Dynamic Full Text Category Search

By

Patrick Wong

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the degree of

Master of Science

Major Subject: Computer Science

Approved:

_____

Mukkai Krishnamoorthy, Thesis Adviser

_____

Martin Hardwick, Committee Member

_____

David L. Spooner, Committee Member

Rensselaer Polytechnic Institute
Troy, New York

March, 2012
(For Graduation May 2012)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to acknowledge my advisor, Mukkai Krishnamoorthy, who gave me the idea on full text category searching. His advice, guidance, and support have been invaluable towards the completion of this thesis.

ABSTRACT

Finding a single or multiple well defined keywords in a document is a well-known problem with many solutions. But what happens if one wants to find a list of words that are related through some predefined categorization hierarchy? Furthermore, what if these hierarchies can change every week, day, or hour? In this paper, we implement a category searching system predicated on finding all related entries within a given document, whether that be a web-based HTML formatted file or a plain text file, of a user chosen predefined category. Users have the option of designing their own categories, using them to find matches within a document, and having their categories available for use by other users. Our system is based on part-of-speech tagging and uses a prebuilt part-of-speech regular expression grammar to locate all nouns and noun-phrases. For verifying our system, we built another system based on phrase matching. Results indicate that our phrase matching system actually has less false negatives and is almost three times as fast compared to our part-of-speech tagging system, but typically incurs more false positives. Overall, both systems on average contain little to no false negatives and are able to process an average HTML file size of 20k bytes in under 15 seconds.

1. INTRODUCTION

Classification and categorization are topics that have ceaselessly captivated the human imagination. People have constantly strived to find, relate, and organize the unknown into well defined collections. One could say that humans are inspired to uncover the truth and understand the unfamiliar. Mendeleev's periodic table and Dewey's Dewey Decimal System are such representations where people have sought to organize and understand the world around them. So it is not surprising that people are interested in organizing web pages into a classification hierarchy.

Automated classification is one of the rising fields in the area of categorization research. With the rise of the internet, more research has been conducted in understanding what web pages mean and how they are interrelated. News centric websites like The New York Times and The Wall Street Journal need ways to tag and categorize their various news articles in order to have them better target their specific audiences.

While much research effort has been spent on creating and improving algorithms designed for automated classification, this has not been the case for the field of full text category searching. We define full text category searching as the concept of locating all keywords in a particular document that are related to a specific category. And therein lays the distinction between the two: automated classification systems try to classify an entire article under some hierarchy, whereas full text category searching highlights all keywords in an article that satisfy the chosen category.

This can be further clarified when we look at two of the popular automated classification methods used to generate useful category groupings: clustering and faceted categorization. Clustering looks at associations and commonalities between words and phrases in order to group documents, whereas faceted categorization assigns attributes to each word or word phrase before using those attributes as a way to classify documents [5]. While both paradigms utilize category searching to determine if a particular document relates to a given user query, they do not highlight keywords that satisfy that particular category. As a result, the two paradigms fail at our intended goal: full text category searching.

In this paper, we implement a full text category searching system that allows users to select or build their own categories in order to locate and highlight all related words within any given document. We targeted web related content, specifically, HTML styled documents as the focused form for our system, but we also handle plain text files. For web-related media, we were especially worried about the processing time, since users expect fast response time when browsing on the web. Therefore, we cannot have it exceed a certain threshold; in this case, we targeted that each web page, regardless of size, needs to be executing under a minute.

Looking at previous works, we have not been able to identify papers or systems that attempt to accomplish full text category searching. Most papers and systems can be classified as either determining the category that a particular document should reside under or returning documents that are related to a given user query. In the former case, we have Devi, Rajaram, and Selvakuberan's [3] investigation on finding the minimum number of attributes necessary to successfully categorize a web page, Iwayama and Tokunaga's [6] work on comparing two cluster-based algorithms and evaluating their effectiveness on categorizing documents, Chekuri, Goldwasser, Raghavan, and Upfal's [2] research on increasing web search precision through automatic web page classification, and Qi and Davison's [10] exploration into the features, algorithms, and the current state of the art of web page classification. And in the latter case, we have Koren, Zhang, and Liu's [8] paper on using collaborative filtering and personalization to customize faceted search results, Tunkelang's [13] work on setting the most effective structured attributes necessary for faceted searching, Tomás and Vicedo's [12] multiple-taxonomy question classification system that returns documents correlating to a user specified category, and Kohlschütter, Chirita, and Nejdl's [7] research on using the web's link structure to determine the context of a web page.

Actually, the closest system to ours would probably be the web search engines, and in particular, the advanced web search option. The advanced search option allows users to specify or filter out words or phrases that they want to query against a wide variety of documents. Essentially, it allows users to enter in their own keywords and find related documents with all of those keywords highlighted. However, there are several limitations. First, those systems have a limit to the number of keywords that a user can

input and thus, can only locate a limited number of entries related to a particular category. Second, the category keywords have to be retyped each time one wants to submit a query. Third, the system conducts the category keyword search on a broad range of web pages and not for a specified document. Therefore, one of the key features of our system is that it gives users an automated function to find category related entries on a specified document, rather than having users manually search for them. And since a category may have many keywords or terms, users who manually scan a document are more likely to miss locating the keywords compared to our system.

The remainder of the paper is organized as follows. We discuss our logic and reasoning behind our design decisions, as well as the problems and challenges faced in section 2. Section 3 describes the implementation of our full text category searching system based off of part-of-speech tagging. In Section 4, we describe the need to verify our full text category searching system and the implementation behind the phrase matching system created subsequently. Results and findings from our two systems are discussed in section 5, and section 6 discusses our conclusions and future works.

2. DESIGN OF ARCHITECTURE

In order to implement a system that locates all words in a given body of text that match a user's specified category, we constructed a prototype system to understand the various problems behind designing such a system. From this analysis, we recognized and detailed several fundamental issues that need to be addressed. We then describe the different techniques we took in our actual implementation to resolve such concerns.

2.1 Background

Before we can build a fully effective full text category searching system, we decided on creating a prototype system to understand the issues we will be encountering. The simplest system to implement will be to utilize the brute force approach. The brute force approach consists of first parsing through the inputted text and verifying whether each individual word examined is in the specified category. For our prototype system, we decided on using a relational database, in particular, PostgreSQL, as the structure to contain all of the various keywords relating to each category. We chose to only support a single category: geographical locations. This database held roughly 1.5 GB worth of geographical data collected from the GeoNames geographical database. Ultimately, we realized that this system, not surprisingly, was ineffective.

For instance, we noticed that there were scores of false negatives occurring when evaluating various documents. In the brute force approach, we will parse through each word individually and determine if that word is found in the designated category. Numerous examples, however, suggest that a combination of words and not just a single word will be considered a match in a category; case in point, the phrase "New York" will be a match in the geographical locations category. Since the prototype system only does single word searching, it will ignore those word phrases and thus, not accurately locate all possible matches within a document.

To make matters worse, even if a match is confirmed, we have to determine whether or not that match makes any sense. Consider the word "Washington" as a match. This word, without any context, makes it difficult to determine if it refers to a person, like George Washington, or to a geographical location, like the state of Washington or the city of Washington D.C. Essentially, without the context of the

situation, there is absolutely no guarantee that the matches found for the specified category are accurate. Putting it simply, false positives can occur rather prevalently as we saw during our testing.

Another major blunder we detected was that our execution time was rather high. On average, when parsing through a document of 500 words, the execution time took around 17 minutes. As one can expect, we were somewhat perplexed by these turn of events. After more thorough investigation, we noticed that the execution time ET is related to query time QT and number of words seen $N_{words\ seen}$ as follows:

$$ET\ (in\ seconds) = QT\ (in\ seconds) \times N_{words\ seen}. \tag{2.1}$$

Accordingly, we deduced that there was about 2 seconds of time delay between each of our queries to the database and the returned result. This makes sense since the execution time took about 17 minutes or around 1,000 seconds and the document contained 500 total words, which when divided from one another, leads to 2 seconds. As one can discern, this is definitely not acceptable.

Realizing that the query time is causing the bottleneck, we investigated the reasoning behind this delay and identified several key components. First, there is the network and remote procedure call (RPC) latency between each request from our system and response from the database. Second, there is the overhead of creating the query execution plan by the database server. Lastly, just the sheer size of our database – it held 1.5 GB worth of geographical data from the GeoNames organization – affects the index size of the binary-trees within it and thus, forces the database to run more search operations to return the query results.

Because of those factors, we realized that the type of storage system that will be used to contain all of the various category word banks is another issue that needs to be tackled. By default, most will assume that using a relational database, like MySQL and PostgreSQL will be the wisest choice. As detailed previously, these databases come with drawbacks, specifically, latency issues. Two major factors influencing this include the size of the data set and the number of queries sent to the database; the former forcing each query to have to sift through more entries in the table or tables, and the later blogging down the database with continuous traffic. In our case, we issue complex queries that need to access and join multiple tables within the database. Consequently,

this leads to the database taking much longer to process the query compared to say accessing a single table with those entries having already been indexed.

Despite the inefficiencies, this prototype system has allowed us to indentify several central problems that need to be addressed. Namely, we need to first limit the number of words that we have to inspect against the selected category word bank; so instead of having to examine each word of an article, we only need to view a subset of them. Secondly, we need to increase the accuracy of locating all matches found in a document; this includes figuring out how to handle various combinations of word phrases. Thirdly, we need to determine the context of each match found within a document and verify that the term satisfies being classified under the particular category. Finally, we need some mechanism to reduce the time it takes to determine if a word from the document is found in the selected category.

2.2 Principle of Operations

From the experiences learned through implementing our prototype category searching system, we understand that we have four major problems to overcome. To achieve this, we introduce a new system we entitle as the part-of-speech tagging system. This system hedges on first eliminating all words that we are absolutely certain cannot be in any given category through the use of part-of-speech tagging. Once we are able to filter out various words, we then compare the remaining words against the chosen category to see if there are matches. For each match, we then confirm the context of the word by verifying the likelihood that the match satisfies the designated category. Below, we will detail our thought process and the implementation behind combating each of the four challenges in our system.

In the case of the first problem, reducing our search space, we decided to look into the English language, and in particular, the part of speech of a word. For the most part, we know that all categories primarily consist of nouns for matches. Nouns, as defined by LingualLinks, are words used to refer to people, places, things, ideas, or concepts. Hence, nouns can be defined as “subjects of the verb, objects of the verb, indirect object of the verb, or object of a preposition” [15]. Relating nouns back into our system, we can conclude that only nouns are explicit enough to distinguish a relation between any specified categories.

Of course, category word banks do not just comprise of single nouns as keys. In fact, noun phrases are just as prevalent. LingualLinks defines noun phrases as a phrase that has a noun at its head. Basically, a noun phrase consists of modifiers such as adjectives and articles that describe a noun in question [16]. As a result, we decided to implement a custom part-of-speech regular expression grammar that looks for specific combinations of part-of-speech tags to form possible noun phrases in addition to finding all possible noun types. These noun phrases will then be queried against the designated category. As a result, by looking only for nouns and noun phrases, we significantly reduce our search space.

Nonetheless, another problem arises: the possibility that a subset of the original noun phrase is found in the given category, whereas the original noun phrase is rejected. This segues nicely to our second issue: increasing the accuracy of our category search system. Situations like the one above occur because each category can only contain so many variations of the same word or word phrase. If the nouns or noun phrases found in a document do not match explicitly with those in the chosen category, then no match will be signaled.

To illustrate such an example let us use the noun phrase, “The New York Giants.” In this case, the chances of that phrase being found in the sports category may not be as probable as the noun phrase, “New York Giants.” Mainly, it is inefficient and naïve to store all possible variations that noun phrases can partake. In our situation, while the noun phrase, “The New York Giants” is perfectly acceptable, it is less plausible for that phrase to appear in documents versus the noun phrase, “New York Giants.” Therefore, to handle such cases, our system ends up breaking down each noun phrase by the positions of each non-noun word. For the noun phrase, “The New York Giants,” the “The” will be eliminated from it, leaving behind just the noun phrase “New York Giants.” Now, the system will begin checking the newly formed noun phrase to distinguish if it is indeed a match in the chosen category; it should be in our example. However, if it is not a match, the noun phrase decomposition will continue by removing the leftmost noun from the noun phrase. In this state, we will have two new noun phrases, “New” and “York Giants.” Obviously, there will be no matches, and our system

will continue removing the leftmost noun and appending that to the previously discarded leftmost nouns. Thus, two new noun phrases are created: “New York” and “Giants.”

Once the chosen category confirms that there is a match, our system then begins to determine the context of the matching word or words – our third challenge. Although it might not seem obvious at first, the context of each potential match is important in determining if that word or word phrase deserves to be classified under the specified category. Without any checks in place, false positives will be a common occurrence throughout any category search. To combat this, our system employs specific categorical related context search algorithms to reduce false positives.

One such check is to only scan for nouns or noun phrases that have the first letter of their initial word capitalized. This assumption can be inferred because, for the most part, the keywords that our category word banks contain are proper nouns, which are nouns that represent a unique entity, and, by default, are usually capitalized. This mechanism is especially useful in reducing false positives found in our geographical locations category, since numerous nouns and noun phrases can be interpreted as a geographical location. For other categories though, we, by default, search case insensitively because there is a possibility that documents contain words specific to the chosen category that are all lowercased. Nevertheless, this is simply not feasible for the geographical locations category because of the sheer volume of keywords in that word bank.

Another mechanism that we devised is to filter out possible nouns or noun phrases that will be considered false positive matches under most circumstances for a given category. This filtering technique creates its own word bank of keywords that it deems as undesirable and ignores all matches that are found within it. Once again, this method is mainly intended for the geographical locations category since there are many words that can be considered a geographical location. For instance, the word “Is,” represents a populated place in Russia with a population size of 4,729 [4]. With these two schemes, our system reduces the number of false positive entries returned.

When designing our system, we did not intend to eliminate or reduce false positive entries. In fact, our objective is to ensure that there are no false negatives. Limiting the number of false positives turned out to be important, since they impacted

the number of total results found after each category search. Indeed, the number of false positives found will on occasion impact the readability and usability of the total match results found. With the notion that our system will require user interaction, we decided to compromise and not focus on completely eliminating all false positives because users will be able to remove entries that they consider not relevant to their designated category.

For our last issue, we spent a great deal of effort looking to have our system finish at a reasonable time frame. Previously we were able to isolate the bottleneck as the query to the relational database. There are various ways to handle this situation, which include optimizing the database with algorithms and/or hardware modifications or using another data structure to store the data. In our case, we were able to reduce the query time to virtually milliseconds through the introduction of dbm files. Dbm, short for database manager, is a way to store arbitrary data through the use of a single key. These files use hashing techniques to enable fast data lookups. Transitioning to dbm over relational databases became practical because of dbm's high-speed storage look up through predefined keys. Thus, this omits the overhead of connecting and preparing queries for a relational database. Likewise, relational databases offer countless additional features over dbm that might be helpful for other data sets, but are not necessary for our system. All we needed is to verify whether or not a word in a document is found within a designated category. Hence, the keys for our dbm files will be the key terms in our category word bank, while the value will be some array of data describing the key and serialized under JavaScript Object Notation (JSON).

By using dbm files, we noticed that it helped to keep the execution time down to a reasonable threshold of around 15 seconds for analyzing a standard HTML web page of size 20 KB. Although, this does not meet our desired target time of around 5 seconds, it is a remarkable improvement over our previous brute force implementation. This reduction in execution time can be mostly attributed to our decision in using dbm files as the storage format for all of our categories.

After painting an overview of how our system handles the various problems identified in our brute force approach, we will now delve into more detail behind the

implementation of our system architecture. Specifically, we will talk about the process with which or system handles full text category searching.

3. SYSTEM DESIGN AND DESCRIPTION

Our full text category searching system is a web-based system that takes in two parameters: the desired category that the user wishes to find matches under and a URL that specifies either the location of a text file, the location of a HTML file, or the link to a web page. Each request is validated by our web server before being passed to our system for processing. The results will be outputted back to the user through the web server.

3.1 System Overview

Our system is primarily designed under the Python programming language and utilizes two Python frameworks: Beautiful Soup and Natural Language Toolkit (NLTK). It consists of four major components: categorization, parse, tag and interpretation. A detailed system diagram can be seen in Figure A.1.

3.2 Web Server

Before our system can begin executing the full text category search request, we have a web server that handles all incoming and outgoing traffic. The web server is as an abstraction layer used to separate a user's request from our category searching system. By adding this layer, we will use it to validate each category search request before passing it to our system. After every successful call, our web server will generate a web page with the resulting output obtained from our system.

3.3 Categorization

The categorization stage allows users the chance to either decide the category they wish to categorize a given text document under or an opportunity to create their own category word bank.

In the former case, this phase will determine which category-related dbm files are necessary for our category search. It will then use them as the category keyword bank for querying all possible entries found in the particular article.

In the latter case, a user can create a new category through our SQL template file as shown in Figure A.2. The template file defines the standards that users are required to conform to if they want to create a new category; in this case, users will have to define

three SQL tables: data, data column description, and alias. The data table allows users to define any number of related fields or attributes for each entry. Each field or attribute can be considered an association of components for each entry. The data column description table describes what each field or attribute in the data table represents. Here users are required to define three fields: column name, column type, and match level. The number of entries in this table should match the number of columns present in the data table. Finally, the alias table is used to add additional keywords that are related to a particular entry within the data table.

Once a user completes the SQL template, a new database will be created. We will then create the dbm file specific for the category by extracting all relevant information from that newly created database. Each dbm file will have their keys represented as all lowercased, in order to ensure case-insensitive matching. Once the dbm file is created, users can submit the category for use as a standard category. Moderators will have a choice in deciding where the category should be placed; all of the standardized categories are organized under a hierarchical classification structure.

3.4 Parse

Currently, our system supports three methods of input: plain text files, HTML styled files, and URL link to web pages. In the parse stage, the system needs to read in and break down the information that it is given; in this case, they are the visible words.

To achieve this task, we need to handle the plain text files and HTML styled files separately. Mainly, this is because a HTML styled file contains numerous extra words, such as HTML tags, that have no relevance to the visible text shown on each web page. The text that we are concerned about is found in between HTML tags; consequently, we will have to strip and break down all HTML tags. On the other hand plain text files contain nothing of that sort. All text present in a plain text file will contribute towards the actual text analyzed, and thus, no preprocessing work will need to be done.

3.4.1 Plain Text File

No special handling is done for this type of file; our system just reads in the plain text document and then passes it to the next stage of the process: tagging.

3.4.2 HTML File

Our system accepts either an actual URL link to a website or an HTML styled file for processing. Both methods are processed similarly; we will acquire the relevant HTML file through the wget command if the user decides to give us a URL link. Our system then reads in the file using a Python library known as Beautiful Soup. This framework has many features that make it easy for screen-scraping data off the web. These attributes include providing a way to navigate, search, and modify a parse tree of HTML tags. The framework automatically creates a tree-like structure modeling the HTML syntax of a particular web page when it is passed that web page's HTML code [1]. Thus, the framework allows us to extract whatever data we desire from a given web page; in our situation, we will like to grab all visible text shown on a particular web page.

To accomplish this task, we will begin by using the Beautiful Soup framework to eliminate all commented out tags from the system. This is to avoid those tags from being considered as part of the visible text collection seen on a rendered HTML file. We then filter out all visible content from the HTML file by removing all text residing between a set of HTML tags. For each word phrase extracted from a set of HTML tags, they are then stored into text blocks for analysis. To illustrate our point, consider this p tag example, $< p > \ Hello\ World < /p >$. We will use the framework to extract the phrase "Hello World." However, not all HTML tags represent information that is viewable on a web page. Tags like script, style, document, head, and title contribute nothing towards displaying the visible text and are duly ignored by our system. Though this technique is simple, it has the drawback of completely ignoring visible text that can be generated specifically through JavaScript functions.

Once all visible text is filtered out though, the system then begins to check for blocks that contain solely non-word characters. This is to prevent erroneous results as well as remove unnecessary processing. Next our system begins to look for any HTML entities that might be present and converts them into the Unicode equivalent. This will help our part-of-speech tagger make better guesses during the tagging process. At last, our system is now ready to pass each block of text for tagging.

3.5 Tag

In order to consider what words have a possibility of being a match in the chosen category, we need to find all nouns and noun phrases within a document. The best way to accomplish this will be to implement a part-of-speech tagger and a custom part-of-speech regular expression grammar tailored towards finding all nouns and noun phrases. This will work by first having a part-of-speech tagger tag the entire text in question and then using a regular expression grammar to locate all nouns and noun phrases. To accomplish this task, we use Natural Language Toolkit (NLTK): a Python framework that consists of many libraries used to aid in natural language processing and text analysis [9]. This bodes well for our system since the library has all of the tools necessary to implement a part-of-speech tagger as well as create a part-of-speech regular expression grammar.

We decided on using NLTK's default part-of-speech tagger as our system's tagger. NLTK's default tagger is a classifier based tagger trained on the PENN Treebank corpus, which is comprised of news related articles. Consequently, the tagger is considerably more accurate on documents that are closely related to news-styled articles like the New York Times or the Wall Street Journal [11].

To build our part-of-speech regular expression grammar we capitalized on NLTK's library of grammar based chunk parser known as RegexpParser. This parser allows users to set regular expression patterns in order to specify the behavior of it [14]. For our system, we defined our grammar to reexamine the already part-of-speech tagged text and find nouns and noun phrases within it. Figure A.3 will illustrate the grammar that we used to implement our system.

3.6 Interpretation

The interpretation stage is responsible for determining whether or not the nouns and noun phrases gathered during the tagging stage satisfy being categorized under the chosen category. It begins by first converting the noun or noun phrase to all lowercase in order to ensure case-insensitive matching. We then query it against the dbm files related to the chosen category. Only two possible outcomes can occur: there is a match or there is no match.

A match in a dbm file will indicate that the noun or noun phrase can potentially be considered a match under the chosen category. We then examine the context of the potential match to determine if it is a positive match or it is a false positive. At this stage, only the geographical locations category has any semblance of determining the context of a match. The geo-related category has a check that examines the first word of each noun or noun phrase and determines whether or not the first letter is capitalized, or all of the letters are capitalized. This helps to reduce false positives for the geo-related category since geographic locations can take names of very common words – like Is, Russia – which makes it hard to inspect all case-insensitive words. Thus, by only looking at capitalized words, we reduced the total number of matches found in our system at the cost of potentially having false negatives. However, the chances of a geographical location not being capitalized are very slim. Moreover, the geographical locations category also has its own filter set that will ignore all matches that are found in it. We can continually add words to this filter that we believe are false positives, thus reducing the total number of false positives found. Currently, our filter contains a random assortment of words that we found unlikely to be a geographical reference when training our filter.

Alternatively, if there is no match, there are two things that can occur depending if the item that is queried is a noun or a noun phrase. If the item queried was a noun, it will be discarded and the next noun or noun phrase will be selected to continue the process. However, if the item queried were a noun phrase, we will begin breaking down the noun phrase by the positions of each non-noun word. Subsequently, if the broken down noun phrase(s) are not found in the chosen category, then the noun phrase decomposition will continue by removing the leftmost noun from the noun phrase.

4. TEST SYSTEM

To verify the effectiveness of our category searching system, we needed to determine the rate that our system on average generates false positives and false negatives. Since we do not know if our category searching system locates all of the correct occurrences of keywords for a specified category, we decided to create a testing system to accomplish this task.

We ended up building a phrase matching system, which gathers exact word phrases from the given text and queries it against the selected category related dbm files. Our phrase matching scheme is constructed in Python and uses the same Python libraries as our part-of-speech tagging scheme. It even utilizes the same categorization, parsing, and interpretation phases as the part-of-speech tagging system. Though, quite evidently, the tagging phase is missing from this scheme and replaced with a phrase matching phase.

Basically, the test system begins by having a simple parser that parses the text document into a list of words. Each word is a sequence of non-whitespaces separated by two whitespace characters. These words are then assigned an offset value to determine their location within the given body of text. The test system then inspects each individual word, in the order they appear in the text, and creates variants of the word in question. This is done in two steps. First, we remove all characters that are not words, digits, commas, periods, dashes, slashes, ampersands, and apostrophes from the examined word. Second, we will iteratively remove all commas, periods, dashes, slashes, ampersands, and apostrophes from the word in question. Take for instance the word, “Memphis-Illinois.”. From our word variant algorithm, we will create the following combinations: “Memphis-Illinois.”, “Memphis-Illinois”, “Memphis”, “Illinois.”, and “Illinois”.

The crux of this system hinges on our implementation of a lookahead mechanism that peeks at a specified number of words in advance (we chose 4 words by default) and appends them one at a time to our given initial word. At each stage, we query that particular phrase against our designated category related dbm files to verify if there are any matches. The main reason for the mechanism is to find word phrases that contain more than one word as a match, like “New York Giants.” Ultimately, we settled at 4

lookahead words because we felt that typically, most category keywords will not exceed 5 words in length.

Our algorithm can be further illustrated below:

$For\ each\ word\ in\ offset\ N, we\ create\ 5\ word\ phrases\ such\ that$:

$word\ phrase\ \#1 \quad word\ at\ offset\ N$

$word\ phrase\ \#2 \quad word\ at\ offset\ N, N+1$

…

$word\ phrase\ \#5 \quad word\ at\ offset\ N, N+1, N+2, \ldots, N+4$

Generally, we will only have to create 5 words phrases for each word. However, there is the possibility of words having a variant. In that case, for each variant, we will follow the algorithm detailed above.

To illustrate such an example, take a look at the following:

$Memphis - Illinois\ is\ the\ game\ of\ the\ week.$

$Original\ Word: Memphis - Illinois$

$Memphis - Illinois$

$Memphis - Illinois\ is$

$Memphis - Illinois\ is\ the$

…

$First\ Variant: Memphis$

$Memphis$

$Memphis\ is$

$Memphis\ is\ the$

…

$Second\ Variant: Illinois$

$Illinois$

$Illinois\ is$

$Illinois\ is\ the$

…

By applying the algorithm to each new word variation, we ensure that we will not miss any category keyword matches. We even ensure that we will not generate duplicate matches since for each match that is found using this method, we will only accept matches that either have the most offset values or the same offset values, but different match value. So, if we find the match “Dallas” and “Dallas Cowboys”, we will only select the “Dallas Cowboys”. And if there are more than one match with the same match value and offset value, we will only accept one of those.

5. RESULTS AND FINDINGS

We conducted testing for our part-of-speech tagging system and phrase matching system against 40 different web pages. Each web page was chosen specifically to match the following guidelines: no two web pages share the same domain name and the articles selected from each domain name are picked at random (the articles themselves might not have any relevance to the category that we are searching under). Both of our systems analyze each of the 40 web pages against our four arbitrarily created categories: top 100 US colleges as ranked by Newsweek, Fortune 100 companies, geographical locations, and US professional sports teams.

5.1 Performance Metrics

When selecting our testing corpus, we decided on evaluating a diverse selection of web pages to observe the effectiveness of our category searching system. We wanted to see how flexible and efficient our system was at finding matches and determining whether or not they are related to the chosen category. Moreover, we were interested at how well our system scales to handling various web page sizes.

In the case of our four arbitrarily created categories, they were handcrafted to evaluate two aspects of our system. First, we wanted them to simulate a variety of possible keyword formats that include: people, places, and objects. Second, we wanted to test how well our category databases hold up to categories that contain a wealth of information. For instance, our geography category consists of almost 500 MB worth of geography-related data.

Our main goal then is to evaluate the number of exact matches located within the document that are both found in the chosen category and have the correct context for being considered a good match. When evaluating a good match, we also have to take into consideration the number of false positives found; hence, to measure our system's effectiveness on returning false positives, we calculate the percentage of false positives to good matches. Thus, we would want to minimize this percentage by reducing the number of false positives returned by the system. False negatives are also another metric that we assess during our testing. Any false negatives hurt the accuracy and credibility of our category searching system; consequently, the lower the number the better. Finally,

we measure the execution time of our system. Execution time is an important indicator of how effective our system is at delivering the results to our users. Since our system is intended to be web-based, reducing the execution time is important in making the system a viable solution for public use.

5.2 Algorithm Performance

From the aggregate results and average execution times obtained in Table B.2 and Table B.3 respectively, we notice that both systems performed reasonably well in all of our tests. However, a slight edge goes to our phrase matching system, since it performs better in almost all of the measured metrics except false positives. We will detail these observations below.

False negatives did not occur in our phrase matching system, but were occasionally present in our part-of-speech tagging system as evident in Table B.2. Two issues contribute towards this problem. First, our system's custom part-of-speech regular expression grammar does not capture all of the possible noun and noun phrase grammars that can be present. For example, our custom part-of-speech regular expression grammar could not locate the phrase "AT&T" because our custom grammar does not contain the sequence of "<IN>+ <CC>+ <NNP>+", which is what our tagger ended up breaking down the noun phrase into: ('AT', 'IN'), ('&', 'CC'), ('T', 'NNP). Second, the part-of-speech tagging system is not always accurate in guessing the correct part-of-speech for some words. These misses can be attributed to the corpus that the NLTK default part-of-speech tagger is trained against. To illustrate this, take the same phrase "AT&T". Our system treats the ('AT', 'IN') component as a separate element by tagging it as a preposition, rather than treating it as a proper noun. Therefore, our system has trouble distinguishing that the "AT" portion is part of the single word, "AT&T", and not an entirely new separate entity.

In the case of our phrase matching system, there are no false negatives found during our testing. This can be attributed to two factors. First, the system separates words based on whitespace characters; thus, punctuation will still be attached to the given word. Examples of this include the phrases "AT&T" or "Michigan-Illinois". Second, the system breaks down each word by its punctuation. This allows us to capture all of the possible variations that the word in question can resemble. In the case of the

phrase “Michigan-Illinois”, our system will allow us to strip it down and locate the entries of “Michigan” and “Illinois”.

False positives, on the other hand, were awfully prevalent in both systems. Once again, context is a major contributing factor towards this high rate. Geographical locations such as “Chicago” can be found related to many different categories such as sports, Fortune 100 companies, and even US colleges. In the case of the sports category, “Chicago” might reference part of a team name, like the “Chicago Bulls” or “Chicago Bears”. For Fortune 100 companies, “Chicago” might be used to distinguish Boeing’s company headquarters. And in the case of US colleges, “Chicago” is another name for the “University of Chicago”. In general, there is not much one can do to effectively eliminate the ambiguity of the words and phrases.

When examining our last and final metric, execution time, we can observe that there is a noticeable difference in execution times when comparing the two systems. From Table B.3, we notice that our testing system is outperforming our part-of-speech tagging system by almost a 3 to 1 margin. We were able to deduce that our part-of-speech tagger was the culprit. This is likely due to the inherent overhead in understanding and then subsequently tagging all of the words in a document. In the phrase matching system, on the other hand, it only incurs a key look up penalty even though it examines more than 5 times as many words compared to our part-of-speech tagging system.

After analyzing the results from the two systems, we can conclude that our testing system actually outperforms our part-of-speech tagging system in terms of speed and accuracy, but has a higher rate of false positives. As a result, we believe that our phrase matching system is the better implementation.

6. CONCLUSION AND FUTURE WORKS

In this paper we present a new implementation of a full text category searching system. We describe our implementation and design decisions behind the system as well as the testing system used to verify our category searching system's effectiveness. After running tests against both systems, we notice that our testing system actually performs better than our category searching system in terms of speed and accuracy. Ultimately, both systems do a reasonable job in reducing false negatives and false positives.

There are several directions that we want our system to accomplish for future research. First, we would like to incorporate more context searching algorithms to reduce the number of false positives returned. Second, we would like to introduce a confidence level metric that will determine the likelihood of a match found being an actual match and not a false positive.

Context searching algorithms for eliminating the ambiguity of matches is lacking in our system. While we do have checks such as looking at only capital words or setting up a filter, they do not entirely reduce all of the false positives occurring. That is why we would like to force all keywords defined in our categories to be set as either a primary or secondary keyword to indicate the likelihood that all matches are actual matches.

Primary keywords will be the keywords that undoubtedly indicate that a word is related to a chosen category. An example of this will be "New York Giants," which is a registered trademark of the New York Football Giants, Inc. and relates specifically to the National Football League (NFL). When a word matches a primary keyword, the document in question has a high probability of being related to the chosen category. As a result, it looks more likely that the other matches in doubt are category matches.

Secondary keywords will be the keywords that indicate that the chosen word could be related to the designated category, but that it is not necessarily the case. For instance, the phrase "New York," will not have enough information to indicate that it is a match under the sports category. Conversely, if a match with a primary keyword like, "New York Giants" were to be found, it will be sufficient to indicate that other secondary keywords found, such as "New York" and "Giants," will be considered actual matches since we are given a context clue that the text is discussing sports related topics.

Therefore, if we cannot find a primary keyword in the given text, we can eliminate all of the secondary keywords found.

Still, there are cases where a document might not contain any primary keywords, but it is actually related to the chosen category. In that situation, eliminating all potential false positives is probably not the right approach. Thus, introducing a confidence level metric for each match keyword in the returned result would be ideal. The confidence level could be introduced using the same primary and secondary keywords as specified earlier. But instead of automatically accepting or throwing away matches because of primary keywords being found, we set numerical values to indicate the likelihood that a match is a genuine match. For instance, we can say that the more times a certain keyword is found in the document, the likelier it is for it being a match. Though the use of confidence levels do not reduce the number of false positives in the returned results, we can reduce false positives from appearing at top of the returned results by sorting the matched keywords in descending order of confidence levels.

Results indicate that our full text category searching system has plenty of potential in being a useful tool. While there is still more work that needs to be done, we have high hopes in the future usability of this system.

REFERENCES

[1] Beautiful Soup. *Beautiful Soup Home Page*. Crummy, n.d. Web. Retrieved March 14, 2012: http://www.crummy.com/software/BeautifulSoup/.

[2] Chekuri, C., Goldwasser, M.H., Raghavan, P., and Upfal, E. Web Search Using Automatic Classification. In *Proceedings of the 6th International World Wide Web Conference*. (Santa Clara, CA, 1997).

[3] Devi, M.I., Rajaram, R., and Selvakuberan, K. Generating best features for web page classification. *Webology*. 5 (1), Article 52. Retrieved March 14, 2012: http://www.webology.org/2008/v5n1/a52.html.

[4] GeoNames Fulltextsearch : is. *GeoNames Search*. GeoNames, n.d. Web. Retrieved March 14, 2012: http://www.geonames.org/search.html?q=is&country=.

[5] Hearst, M.A. Clustering Versus Faceted Categories for Information Exploration. *Communications of the ACM*. 41 (4). 59-61.

[6] Iwayama, M. and Tokunaga, T. Cluster-Based Text Categorization: A Comparison of Category Search Strategies. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. (Seattle, WA, 1995).

[7] Kohlschütter, C., Chirita, P.-A., and Nejdl, W. Using Link Analysis to Identify Aspects in Faceted Web Search." In *ACM SIGIR '06 Workshop on Faceted Search*. (Seattle, WA, 2006).

[8] Koren, J., Zhang, Z., and Liu, X. Personalized Interactive Faceted Search. In *Proceedings of the 17th International Conference on the World Wide Web (WWW '08).* (Beijing, CN, 2008).

[9] Natural Language Toolkit. *Natural Language Toolkit Home Page*. NLTK, n.d. Web. Retrieved March 14, 2012: http://www.nltk.org/.

[10] Qi, X. and Davison, B.D. Web Page Classification: Features and Algorithms. *ACM Computing Surveys*. 41 (2).

[11] Tagging, Chunking & Named Entity Recognition with NLTK. *Natural Language Processing APIs and Python NLTK Demos*. Text-Processing, n.d. Web. Retrieved March 14, 2012: http://text-processing.com/demo/tag/.

[12] Tomás, D. and Vicedo, J.L. Multiple-Taxonomy Question Classification for Category Search on Faceted Information. In *Proceedings of the 10th International Conference on Text, Speech and Dialog*. (Pilsen, CZ, 2007).

[13] Tunkelang, D. Dynamic Category Sets: An Approach for Faceted Search. In *ACM SIGIR '06 Workshop on Faceted Search*. (Seattle, WA, 2006).

[14] type RegexpParser. *Package nltk*. NLTK Google Code, n.d. Web. Retrieved March 14, 2012: http://nltk.googlecode.com/svn/trunk/doc/api/nltk.chunk.regexp.RegexpParser-class.html.

[15] What is a noun?. *LinguaLinks*. SIL International, 05 Jan 2004. Web. Retrieved March 14, 2012: http://www.sil.org/linguistics/GlossaryOfLinguisticTerms/WhatIsANoun.htm.

[16] What is a noun phrase?. *LinguaLinks*. SIL International, 05 Jan 2004. Web. Retrieved March 14, 2012: http://www.sil.org/linguistics/GlossaryOfLinguisticTerms/WhatIsANounPhrase.htm.

Appendix A

FIGURES REFERENCED

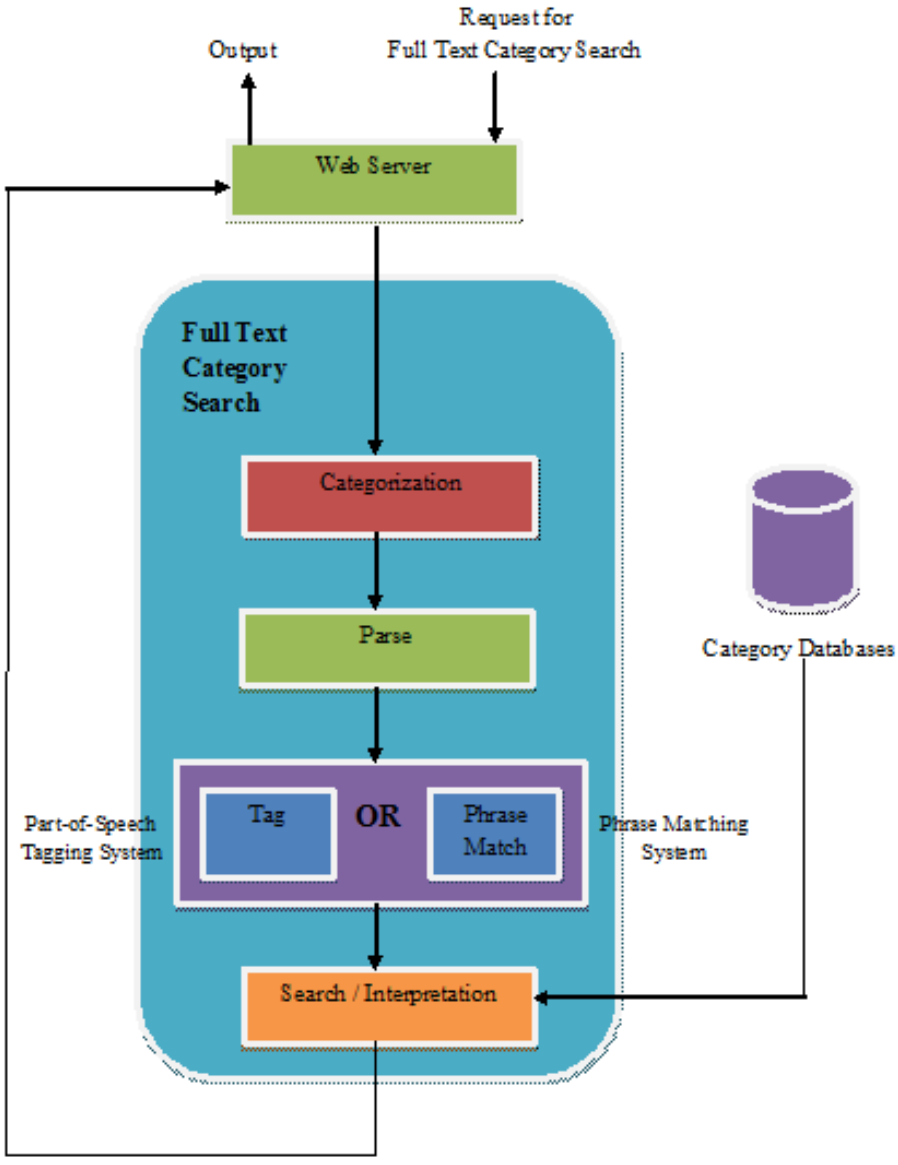

Figure A.1: Full Text Category Search System

```sql
CREATE TABLE IF NOT EXISTS data (
  col1 INTEGER PRIMARY KEY,
  col2 varchar(200) NOT NULL,
  col3 varchar(100) NOT NULL,
  col4 varchar(200) NOT NULL,
  col5 double NOT NULL,
  col6 double NOT NULL
);

CREATE TABLE IF NOT EXISTS data_column_description (
  id INTEGER PRIMARY KEY,
  col_name varchar(200) NOT NULL,
  col_type varchar(50) NOT NULL,
  match_level INTEGER NOT NULL
);

INSERT INTO data_column_description (id, col_name, col_type, match_level) VALUES (1, 'Id', 'id', 0);
INSERT INTO data_column_description (id, col_name, col_type, match_level) VALUES (2, 'Team Name', 'sports', 1);
INSERT INTO data_column_description (id, col_name, col_type, match_level) VALUES (3, 'League', 'data', 2);
INSERT INTO data_column_description (id, col_name, col_type, match_level) VALUES (4, 'Stadium', 'data', 2);
INSERT INTO data_column_description (id, col_name, col_type, match_level) VALUES (5, 'Latitude', 'latitude', 2);
INSERT INTO data_column_description (id, col_name, col_type, match_level) VALUES (6, 'Longitude', 'longitude', 2);

CREATE TABLE IF NOT EXISTS alias (
  id INTEGER PRIMARY KEY,
  data_id INTEGER NOT NULL,
  alias_name varchar(200) NOT NULL,
  FOREIGN KEY (data_id) REFERENCES data(id)
);

INSERT INTO data (col2, col3, col4, col5, col6) VALUES ('Dallas Cowboys', 'NFL', 'Cowboys Stadium', '32.747778', '-97.092778');
INSERT INTO alias (data_id, alias_name) VALUES (17, 'America''s Team');
```

Figure A.2: SQLite Template Format

```
grammar = """
            NP:
                {<NNP>+ <IN> <DT>? <NNP>* <NNPS>*}
                {<NNP>+ <.> <NNP|.>*}
                {<NNP>+ <CC> <NNP>+ <NNPS>*}
                {<NNP>+ <CC> <NNPS>+ <NNP>*}
                {<NNP>+ <NN>+ <NNS>*}
                {<NNP>+ <NNS>+ <NN>*}
                {<NNP>+ <NNPS>+ <IN>* <NNP>*}
                {<NNPS>+ <NNP>+ <NNPS>*}
                {<NNP>+}
                {<NNPS>+}
                {<NN>+}
                {<NNS>+}
"""
```

Figure A.3: Regular Expression Grammar used to locate Nouns and Noun Phrases

Appendix B

TABLES REFERENCED

Table B.1: List of Websites used for Analysis

Site #	Website URL
1	http://www.tsn.ca/blogs/darren_dreger/?id=389863
2	http://www.nytimes.com/2012/03/01/technology/impatient-web-users-flee-slow-loading-sites.html
3	http://online.wsj.com/article/SB10001424052970203986604577255532947217336.html
4	http://espn.go.com/mens-college-basketball/story/_/id/7635525/duke-blue-devils-north-carolina-tar-heels-highlights-weekend-games-college-basketball
5	http://www.al.com/sports/index.ssf/2012/03/sec_should_make_move_to_nine-g.html
6	http://finance.yahoo.com/news/if-you-re-using--password1---change-it--now-.html
7	http://www.defendingbigd.com/2012/3/4/2845188/dallas-stars-calgary-flames-recap-shootout-3-2-win-nhl-standings-eriksson
8	http://www.ft.com/intl/cms/s/0/3eef0bc4-6f73-11e1-9c57-00144feab49a.html
9	http://stackoverflow.com/questions/816285/where-is-pythons-best-ascii-for-this-unicode-database
10	http://news.sciencemag.org/sciencenow/2012/03/examining-his-own-body-stanford-.html
11	http://www.extremetech.com/extreme/122693-scientists-create-graphene-from-scratch-heralds-new-age-of-designer-materials
12	http://news.stanford.edu/news/2012/march/molecules-designer-electrons-031412.html
13	http://www.abc.net.au/news/2012-03-18/thorpe-london-hopes-end/3896618
14	http://phx.corporate-ir.net/phoenix.zhtml?c=176060&p=irol-newsArticle
15	http://www.spacedaily.com/reports/Russia_May_Sink_Satellite_Salvage_Plan_For_Antarctic_Internet_Connection_999.html
16	http://news.discovery.com/human/pink-slime-psychology-120319.html
17	http://www.bbc.co.uk/news/entertainment-arts-16424579
18	http://singularityhub.com/2012/03/18/8200-strong-researchers-band-together-to-force-science-journals-to-open-access/
19	http://techlaw.justia.com/2012/03/16/samsung-sued-over-emoticon-patent-but-not-by-apple/
20	http://www.techdirt.com/articles/20120319/01045818152/pirate-bay-claims-its-going-to-host-site-via-drones-flying-over-international-waters.shtml
21	http://www.infoworld.com/t/mobile-technology/android-leads-asian-smartphone-invasion-188927
22	http://www.securelist.com/en/blog/687/A_unique_fileless_bot_attacks_news_site_visitors
23	http://www.gizmag.com/subaru-production-gt-86-brz/21866/
24	http://web.mit.edu/press/2012/biplane-to-break-the-sound-barrier.html
25	http://www.techweekeurope.co.uk/news/google-data-centre-cooled-by-bathtub-water-67756

26	http://arstechnica.com/tech-policy/news/2011/07/what-the-1930s-fashion-industry-means-for-big-contents-six-strikes-plan.ars
27	http://www.cumc.columbia.edu/news-room/2012/03/%E2%80%9Cpersonalized-immune%E2%80%9D-mouse-offers-new-tool-for-studying-autoimmune-diseases/
28	http://torrentfreak.com/google-defends-hotfile-and-megaupload-in-court-120319/
29	http://www.datacenterknowledge.com/archives/2012/03/19/is-solar-power-for-data-centers-a-bad-idea/
30	http://www.vancouversun.com/sports/roundup+Pouliot+scores+twice+Bruins+rout+Maple+Leafs/6326939/story.html
31	http://www.latimes.com/sports/la-sp-elliott-nhl-20120320,0,7549138.column
32	http://www.cbssports.com/collegebasketball/story/17895772/sweet-16-look-ahead-marshalls-availability-will-be-biggest-story
33	http://mlb.mlb.com/news/article.jsp?ymd=20120319&content_id=27426592
34	http://www.washingtonpost.com/sports/nationals/mets-repay-loans-to-mlb-25-million-and-bank-of-america-40-million/2012/03/19/gIQAoZcJOS_story.html
35	http://wwos.ninemsn.com.au/article.aspx?id=8437878
36	http://www.cbsnews.com/8301-202_162-57399610/sweden-moving-towards-cashless-economy/
37	http://abcnews.go.com/blogs/business/2012/03/highest-gas-price-recorded-for-march/
38	http://insidemovies.ew.com/2012/03/19/john-carter-disney-flop/
39	http://www.economist.com/node/21550256
40	http://www.telegraph.co.uk/news/newstopics/howaboutthat/9154181/Little-known-artist-chosen-as-artist-in-residence-at-The-Savoy.html

Table B.2: Aggregate Results

Category	System	Matches Found	Correct Matches	False Negatives *	False Positives **
Top 100 US Colleges	POS-Tagging	232	160	0 (0 %)	72 (45 %)
Fortune 100	POS-Tagging	312	128	18 (14.06 %)	184 (143.75 %)
Geography	POS-Tagging	1886	542	3 (0.55 %)	1334 (246.13 %)
Sports	POS-Tagging	313	173	0 (0 %)	140 (80.92 %)
Top 100 US Colleges	Phrase Matching	237	162	0 (0 %)	75 (46.3 %)
Fortune 100	Phrase Matching	339	146	0 (0 %)	193 (132.19 %)
Geography	Phrase Matching	1974	546	0 (0 %)	1433 (262.45 %)
Sports	Phrase Matching	340	172	0 (0 %)	168 (97.67 %)

* False Negative % = False Negatives / Correct Matches

** False Positive % = False Positives / Correct Matches

Table B.3: Average Execution Time

Category	System	Avg Text Size (KB) *	Avg Elapsed Time (s)
Top 100 US Colleges	POS-Tagging	4.7	8.45
Fortune 100	POS-Tagging	4.7	8.48
Geography	POS-Tagging	4.7	8.73
Sports	POS-Tagging	4.7	8.47
Top 100 US Colleges	Phrase Matching	4.7	3.03
Fortune 100	Phrase Matching	4.7	3.04
Geography	Phrase Matching	4.7	1.58
Sports	Phrase Matching	4.7	3.19

* Average size of the visible text that was extracted from the 40 websites

Table B.4: Results for Top 100 US Colleges as evaluated by POS-Tagging

Site #	Text Size (KB)	Elapsed Time (s)	Matches Found	Correct Matches	False Negatives	False Positives
1	6.9	9.82	2	0	0	2
2	3.9	8.21	3	1	1	2
3	6.3	9.08	1	0	0	1
4	8.8	10.61	56	54	0	2
5	2.8	7.51	9	9	2	0
6	3.6	7.81	1	0	0	1
7	4.5	8.23	0	0	0	0
8	6.6	9.02	3	1	0	2
9	3.6	7.87	0	0	0	0
10	5.8	8.72	7	4	0	3
11	3.3	7.53	4	4	0	0
12	3.8	7.87	6	6	0	0
13	3.9	8.12	0	0	0	0
14	5.9	9.26	3	0	0	3
15	6.3	9.51	0	0	0	0
16	3.1	7.75	0	0	0	0
17	2.6	7.56	0	0	0	0
18	5.8	8.76	1	1	0	0
19	2.1	7.38	2	0	0	2
20	2.7	7.44	0	0	0	0
21	5.2	8.69	2	0	0	2
22	8.1	9.67	4	0	0	4
23	2.9	7.69	5	0	0	5
24	5	8.38	4	4	0	0
25	3.4	7.8	9	0	0	9
26	6.4	8.99	5	1	0	4
27	5.8	9.01	14	11	0	3
28	5.3	8.46	6	0	0	6
29	3.6	7.86	1	0	0	1
30	5.1	8.74	0	0	0	0
31	5	8.65	5	0	0	5
32	6.9	10.21	67	64	0	3
33	6.2	9.43	3	0	0	3
34	1.4	6.92	0	0	0	0
35	1.9	7.17	1	0	0	1
36	6.1	8.91	0	0	0	0
37	0.9	6.66	0	0	0	0
38	2.1	7.63	1	0	0	1

39	10.8	11.29	7	0	0	7
40	3.5	7.93	0	0	0	0
Total	-	-	232	160	0	72

Table B.5: Results for Fortune 100 Companies as evaluated by POS-Tagging

Site #	Text Size (KB)	Elapsed Time (s)	Matches Found	Correct Matches	False Negatives	False Positives
1	6.9	9.86	10	0	0	10
2	3.9	8.25	9	7	0	2
3	6.3	9.13	9	7	17	2
4	8.8	10.61	8	0	0	8
5	2.8	7.51	0	0	0	0
6	3.6	8.28	8	6	0	2
7	4.5	8.25	7	0	0	7
8	6.6	9.03	16	6	0	10
9	3.6	7.85	5	1	0	4
10	5.8	8.78	7	0	0	7
11	3.3	7.53	0	0	0	0
12	3.8	7.88	2	0	0	2
13	3.9	8.11	4	0	0	4
14	5.9	9.28	41	36	0	5
15	6.3	9.51	1	0	0	1
16	3.1	7.74	1	1	0	0
17	2.6	7.57	3	0	0	3
18	5.8	8.76	2	0	0	2
19	2.1	7.36	5	4	0	1
20	2.7	7.44	0	0	0	0
21	5.2	8.71	3	2	0	1
22	8.1	9.72	1	1	0	0
23	2.9	7.69	1	0	0	1
24	5	8.35	1	0	0	1
25	3.4	7.76	22	16	0	6
26	6.4	9	5	0	0	5
27	5.8	8.99	9	0	0	9
28	5.3	8.44	22	22	0	0
29	3.6	7.94	11	10	0	1
30	5.1	8.76	19	0	0	19
31	5	8.68	17	0	0	17

32	6.9	10.21	24	0	0	24
33	6.2	9.41	6	0	0	6
34	1.4	6.92	4	1	1	3
35	1.9	7.15	1	0	0	1
36	6.1	8.94	0	0	0	0
37	0.9	6.67	0	0	0	0
38	2.1	7.66	15	7	0	8
39	10.8	11.29	11	1	0	10
40	3.5	7.94	2	0	0	2
Total	-	-	312	128	18	184

Table B.6: Results for Geographical Locations as evaluated by POS-Tagging

Site #	Text Size (KB)	Elapsed Time (s)	Matches Found	Correct Matches	False Negatives	False Positives
1	6.9	11.76	57	15	0	42
2	3.9	8.09	17	3	0	4
3	6.3	8.84	18	5	0	13
4	8.8	11.04	89	47	0	42
5	2.8	7.42	13	11	0	2
6	3.6	8.32	2	1	0	1
7	4.5	8.05	14	13	0	1
8	6.6	9.29	58	20	1	38
9	3.6	7.66	14	0	0	14
10	5.8	8.93	47	14	0	33
11	3.3	7.37	9	4	0	5
12	3.8	8.24	37	8	0	29
13	3.9	8.27	47	10	0	37
14	5.9	10.07	120	45	2	75
15	6.3	9.76	52	10	0	42
16	3.1	7.75	23	0	0	23
17	2.6	7.81	46	5	0	41
18	5.8	8.8	38	2	0	36
19	2.1	7.31	39	3	0	36
20	2.7	7.6	11	0	0	11
21	5.2	8.67	55	25	0	30
22	8.1	9.38	30	0	0	30
23	2.9	7.73	59	10	0	49
24	5	8.26	36	6	0	30

25	3.4	7.89	28	13	0	15
26	6.4	8.99	59	7	0	52
27	5.8	9.21	62	17	0	45
28	5.3	8.33	10	3	0	7
29	3.6	8.03	35	4	0	31
30	5.1	9.79	120	31	0	89
31	5	10.9	70	10	0	60
32	6.9	11.64	210	102	0	108
33	6.2	10.2	108	11	0	97
34	1.4	7.07	18	4	0	14
35	1.9	7.22	29	16	0	13
36	6.1	8.8	59	22	0	37
37	0.9	6.69	7	2	0	5
38	2.1	8.14	33	0	0	33
39	10.8	11.49	56	27	0	29
40	3.5	8.35	51	16	0	35
Total	-	-	1886	542	3	1334

Table B.7: Results for Sports as evaluated by POS-Tagging

Site #	Text Size (KB)	Elapsed Time (s)	Matches Found	Correct Matches	False Negatives	False Positives
1	6.9	10.03	33	31	0	2
2	3.9	8.21	2	0	0	2
3	6.3	9.15	2	0	0	2
4	8.8	10.61	15	0	0	15
5	2.8	7.49	2	0	0	2
6	3.6	7.83	1	0	0	1
7	4.5	8.33	34	33	0	1
8	6.6	9.04	8	0	0	8
9	3.6	7.86	0	0	0	0
10	5.8	8.71	3	0	0	3
11	3.3	7.54	0	0	0	0
12	3.8	7.99	0	0	0	0
13	3.9	8.1	4	0	0	4
14	5.9	9.32	16	0	0	16
15	6.3	9.61	2	0	0	2
16	3.1	7.77	1	0	0	1
17	2.6	7.56	1	0	0	1

18	5.8	8.72	1	0	0	1
19	2.1	7.35	1	0	0	1
20	2.7	7.45	0	0	0	0
21	5.2	8.72	6	0	0	6
22	8.1	9.64	0	0	0	0
23	2.9	7.69	0	0	0	0
24	5	8.39	10	0	0	10
25	3.4	7.77	2	0	0	2
26	6.4	8.97	1	0	0	1
27	5.8	9	4	0	0	4
28	5.3	8.44	1	0	0	1
29	3.6	7.9	3	0	0	3
30	5.1	8.78	48	48	0	0
31	5	8.73	26	26	0	0
32	6.9	10.21	31	1	0	30
33	6.2	9.48	25	25	0	0
34	1.4	7.03	9	9	0	0
35	1.9	7.16	0	0	0	0
36	6.1	8.96	9	0	0	9
37	0.9	6.67	0	0	0	0
38	2.1	7.18	0	0	0	0
39	10.8	11.27	12	0	0	12
40	3.5	7.9	0	0	0	0
Total	-	-	313	173	0	140

Table B.8: Results for Top 100 US Colleges as evaluated by Phrase Matching

Site #	Text Size (KB)	Elapsed Time (s)	Matches Found	Correct Matches	False Negatives	False Positives
1	6.9	4.47	2	0	0	2
2	3.9	2.55	4	2	0	2
3	6.3	3.88	1	0	0	1
4	8.8	5.67	56	54	0	2
5	2.8	1.95	10	10	0	0
6	3.6	2.17	1	0	0	1
7	4.5	2.96	0	0	0	0
8	6.6	3.95	3	1	0	2
9	3.6	2.16	0	0	0	0
10	5.8	3.45	7	4	0	3

11	3.3	2.24	4	4	0	0
12	3.8	2.29	6	6	0	0
13	3.9	2.7	0	0	0	0
14	5.9	4.92	3	0	0	3
15	6.3	3.62	0	0	0	0
16	3.1	2.16	1	0	0	1
17	2.6	1.87	0	0	0	0
18	5.8	3.55	1	1	0	0
19	2.1	1.6	2	0	0	2
20	2.7	1.89	0	0	0	0
21	5.2	3.3	2	0	0	2
22	8.1	4.69	4	0	0	4
23	2.9	2.04	5	0	0	5
24	5	3.23	4	4	0	0
25	3.4	2.29	9	0	0	9
26	6.4	3.65	5	1	0	4
27	5.8	3.38	14	11	0	3
28	5.3	3.29	6	0	0	6
29	3.6	2.37	1	0	0	1
30	5.1	3.37	0	0	0	0
31	5	3.27	5	0	0	5
32	6.9	4.43	67	64	0	3
33	6.2	4.06	3	0	0	3
34	1.4	1.15	0	0	0	0
35	1.9	1.42	1	0	0	1
36	6.1	3.75	1	0	0	1
37	0.9	0.9	0	0	0	0
38	2.1	1.56	1	0	0	1
39	10.8	6.41	8	0	0	8
40	3.5	2.39	0	0	0	0
Total	-	-	237	162	0	75

Table B.9: Results for Fortune 100 Companies as evaluated by Phrase Matching

Site #	Text Size (KB)	Elapsed Time (s)	Matches Found	Correct Matches	False Negatives	False Positives
1	6.9	4.4	11	0	0	11
2	3.9	2.56	9	7	0	2
3	6.3	3.89	26	24	0	2

4	8.8	5.65	9	0	0	9
5	2.8	1.96	0	0	0	0
6	3.6	2.18	8	6	0	2
7	4.5	2.95	7	0	0	7
8	6.6	3.95	16	6	0	10
9	3.6	2.18	5	1	0	4
10	5.8	3.45	7	0	0	7
11	3.3	2.24	1	0	0	1
12	3.8	2.29	2	0	0	2
13	3.9	2.7	4	0	0	4
14	5.9	5.01	42	36	0	6
15	6.3	3.64	1	0	0	1
16	3.1	2.16	1	1	0	0
17	2.6	1.88	3	0	0	3
18	5.8	3.89	4	0	0	4
19	2.1	1.61	6	4	0	2
20	2.7	1.9	0	0	0	0
21	5.2	3.29	3	2	0	1
22	8.1	4.71	1	1	0	0
23	2.9	2.07	2	0	0	2
24	5	3.25	1	0	0	1
25	3.4	2.29	22	16	0	6
26	6.4	3.66	5	0	0	5
27	5.8	3.39	8	0	0	8
28	5.3	3.31	22	22	0	0
29	3.6	2.38	11	10	0	1
30	5.1	3.36	20	0	0	20
31	5	3.32	17	0	0	17
32	6.9	4.41	24	0	0	24
33	6.2	4.07	6	0	0	6
34	1.4	1.15	6	2	0	4
35	1.9	1.43	1	0	0	1
36	6.1	3.81	0	0	0	0
37	0.9	0.9	0	0	0	0
38	2.1	1.56	15	7	0	8
39	10.8	6.44	11	1	0	10
40	3.5	2.4	2	0	0	2
Total	-	-	339	146	0	193

Table B.10: Results for Geographical Locations as evaluated by Phrase Matching

Site #	Text Size (KB)	Elapsed Time (s)	Matches Found	Correct Matches	False Negatives	False Positives
1	6.9	2.93	62	15	0	47
2	3.9	1.62	17	3	0	14
3	6.3	1.93	18	5	0	13
4	8.8	2.98	92	47	0	45
5	2.8	0.9	13	11	0	2
6	3.6	1.01	3	1	0	2
7	4.5	1.23	15	13	0	2
8	6.6	1.62	62	21	0	41
9	3.6	0.97	17	0	0	17
10	5.8	1.35	48	14	0	34
11	3.3	0.87	10	4	0	6
12	3.8	1.04	37	8	0	29
13	3.9	1.16	50	10	0	40
14	5.9	1.88	121	46	0	75
15	6.3	4.02	57	10	0	47
16	3.1	1.45	27	0	0	27
17	2.6	1.46	47	6	0	41
18	5.8	1.93	40	2	0	38
19	2.1	1.37	43	3	0	40
20	2.7	0.94	11	0	0	11
21	5.2	1.58	55	25	0	35
22	8.1	1.76	39	0	0	39
23	2.9	1.29	60	10	0	50
24	5	1.42	38	6	0	32
25	3.4	1.07	29	13	0	16
26	6.4	1.62	62	7	0	55
27	5.8	1.91	62	17	0	45
28	5.3	1.3	10	3	0	7
29	3.6	1.14	43	4	0	39
30	5.1	1.69	121	31	0	90
31	5	1.51	76	12	0	64
32	6.9	2.65	211	102	0	109
33	6.2	1.9	113	11	0	102
34	1.4	0.69	18	4	0	14
35	1.9	0.79	28	16	0	12
36	6.1	1.51	62	22	0	40
37	0.9	0.56	7	2	0	5
38	2.1	0.82	36	0	0	36

39	10.8	3.54	60	26	0	34
40	3.5	1.59	54	16	0	38
Total	-	-	1974	546	0	1433

Table B.11: Results for Sports as evaluated by Phrase Matching

Site #	Text Size (KB)	Elapsed Time (s)	Matches Found	Correct Matches	False Negatives	False Positives
1	6.9	4.69	35	31	0	4
2	3.9	2.59	3	0	0	3
3	6.3	3.98	2	0	0	2
4	8.8	5.74	16	0	0	16
5	2.8	1.99	3	0	0	3
6	3.6	2.22	1	0	0	1
7	4.5	3.01	34	33	0	1
8	6.6	4.04	9	0	0	9
9	3.6	2.25	0	0	0	0
10	5.8	3.5	3	0	0	3
11	3.3	2.27	1	0	0	1
12	3.8	2.33	3	0	0	3
13	3.9	2.75	4	0	0	4
14	5.9	8.7	17	0	0	17
15	6.3	4.02	2	0	0	2
16	3.1	2.19	2	0	0	2
17	2.6	1.91	1	0	0	1
18	5.8	3.59	1	0	0	1
19	2.1	1.63	1	0	0	1
20	2.7	1.9	2	0	0	2
21	5.2	3.37	6	0	0	6
22	8.1	4.76	0	0	0	0
23	2.9	2.08	0	0	0	0
24	5	3.28	11	0	0	11
25	3.4	2.31	3	0	0	3
26	6.4	3.73	1	0	0	1
27	5.8	3.45	4	0	0	4
28	5.3	3.35	2	0	0	2
29	3.6	2.4	3	0	0	3
30	5.1	3.46	48	48	0	0
31	5	3.39	30	26	0	4

32	6.9	4.55	31	1	0	30
33	6.2	4.16	26	25	0	1
34	1.4	1.18	8	8	0	0
35	1.9	1.45	1	0	0	1
36	6.1	3.8	10	0	0	10
37	0.9	0.92	0	0	0	0
38	2.1	1.59	0	0	0	0
39	10.8	6.56	16	0	0	16
40	3.5	2.43	0	0	0	0
Total	-	-	340	172	0	168

Appendix C
SOURCE CODE

All of our code is written in Python.

It is located in github at the following URL:
<https://github.com/electricity345/Full.Text.Classification.Thesis>