and jr mechanic, customer experience manager and nps, ...
Realtor	realtor assistant, real estate, real estate sales, sales, real estate agent, ...	residential real estate, realty, sales insurance life license, series , strategic sales, ...	realtor assistant, real estate, real estate sales, realty, sales, real estate agent, ...
CDL	cdl driver, driver, plins marketing, chewstr, realpage senior living, ...	cdl a, class a, cdl-a, commercial drivers license, driver, cdl/a, truck driver, ...	cdl driver, cdl a, driver, plins marketing, truck driver, ...
Data Scientist	machine learning, data analyst, data mining, analytics, big data, statistics, ...	machine learning, data management assistant, data assistant, data specialist, ...	machine learning, data analyst, data mining, analytics, big data, statistics, ...
Machine Learning	computer vision, data mining, data scientist, matlab, image processing, ...	text mining, nlp, natural language processing, machine-learning, pattern recognition algorithms, ...	computer vision, data mining, data scientist, matlab
Data Mining	machine learning, data analyst, predictive modeling, sas, ...	machine learning, information extraction, text mining, nlp, natural language processing, ...	machine learning, data analyst, data scientist, predictive modeling, sas, analyst, ...
Big Data	hadoop, data scientist, analytics, java, business intelligence, business analyst, ...	hadoop, bigdata, mapreduce, chief research engineer, hbase, vp data analytics, ...	hadoop, data scientist, analytics, java, business intelligence, ...

TABLE III. SAMPLE RESULTS USING SOLR FOR CONTENT-BASED FILTERING (CBF)

Term	Before Solr	Removed Terms (Outliers)	Final List
Cashier	retail, retail cashier, customer service, cashiers, receptionist, cashier jobs, teller, ...	receptionist, teller	retail, retail cashier, customer service, cashiers, cashier jobs
Food Service	food, constr, foodservice, restaurant, restaurant food service, ...	constr., foodservice, customer service, sales	food, restaurant, restaurant food service
Sales Rep	sales, sales representative, electrican and jr mechanic, customer experience manager and nps, ...	electrican and jr mechanic, customer experience manager and nps	sales, sales representative
CDL	cdl driver, cdl a, driver, plins marketing, truck driver, ...	plins marketing, truck driver, chewstr	cdl driver, cdl a, driver
Data Scientist	machine learning, data analyst, data mining, analytics, big data, statistics, ...	data analyst, data mining, analytics, statistics	machine learning, big data

go with this term. Conversely, job seekers searching for less-skilled jobs such as "food service" positions tend to search for any job, not a specific type of job or related jobs. Thus, the related terms for those kinds of jobs in the job seeker list are not clean or accurate. The intersected list, while much smaller than either the job seeker or recruiter list, provides a very high precision mapping of semantically-related terms for safe use in automatically expanding user queries in a way they can understand and adjust.

V. CONCLUSION

Discovering the semantic similarity between domain-specific search phrases is very important for moving from a keyword-based search to a semantic search system. In this paper we proposed a novel technique to discover the latent semantic similarity between search terms by utilizing query logs from a two-sided market and applying a final content filtering stage. The proposed technique combines the well known semantic similarity scoring technique PMI with the co-occurrence score to rank the final list of the discovered related terms. This technique was applied successfully to discover a very high-precision list of related search terms using more than 1.6 billion search records from the search logs of CareerBuilder.com. One of the discovered limitations of this proposed system is that it's unable to resolve the ambiguity that exists in some edge cases between multiple meanings of the same term. For example, an intermediate list of the related phrases for the term "driver" contains both "truck driver" and "embedded system driver". Our system did a good job of picking the most common meaning of each term through requiring agreement across multiple perspectives (job seeker queries, recruiter queries, and job postings), but the other senses of the term were mostly lost in the process. We plan to resolve this challenge in future work by using a domain-specific classification system to group co-terms and determine if multiple disjoint clusters of co-terms appear across the different classifications, which would imply different word senses for the initial term within each cluster. With the exception of a few of these ambiguous edge cases, the proposed system serves as an excellent, automated model for discovering high-precision latent semantic links between important jargon within a specific domain and augmenting search queries to automatically include this enhanced understanding.

ACKNOWLEDGMENTS

The authors would like to thank Big Data team at CareerBuilder for their support with implementing the proposed system within CareerBuilder's Hadoop ecosystem. The authors would also like to show deep gratitude to the Search Development group at CareerBuilder for their help integrating this system within CareerBuilder's search engine to enable an improved semantic search experience.

REFERENCES

[1] T. Grainger and T. Potter, *Solr in Action*. Manning Publications Co., 2014.

[2] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[3] B. Rosario, "Latent semantic indexing: An overview," *Techn. rep. INFOSYS*, vol. 240, 2000.

[4] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *JASIS*, vol. 41, no. 6, pp. 391–407, 1990.

[5] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, pp. 285–295, ACM, 2001.

[6] P. Bhattacharyya, A. Garg, and S. F. Wu, "Analysis of user keyword similarity in online social networks," *Social network analysis and mining*, vol. 1, no. 3, pp. 143–158, 2011.

[7] P. Bhattacharyya, A. Garg, and S. F. Wu, "Social network model based on keyword categorization," in *Social Network Analysis and Mining, 2009. ASONAM'09. International Conference on Advances in*, pp. 170–175, IEEE, 2009.

[8] X. Jin and B. Mobasher, "Using semantic similarity to enhance item-based collaborative filtering," in *Proceedings of The 2nd IASTED International Conference on Information and Knowledge Sharing*, pp. 1–6, 2003.

[9] G. Bouma, "Normalized (pointwise) mutual information in collocation extraction," in *Proceedings of the Biennial GSCL Conference*, pp. 31–40, 2009.

[10] P. D. Turney, "Mining the web for synonyms: PMI-IR versus lsa on toefl," in *Proceedings of the 12th European Conference on Machine Learning*, EMCL '01, (London, UK, UK), pp. 491–502, Springer-Verlag, 2001.

[11] H. Zhang, M. Korayem, E. You, and D. J. Crandall, "Beyond co-occurrence: discovering and visualizing tag relationships from geo-spatial and temporal similarities," in *Proceedings of the fifth ACM international conference on Web search and data mining*, pp. 33–42, ACM, 2012.

[12] J. Daciuk, J. Piskorski, and S. Ristov, "Natural language dictionaries implemented as finite automata," in *Scientific Applications of Language Methods. Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory*, pp. 133–204, World Scientific Publishing, 2010.

[13] J. Daciuk and D. Weiss, "Smaller representation of finite state automata," in *Theoretical Computer Science. Volume 450*, pp. 10–21, Elsevier Science Publishers, 2012.

[14] A. Aizawa, "An information-theoretic perspective of tf–idf measures," *Information Processing & Management*, vol. 39, no. 1, pp. 45–65, 2003.

[15] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pp. 1–10, IEEE, 2010.

[16] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[17] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.