A Preliminary Parser for the Natural Language Processing of Supreme Court Cases

Rob J. Dallara
Department of Computer Science,
University of North Carolina at Chapel Hill
dallara@live.unc.edu

Abstract

The problem of Natural Language Processing is one of the most elusive, yet most promising problems in the field of computer science. Although efforts to create systems that can process language in the same ways that humans do have persisted for decades [1], their results so far have proved elusive: humans are still the only entities capable of natural language processing. If computing devices were able to handle natural language in the same way as people do, the possibilities would be unlimited. Computers would be able to easily derive meaning from newspaper articles, make accurate medical diagnoses, write works of literature, etc… (perhaps even better than humans do!). In this paper, I present a possible method for parsing U.S. Supreme Court cases, along with a Python-based parser for conducting this task. I then discuss its corresponding results, as well as possible methods for improving them. Although this parser's methods are fairly simple compared to other NLP systems currently in the field, they are unique in the texts that they analyze.

1. Introduction

Natural Language Processing (often abbreviated NLP) is the ability to usefully process and gain useful information out of natural human language [2]. While humans can often process natural language quite well, computer scientists are still far from producing systems that can handle natural language as well as humans do (and in the same way that they do). Numerous efforts have been put forth to try and figure out how to make computers handle natural language at least as well as humans for decades, from early systems decades ago to more sophisticated ones such as IBM's Watson computer [3].

The success of these efforts would yield enormous applications for the world of computing as well as humanity as a whole. Computers and robots might be able to communicate with people and with one-another in ways only seen in science fiction, computers might be able to write works of literature on par with those of humans completely unassisted, smart phones might be able to accomplish user tasks just by listening to their instructions, etc… NLP systems could also be used to assist doctors in medical diagnoses, catch nearby potential criminals by interpreting what they say, etc… The potential applications of human-level artificial NLP systems are literally limitless.

In this paper, I present a new system for parsing the natural language text of recent U.S. Supreme Court cases. To the best of my knowledge, this is the first method created to deal with NLP in these particular types of texts (which are unique in their style, organization, vocabulary used, etc…). I will first give a more general discussion of NLP and its history in section 2 in order to get the reader up to speed on where natural language processing is today. I will then give a general, high-level overview of my project's design and implementation. Then in section 4, I will dive into the greater details for how different features and designs of my parser actually work at the code level (along with some concrete examples to assist the reader). Section 5 will cover my parser's results, their meaning, and my parser's strengths and weaknesses. Section 6 will cover some other related works on NLP in more detail, and section 7 will conclude the paper.

2. Background

2.1 The Early History of NLP

The earliest NLP systems were often solely based on large, complex, hard-coded rules. While these systems could often capture fairly large portions of everyday languages, they lacked the flexibility to adapt to new grammatical constructs, languages, and more advanced grammatical constructs that occasionally appear in texts [1].

2.2 More Recent NLP Examples

Since the 1980s, NLP systems have increasingly used machine learning methods as well as statistical means to process natural human language [7]. These approaches are much more flexible because they allow for more flexible handling of words and grammatical constructs (including new ones) depending on their context in the language used at the time. A key element of these methods (as with older ones in decades past) has been part-of-speech (POS) tagging of different words as they come up in a piece of natural language. These can be figured out based on both grammar rules and non-deterministic means such as the ones mentioned above. From this point, it becomes much easier to work with the rest of the text and discern meaning from it. One shining example of a recent NLP project is IBM's Watson System [4], which successfully beat humans at jeopardy in 2011 [5]. It was able to do so through large bodies of training text, multiple machine learning algorithms, inferences about different questions as they were asked, and probabilistic means to give "confidence" scores to different possible candidate answers [3]. Watson is now being deployed in other information-intensive fields to process natural language in real-world contexts (such as health care diagnoses [4] and information extraction to find causes of disease and development problems in Africa [6]).

2.3 Why is NLP Hard?

Few constructs are as challenging for a computer to master as natural human language. Natural language has very complex grammatical rules, idioms, words with multiple meanings, and numerous other ambiguities which even confuse humans on a regular basis. It is also very difficult for computers to figure out which words are "connected" to other words under certain circumstances. For example, the sentence "I saw a man with binoculars" could mean one of two things: "I saw a man" (and the man had binoculars), or "I saw a man using my binoculars." While a human might easily assume the latter possibility, a computer might not without a large amount of background knowledge of the world. Worse, if a computer makes a wrong assumption about a critical piece of text, this error could propagate throughout the rest of the current sentence (or even text as a whole), causing the NLP parsing algorithm to completely misinterpret the information that the text is trying to provide!

2.4 Possible Future Directions of NLP

Given the present challenges that natural language processing currently faces, it will take quite a while for NLP systems to reach and ultimately surpass the natural language capabilities of humans. While that is probably the ultimate goal of NLP, there are several intermediate goals that could potentially be achieved in the meantime. One such goal is improved part-of-speech tagging of human language in order for NLP systems to glean more useful information out of language texts and utterances. Ultimately, these pieces of information from separate sentences could be united together for NLP systems to obtain a deeper understanding of what certain texts actually say and mean. However, this may require refinements to many of the traditional NLP methods mentioned in this paper as well as new techniques perhaps based on improved understanding of how the human brain works (such as bigger, improved neural networks to deal with NLP). Ultimately, the evolution and future improvements made to natural language processing algorithms will probably depend largely on general advances made in the field of Artificial Intelligence [7].

3. Overview of Project

3.1 Stanford Java POS Tagger

In order to tag the different parts of speech in the different Supreme Court cases analyzed by my parser, I decided to use a Java-based part-of-speech (POS) tagger created by The Stanford Natural Language Processing group [8], the algorithm of which is based on the one described in [9]. The smaller version of the tagger makes use of two English "tagger" files (used in

the POS tagging process), while the full version of the tagger has a much wider range of tagger options, including taggers for other languages. Since my project solely focused on different U.S. Supreme Court cases, I chose to use the "basic" version of the tagger.

The English part of the tagger works by taking advantage of the University of Pennsylvania's Treebank tag set [10]. This tag set contains different labels for different common constructs in the English language. For example, the "NN" label is used to label common nouns, while the "RB" label is used to label adverbs. The tagger looks at individual words and their surroundings in order to come up with labels for each word that best fits their different grammatical contexts in different sentences. An example of a fully labeled sentence can be found below in figure 1.

```
Reading POS tagger model from taggers/english-left3w
I_PRP saw_VBD the_DT man_NN with_IN binoculars_NNS
```

Figure 1.

3.2 Liblinear Tool

I used the liblinear tool in order to actually perform logistic regression on my sets of features for each of the court cases used in the project [11]. The liblinear tool is a library designed for machine learning projects conducting linear classification on large data samples as well as large sets of data. It can be used to create models from training data using logistic regression and several other means (such as vector regression). Such models can then be used on test data, ultimately conveying the final accuracy for the results. This is known as supervised learning [12] and is very common in the field of machine learning.

For my project, I decided to manually label 100 U.S. Supreme Court cases as training data, and 100 Supreme Court cases as test data. The first step of my main parser algorithm looks at the texts and labels of the different training cases, looks for certain features in each of them, and puts the result into a file. I then use liblinear's "train" method to create three different models based on the training data. I then input all the unlabeled test data into each of these models, and the parser then outputs the final overall accuracies from each model.

3.3 Use of Logistic Regression

In particular, my court case parser makes use of logistic regression. Logistic regression attempts to classify data into two or more classes [13] using a series of predictor variables. Some examples of predictor variables used in my Supreme Court case parser are the numbers of different grammatical constructs found and the number of appearances of different words in a given case text. The classifications of different variables in a logistic regression problem are estimated using their features fed through a logistic function. This function takes the form of the equation below in figure 2, along with its corresponding graph in figure 3 (both figures are obtained from [13]).

$$F(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}},$$

Figure 2.

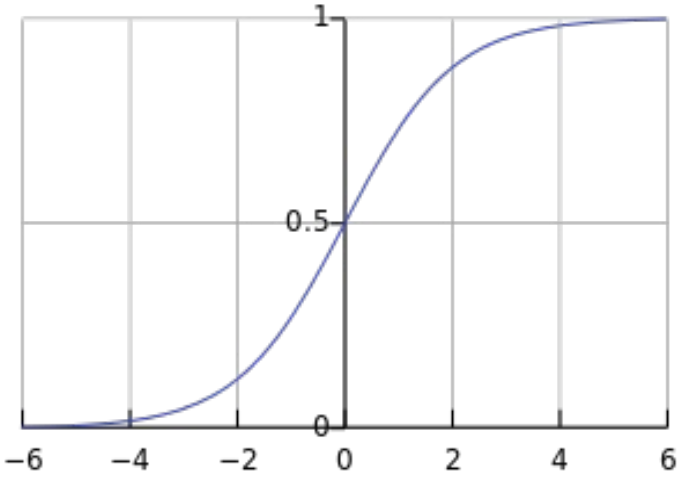

Figure 3

4. Implementation Details

4.1 Prior Selection of Training Cases

The 100 training cases were chosen out of a corpus of more than 9000 Supreme Court cases heard between 1960 and 2012 from the "Findlaw" website [14]. The actual comprehensive set of case URLs and

years were obtained using the MATLAB script in the zip file at [15]. The URLs and years of all the cases in that set were then placed into corresponding text files from the MATLAB command window (a case URL and its corresponding year are placed at the exact same line in their respective files in order to ease the process of finding case years later on). In order to choose a wide variety of different Supreme Court cases related to many different issues during that time frame, the 100 cases were randomly picked based on a hundred random numbers all ranging from 1960 to 2012 inclusive. Some of the cases chosen by year were landmark Supreme Court cases (such as Engel v. Vitale, Tinker v. Des Moines, etc…), and these were chosen manually. All the other training cases within their year were chosen completely randomly.

4.2 Division of Training and Test Data

Both the training and the test data were chosen randomly from the "Find Law" website [14] mentioned above using the MATLAB script found at [15]. After the 100 disjoint training cases were chosen out of all the cases embedded in a data structure using this script, 100 additional cases (non-overlapping within themselves and with the training cases) were chosen as test data. Both the training and test data were then labeled in preparation for logistic regression with "+1" and "-1" labels. The "+1" labels were given to the URLs of cases in which the petitioner won, while "-1" labels were given to the URLs of cases where the petitioner lost. I used the Oyez Project website [32] for help figuring out whether or not the petitioner won in each of the cases in my data.

For clarification, a "petitioner" (sometimes called the "appellant") in a court case is the party who presents a "petition" to the court. In Supreme Court cases, this is usually the party who lost in a lower court [16]. The petitioner can be either the plaintiff (the party who brings a case against the other party) [17] or defendant (the party whom the case is brought against) [18] in such court cases. The other party in such cases is called the "respondent", against whom the petition is filed [33].

Although I obtained my test data randomly using the URLs generated by the MATLAB script [15] I used earlier, I had to manually substitute two of the URLs with different ones corresponding to the same case (they are listed in the "Important Test Data Note" file in the "Misc" folder in my project's zip file). One of the URLs was replaced due to the fact that it was merely a decree for the actual opinion reported, and the other was replaced due to the fact that its text was the result of a denied "writ of certiorari" (occurs when the losing party of a case asks a higher court to review the decision of a lower court [19]) for the original decided case at the Supreme Court level. The new URLs and their corresponding years were then added manually to the comprehensive case URL and year files respectively since the MATLAB script did not "catch" them earlier (the old URLs and corresponding years still remained in the files, but were not used in my actual tests).

4.3 Parsing of Individual Court Cases

My parser makes use of Python's *urllib2* library in order to grab the HTML from each web page corresponding to each Supreme Court case. The URL of each case is stored in a specific file depending on whether or not the case is in my training set or test set. My parser reads one line at a time from the correct file (which depends on whether or not it is currently handling training or test data), each of which contains both the URL corresponding to the case and its corresponding label ("+1" or "-1")(my test cases also had to be labeled because liblinear makes use of labels combined with instance features in order to calculate the final accuracy of the test data). The first thing my parser does when looking at each individual case is parse the complete HTML structure of its corresponding web page on the "Findlaw" website [14].

Once this HTML is obtained and saved to a variable, it undergoes filtering to eliminate certain undesirable characters which could negatively affect the accuracy of the features extracted for the case later on in my parsing algorithm. The first thing that is filtered out by my parser is undesirable whitespace characters (e.g. newline characters, " " entities in the HTML), as some words are directly followed by them without regular spaces in between them (by "regular space", I mean ASCII character 0x20). Other undesirable characters such as quotation marks, numbers (as these often correspond to references within the case, and shouldn't be in the final bag of words), question marks, and periods are also eliminated during this filtering phase (although periods are still allowed in "U.S" and "U.S.A." terms, and this is also accounted for).

To get the final "filtered" text consisting of just the actual words of the case, my algorithm then goes through the filtered html one last time and tries to figure

out where the actual case text begins and ends in the HTML. This is done through special “markings” in the HTML not present anywhere else on the page (and found in all the web pages for each case because of the way the “Findlaw” website consistently organizes the HTML of different case pages). In between those markings, the text is parsed through one last time in order to filter out all the opening and closing brackets of HTML elements that might be intermingled in the actual case text (such as paragraph and hyperlink elements). The final result is the actual text of the case with as much noise removed from it as possible.

4.4 POS Tagging of Case Texts

One of the features in my parser makes use of short grammatical phrases in which each word is labeled by its part-of-speech (POS). These phrases are used in my parser in order to detect court cases potentially dealing with particular issues at particular times from 1960 to 2012. For example, a case with the phrase “right of women” from the 1960s or 1970s is probably more likely to rule in favor of women’s rights, while a case with the phrase “national security” might be more likely to rule on the side of the party arguing on the side of national security (particularly if the case occurred after the 9/11 attacks).

In order to get more information out of different cases, these phrases are searched for in the texts of different court cases (and in particular grammatical formats, as they will often fit in their text in a certain way). In order to actually do this part, I made use of the Java POS tagger from the Stanford NLP (Natural Language Processing) group [8]. Although I also looked into possibility of using the nltk [20, 21] POS tagger for the Python programming language, I decided to stick with the Java tagger since it seemed to produce more accurate POS labelings on short random input texts than the nltk tagger.

The Stanford part-of-speech tagger makes use of many different taggers in order for one to actually label different texts with their POS, including some which work on other languages. However, since my project solely focused on U.S. Supreme Court cases, I used the basic version of the tagger which only had English taggers available for POS labeling. There were two taggers available for English: a tagger which looks at the most recent 3 words to the left of a position, and one that looks for surrounding words in both directions (the taggers were found in the “models” folder within the top folder containing the tagger files). While I tried using the “bidirectional” tagger on the court cases in my repository, my parser ran extremely slowly and crashed when it ran out of memory. This may be due to the fact that this particular English tagger looks for words in both directions without bound (or with an extremely high bound). In contrast, my parser worked much more quickly with the tagger which looks at the most recent 3 words to the left, and so I decided to make use of that tagger for my project (although my code does allow the user to specify any arbitrary tagger as an argument to the program if he/she wants to use another English tagger compatible with the Stanford POS tagger, should one be created).

In order to label each of my grammatical phrases chosen for the parser, I put each of them on a separate line of the “short_grammatical_phrases_to_look_for.txt” file in my project directory. I then created and ran the “PhraseLabeler.java” program which takes the short grammatical phrases from this file and runs them through the Stanford POS tagger. The results for each phrase were then printed to *stdout* and copied into the “java_grammatical_phrase_labelings.txt” file to be used later as features in my parser’s algorithm. More specifically, the number of each grammatical phrase (and total number of grammatical phrases found) are recorded as features for each court case, in both the training and test data.

4.5 Bag of Words Approach to Features

In order to get comprehensive case-specific information from each of the court cases in my data, I decided to look at the “bag of words” for each of those cases. In order to do this, I first obtained all the complete texts of all the court cases in my training and test data combined. This was done to obtain a complete, comprehensive set of words appearing at least once across all the cases. For each case (whether in the training or test data), the number of each word is counted up throughout the whole text (and any words in the “master” word set that don’t appear at all are given counts of zero). Each of these counts for individual words become separate features for the algorithm to pass into liblinear in its final stage.

4.6 Other Individual Parser Features

One important feature in my parser is the ability to look for certain pairs of words in Supreme Court cases that might be related to the time period in which they were heard. For example, if the words "business" and "regulate" appear close to one another in a particular case, it could be an indication that this case dealt with the constitutionality of different regulations on private businesses. Such a case might be more likely to rule against certain business regulations if it took place after 1980 (an era of overall decreasing business regulations) as opposed to before 1980 (when different regulations against business were either steady or increasing). Such pairs of nearby words are more flexible than the simple grammatical constructs also used by my parser (the grammatical constructs have to match exactly in POS-labeled text!). A pair of words will count as long as they are within a certain range of each other (the default is within five words, although this could also be tweaked in theory). Another way in which pairs of related words are less rigid in my parser is the fact that it also looks for certain variations of these words to see if any pair up in the same way as the "rigid relations" do. For example, if the words "freedom" and "woman" appeared close to one another, my parser would still catch that because "freedom" is a variant of the word "right", and the words "right" and "woman" form a historical relation pair. Ultimately, my parser looks for how many of each such historical pair appear in a case, as well as the total number of historical relations overall.

In addition, my parser looks for certain keywords in the texts of Supreme Court cases that could give overwhelming evidence on how the case was ruled. For example, the word "reverse" (or any of its variants) could hint that a particular case overturns the decision of a lower court (so its outcome will be favorable to the opposite party of that lower court). Although my parser does not look at cases from lower courts, such key words in conjunction with text solely in the Supreme Court case in question could dramatically change the likelihood that a case was ruled one way or another. Since the keywords are searched for individually, my parser does not take into account their part-of-speech like it does for short grammatical phrases. My parser looks for the total numbers of each keyword as well as the total number of keywords overall and treats them as separate features for analysis in liblinear at the last stage of the algorithm. Finally, my parser also tries to obtain the year for each text and use it as a single feature. In order to do this, I looked at the comprehensive file of case URLs generated earlier for the case in question, and found its year in the corresponding location in the file containing all the case years.

4.7 Final Merging and Results in Liblinear

In order to actually get the final results for my parser's accuracy over all the training and test data chosen earlier, my parser invokes a main procedure when it is run which parses all of the training cases as well as all the test cases. The main procedure first obtains all the text from all the Supreme Court case data combined (both training and text) to figure out the complete bag of words that the algorithm will have to deal with. Once this is done, two different loops iterate through each set of case data one case at a time, resulting in complete feature files for both the training and test cases. In particular, these loops both invoke a procedure which forms a "features" string for each individual case which liblinear can understand. This feature string contains the label of the particular case, along with all the numbers of different features found when invoking different methods to find quantitative information about these features.

Once all the feature data is created for both the training and test cases, my parser invokes a training shell script which executes the "train" program in the liblinear library on the training data features file (the "train" program had to be compiled manually beforehand by invoking the "make" command in the Linux command line while in the top liblinear directory). The training script actually invoked the "train" program three different times in order to train three different models based on my data on three different types of logistic regression supported in liblinear: L2-regularized logistic regression, L1-regularized logistic regression, and L2-regularized logistic regression (dual) [11] (the first two are differentiated in the way they use regularization to help solve the problem and prevent overfitting of data [22, 23], while the last one can be used to work with the "dual" form of L2-regularized logistic regression). The dual form of logistic regression is discussed more in [24]. Finally, my "testing" shell script ran all of my test data features against the three models created during training, and the final accuracies for each of the three tests were then output on the command line.

5. Evaluation

5.1 Overall Parser Accuracy

My parser ran on the same set of test data on each of the three training models generated earlier using the different types of logistic regression that liblinear supports. The final accuracies for these different tests are output in the table below.

Type of Logistic Regression	Accuracy
L2-regularized logistic regression	66%
L1-regularized logistic regression	67%
L2-regularized logistic regression (dual)	67%

Overall, my parser seems to be fairly accurate at labeling test data given a significant prior balanced set of training case data. L1-regularized logistic regression and the dual version of L2-regularized logistic regression both have accuracies of 67%, suggesting that they are the most accurate methods for labeling Supreme Court cases based on their text in the context of the features I use in my parser. On average, the parser's accuracy is about 66.66%, significantly higher than "random chance" (50% accuracy), suggesting it could potentially be refined even further for real-world labeling of Supreme Court cases and other legal documents.

5.2 Accuracy Comparisons Between Logistic Regression Types

The accuracies between the different types of logistic regression I used in liblinear for evaluation are fairly close to one another. This may be due to similarities in how these different types of logistic regression actually work. For example, L1 and L2-regularized logistic regression are only different in the "regularization" term that they use [23], and the dual version of L2-regularized logistic regression is just a dual version of the same type of problem. This suggests that different types of logistic regression may be equally suited for the type of problem that my parser deals with, but the best type of logistic regression to use on data may still be different from problem to problem depending on the nature of the data and how it should best be divided up into classes to yield the highest possible accuracy during analysis.

5.3 Strengths and Weaknesses of Parser

There are probably a number of factors that made my parser fairly accurate over my data set of Supreme Court cases. My parser attempts to look at several different types of features, from useful word pairs to short grammatical phrases which can give strong hints about how a case was ultimately decided. It also gives much more weight to certain keywords when it finds them. The combination of all these different features together may have helped my parser better distinguish between different types of Supreme Court cases. However, the bag of words probably had the most impact on my parser's accuracy, as it contained the most individual features (one for **each** possible word!) and could best take advantage of the fact that Supreme Court case decision texts are written differently depending on how the case was actually decided.

Nonetheless, the parser does have several weaknesses that could have reduced its accuracies on my evaluation. First, its set of training data only contains 100 cases, and the set of test data also contains that many cases. This was due to the fact that the cases were not already labeled according to who won, and hence had to be manually labeled. As a result, the set of training data may have not been comprehensive enough to truly best train my parser, and the set of test data may have been too small to yield the "true" accuracy of my parser. In addition, most of the cases I used in my data set were not "landmark" Supreme Court cases, meaning that some of the cases parsed had nothing to do with particular historical trends taking place between 1960 and 2012.

There are also some potential weaknesses in my parser's algorithm. First, I only use the Stanford POS tagger on an individual case basis, and since the tagger has to load an English tagger **each** time it parses over a text, the tagger is loaded once again on every single case text, resulting in increased run-time of the algorithm (although I had to run it this way to avoid running out of memory when executing my parser's algorithm). In addition, my parser's algorithm does not necessarily filter out every last possible special character or construct present in the text: it only filters

out the most common ones (such as numbers (which often correspond to references in the case texts) and parentheses (the algorithm only cares about what text is inside parentheses). My parser's algorithm also doesn't take into account variations of words (e.g. "analyze", "analyzing") in its bag of words approach (although it does this for the two-word pairs and keywords it searches for). Furthermore, since my parser deals with legal texts (which can be very verbose and full of advanced words not often used in other contexts), some of the simple words it focuses on may not be as relevant as they might be in other contexts (such as newspaper articles, Wikipedia pages, etc...).

6. Related Work

Byrd et al. [25] from the IBM Thomas J. Watson Research center propose a hybrid NLP pipeline to try to improve early detection of heart failure. Their method relies on clinical notes from health records to look for certain diagnostic criteria signifying heart failure. Like my Supreme Court Case parser, it also conducts POS tagging and looks for certain keywords and phrases, but it also employs dictionaries and tries to detect boundaries of sentences and paragraphs (something which my parser does not do). In the end, their method has approximately 93% precision and 90% recall (the definitions of these terms are discussed in [26]), suggesting its usability in real-world medical settings.

Many of today's NLP algorithms make use of statistical methods (with the help of training data) to produce high accuracy results on a wide, flexible range of test data in certain domains. However, if these same algorithms are migrated to new or different domains after training, they will often lose their accuracy due to the fact that none of the training data was in these new/different domains. Zhang, Damerau, and Johnson [27] propose several possible methods for dealing with this, such as adding a few new labeled examples from the new domains and trying to estimate performance degradation in advance so manual improvements can be made. Zhang et al. focus on this problem in the area detecting sentence segmentation.

Zirn and Stuckenschmidt [28] also propose a means of figuring out which members of German political party coalitions were assigned certain ministries based on analysis and comparisons of different political documents. In particular, it looks at many different topics (and how they are related) that come up in these documents to try to come up with its results. This method is more accurate than a simpler baseline method mentioned in Section 3.4 of their paper, which involves looking at words surrounding different keywords for different individual topic dimensions in a document.

Other methods for dealing with NLP problems have also been proposed in recent research, such as improved neural networks and improved statistical models that are based in the actual distributions of words in real languages. For example, Collobert et al. [29] propose a unified neural network architecture to deal with a variety of tasks related to natural language processing. Their emphasis is avoidance of NLP algorithms for specific tasks only in order to get an improved overall NLP system. In addition, Goldwater, Griffiths, and Johnson propose improved statistical models based on the actual frequencies of words in real natural languages [30]. In particular, they take advantage of the fact that word frequencies in language tend to follow a power law [31] distribution. In other words, a small subset of words occur much more frequently than a larger subset of infrequently occurring words in natural language.

7. Conclusions and Future Work

Natural Language Processing is a very challenging, yet extremely promising problem in the field of Computer Science. I have developed a simple NLP parser for U.S. Supreme Court cases and their decisions which has a fairly high accuracy (approximately 66.66%). Although this parser does have some weaknesses in the comprehensiveness of its algorithm and handling of words and short grammatical phrases (a lot of this is unique to the nature of legal texts), it is unique in the fact that it solely focuses on recent U.S. Supreme Court cases, and is the first parser to my knowledge to do so. There are a number of future improvements that could be made to this parser. These range from a larger set of training/test data to improvements to the case webpage HTML filtering procedure and even more comprehensive sets of grammatical phrases/keywords to search for. Ultimately, an accurate parser of Supreme Court cases such as this one could be refined to work in other domains with complex texts and help increase the accuracies and efficiencies of NLP algorithms to levels much closer to human abilities.

References

[1] Nadkarni, Prakash M., Lucila Ohno-Machado, and Wendy W. Chapman. "Natural language processing: an introduction." *Journal of the American Medical Informatics Association* 18.5 (2011): 544-551.

[2] "Introduction to Natural Language Processing." Introduction to Natural Language Processing - The Mind Project. The Mind Project, 2006. Web. <http://www.mind.ilstu.edu/curriculum/protothinker/natural_language_processing.php>.

[3] High, Rob. "The Era of Cognitive Systems: An Inside Look at IBM Watson and How it Works." *Redguites for Business Leaders* (2012).

[4] "IBM Watson." *IBM Watson*. N.p., n.d. Web. <http://www.ibm.com/smarterplanet/us/en/ibmwatson/>.

[5] "What Is Watson?" *IBM Watson: What Is Watson?* N.p., n.d. Web. <http://www.ibm.com/smarterplanet/us/en/ibmwatson/what-is-watson.html>.

[6] "IBM Brings Watson to Africa." *IBM News Room*. N.p., 06 Feb. 2014. Web. <https://www-03.ibm.com/press/us/en/pressrelease/43106.wss>.

[7] "Natural Language Processing." *Wikipedia*. Wikimedia Foundation, 27 Nov. 2014. Web.

[8] "Stanford Log-linear Part-Of-Speech Tagger." *The Stanford NLP (Natural Language Processing) Group*. Stanford University, n.d. Web. <http://nlp.stanford.edu/software/tagger.shtml>.

[9] Toutanova, Kristina, et al. "Feature-rich part-of-speech tagging with a cyclic dependency network." *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics, 2003.

[10] "The University of Pennsylvania (Penn) Treebank Tag-set." *The University of Pennsylvania (Penn) Treebank Tag-set*. N.p., n.d. Web. <http://www.comp.leeds.ac.uk/amalgam/tagsets/upenn.html>.

[11] Lin, Chih-Jen et al. "LIBLINEAR -- A Library for Large Linear Classification." *LIBLINEAR -- A Library for Large Linear Classification*. Machine Learning Group at National Taiwan University, n.d. Web.

[12] Cord, Matthieu, and Pádraig Cunningham. "Supervised Learning." *Machine Learning Techniques for Multimedia Case Studies on Organization and Retrieval*. Berlin: Springer, 2008. 21. Print.

[13] "Logistic Regression." *Wikipedia*. Wikimedia Foundation, 12 July 2014. Web.

[14] *Findlaw: Cases and Codes*. N.p., n.d. Web. <http://caselaw.findlaw.com/>.

[15] "Dissent." *Dropbox*. N.p., n.d. Web. <https://www.dropbox.com/sh/tqa65yswszvch3t/AACoIkeur_tvgJBZddw9MrUGa?dl=0>.

[16] "Petitioner." *LII / Legal Information Institute*. N.p., n.d. Web. <http://www.law.cornell.edu/wex/petitioner>.

[17] "Plaintiff." *TheFreeDictionary.com*. N.p., n.d. Web.

[18] "Defendant." *TheFreeDictionary.com*. N.p., n.d. Web.

[19] "Writ of Certiorari." *Wex Legal Dictionary*. Cornell University Law School, n.d. Web. <http://www.law.cornell.edu/wex/writ_of_certiorari>.

[20] "Natural Language Toolkit." *Natural Language Toolkit — NLTK 3.0 Documentation*. N.p., n.d. Web.

[21] Bird, Steven, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. Beijing: O'Reilly, 2009. Print.

[22] "Regularization (mathematics)." *Wikipedia*. Wikimedia Foundation, n.d. Web.

[23] Rostami, Afshin. "L1 vs. L2 Regularization and Feature Selection." (n.d.): n. pag. Web. <http://cs.nyu.edu/~rostami/presentations/L1_vs_L2.pdf>.

[24] Yu, Hsiang-Fu, Fang-Lan Huang, and Chih-Jen Lin. "Dual coordinate descent methods for logistic regression and maximum entropy models." *Machine Learning* 85.1-2 (2011): 41-75.

[25] Byrd, Roy J., et al. "Automatic identification of heart failure diagnostic criteria, using text analysis of clinical notes from electronic health records." *International journal of medical informatics* (2013).

[26] "Measuring Search Effectiveness." (n.d.): n. pag. Web. <https://www.creighton.edu/fileadmin/user/HSL/docs/ref/Searching_-_Recall_Precision.pdf>.

[27] Zhang, Tong, Fred Damerau, and David Johnson. "Updating an NLP system to fit new domains: an empirical study on the sentence segmentation problem." *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*. Association for Computational Linguistics, 2003.

[28] Zirn, Cäcilia, and Heiner Stuckenschmidt. "Multidimensional topic analysis in political texts." *Data & Knowledge Engineering* 90 (2014): 38-53.

[29] Collobert, Ronan, et al. "Natural language processing (almost) from scratch." *The Journal of Machine Learning Research* 12 (2011): 2493-2537.

[30] Goldwater, Sharon, Thomas L. Griffiths, and Mark Johnson. "Producing power-law distributions and damping word frequencies with two-stage language models." *The Journal of Machine Learning Research* 12 (2011): 2335-2382.

[31] "Power Law." *Power Law*. Princeton University, n.d. Web. <http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Power_law.html>.

[32] *The Oyez Project at IIT Chicago-Kent College of Law*. N.p., n.d. Web.

[33] "Respondent." *Wex Legal Dictionary*. Cornell University Law School, n.d. Web.