Legal Informatics – Final Paper Submission – Creating a Legal-Focused Search Engine

I. BACKGROUND

There is a large need for improved access to legal help. For example, 9 out of 10 employees had a legal concern in the last year [1]. Additionally, under-served U.S. citizens with legal issues comprise a latent legal market valued at $45 billion [1]. The latent legal market exists for a variety of reasons. First, the traditional lawyer costs often discourage consumers from seeking any help. Second, even if consumers have decided on wanting to use legal counsel, finding a lawyer with the relevant expertise can be difficult. However, with the prevalence of the Internet and the digitization of information, individuals have become increasingly inclined to look online for obtaining legal answers. For example, 3 out of 4 consumers who are seeking an attorney now use online resources to help them find their legal counsel [2]. And a recent ABA study has shown that, out of the online models that people use to help them find a lawyer, Q&A websites were the most popular [3].

II. PROBLEM AND SOLUTION

Although there exists a wealth of information freely available on the Internet, much of it remains disorganized and it can be difficult to find the exact info one is interested in. Google does an excellent job of helping consumers find relevant links for a query, but it is a general search engine with results that are not always amenable to answering a legal question. For example, if a legal question related to enforcing non-compete agreements is typed into Google, the top search results are general definitions of the legal term as well as recent, related news. This lack of specificity can make it difficult for an average consumer to get the answer they want to their legal question. On the other end of the spectrum are legal case search engines like CaseText. CaseText, for example, returns a set of relevant cases that are annotated by legal professionals and professors. However, CaseText’s target demographic comprises lawyers, academics and law students. An average consumer with a legal question is not necessarily looking for technical cases and statutes.

There are a set of services similar to Rocket Lawyer where questions are entered online, and these questions will be sent to relevant lawyers on the back-end who can answer the questions directly. While these services provide answers from legal professionals who are knowledgeable in the area, they fail to address the growing demand for instantaneous results that web consumers have become accustomed to due to services like Google. Since a large proportion of answers to legal questions are already publicly available on the Internet through legal Q&A websites like Yahoo Answers and Avvo, the goal is to create a platform that can aggregate these responses and make them easy to find for the average user. We thus focused on creating a legal-specific search engine that allows the user to enter a fully-formed legal question and situation. After applying natural language processing and search ranking algorithms, legal-specific search results are then returned to the user with a focus on providing the most relevant sets of questions and answers from legal Q&A websites.
overview - we have built an initial version of a search engine that finds relevant previously answered question and answers from the web and presents the results to the user in a clear, easy to understand format. this system consists of 3 major components, and we have used well-known open source software projects to build these parts of the system. here is a brief description of the different components: (1) crawling and indexing – for web crawling, we used an open-source java-based system called apache nutch, which crawls a seed set of inputted html urls. by crawling websites like avvo, we get the q&a content, which can then be presented to the user. after crawling the content, we need to organize the q&a results by labeling them with tags so users can efficiently find relevant legal answers. upon crawling the info from these q&a websites, we index the questions, answers, and their relevant tags into a database. for indexing and storage, we use another open-source java-based system called apache solr. we integrate the apache nutch framework together with the apache solr framework in order to have the crawler feed the crawled information directly into solr for indexing and storing. (2) query processing – after a user enters in a legal question query, we need to then take that legal question and compare it against our index of information to find the most relevant search results. to accomplish this, we once again leverage apache solr, which utilizes a distance metric to determine the similarity between two documents. this allows us to quantitatively determine which q&a questions should be shown to the user. (3) front-end and back-end interaction – the third and final step is to display results on the front-end for the user to read. we used a javascript library called ajax solr, which allows us to access the data indexed by solr in the database back-end and send it to the front-end to be displayed. we can use html and css to customize the actual layout and user interface of the page. the above is a brief overview of the methods we used, and we will now explain each of the individual components in detail.

Crawling and Indexing - Building a legal search engine requires a large index of legal information, where a subset of that large index would eventually be shown to the user as the search results for their query. To obtain this large index of legal information, we used a web crawler that browses specified web domains in an effort to “crawl” their content. We leveraged the open-source java-based web crawler, Apache Nutch [4]. With Nutch, one can specify specific seed URLs for the crawler to start its descent. The crawler will grab the content from the initial seed IR:, and then move onto links that were present on the seed webpage. By modifying parameters such as crawling depth and the number of links to crawl, one can better tailor the crawler for obtaining the desired legal information. Given that we are building a legal-focused search engine, we limited the crawled web domains to legal-related content, such as the Avvo and Rocket Lawyer websites.

By default, Apache Nutch does not perform any advanced parsing of a webpage’s content. All text is treated the same. However, for legal Q&A websites, we wanted to be able to have the crawler parse the content in a more meaningful way. First, a “document” for us consisted of a single legal question and the lawyers’ answers. For a single document, we wanted the legal question to be indexed as a separate field than the answers. Second, we wanted to be able to index certain tags for the legal Q&A questions, such as the practice area that the question
pertained to and the state in which the question was asked. All of this info was found within the given webpage’s HTML structure, but advanced parsing was required to identify the target info and retrieve it. Thus, we combined Nutch’s crawling abilities with JSoup, an open-source Java-based HTML parser. We were able to use JSoup to “decompose” a given legal question and answer into its various parts and tags by using CSS selectors. In addition to legal question and answers, we also aim to crawl other legal-related content. This content includes websites with relevant legal info, law firm websites, cases and statutes, and related legal news or blogs. JSoup will allow us to more meaningfully parse this non-Q&A legal content.

An impressive feature of Nutch is its ability to run on Hadoop, which is an open-source framework that leverages clusters of processors for distributed storage and analysis of large data sets. Although we have thus far been using a single computer to crawl and store information, we would eventually need to migrate to a cluster of processors for speed purposes if our legal search engine gained enough traction.

The crawling and indexing process is illustrated in Figure 1.

Figure 1. Diagram illustrates the various steps in crawling the Internet for legal-related content, parsing that content, and indexing that content on a storage server.

Query Processing and Search Results – After crawling the web and creating a large dataset of legal-related material, we needed to develop methods to efficiently return the most relevant and personalized results to an individual. This is an important component of the search engine because as the amount of legal data we collect from the web increases, it will take more time to process and return the relevant documents. Apache SOLR plugs nicely into the Apache Nutch web crawler so we can easily index the data that we are collecting from the web. Additionally, the input query from the user should go through a set of basic pre-processing steps so that we can better find similar documents stored in the SOLR index. The pre-processing steps involve fixing all spelling issues in the sentence, removing all the punctuation within the query, and finally removing the stop words in the query. This set of steps helps to normalize the words
in the query so that they can easily be compared against the words in each of the documents we have stored in SOLR.

In order to compare how similar a query q is with a document d, there are several different methods that can be used. We have started by utilizing the basic cosine similarity metric. The first step involves converting the words in all the documents (after preprocessing) into a set of vectors where component i consists of the Term Frequency-Inverse Document Frequency (TF-IDF) score for a given word w.i within the document. The set of all w.i’s will consist of all the words that appear at least once in one of the documents. The TF-IDF score captures frequency of each word within the document and gives a higher weightage to words that are extremely rare across all documents (Figure 2). With this vector representation of each document, we can then easily apply the cosine similarity metric and determine the similarity between any two documents in the set. The cosine similarity metric calculates the angle between the vector representations of the two documents. Intuitively, we see that the larger the angle between two vectors, the more different the underlying documents will be. Similarly, the smaller the angle between two vectors, the more similar the two documents will be. The model we have described is very basic and only uses the word frequency as a metric to capture similarity between two documents. Despite the simplicity of the model, the search results have generally been turning out to be very relevant to the input query, which is a positive sign! We hope to incorporate more advanced natural language understanding techniques that can take advantage of the fully-formed nature of a given user’s query.

![Figure 2](vector_space_model.png)

**Figure 2**: Each document is represented as an n dimensional vector where component i represents the TF-IDF score for word i in the corpus.

**Front-End & Back-End** – The above methods discussed the processes that would occur on the back-end. However, the back-end processes need to be connected with a front-end interface that would be shown and used by an individual. To accomplish this, we used AJAX SOLR [5], which is a JavaScript framework for connecting a front-end interface to an Apache SOLR back-end. On the front-end, a user would enter in a legal question, and optionally, the state in which they reside as well as the relevant area of law. The AJAX SOLR framework then employs a “manager” that takes the front-end query and sends it to the back-end SOLR server. SOLR then performs the search algorithms to find the most relevant search results in the index. These results are sent back through the AJAX SOLR manager, which then translates
those results into HTML that can be used in the front-end to show to the user. The process is illustrated in Figure 3.

![Diagram of how AJAX SOLR is used as a bridge between the front-end user interface and the back-end processes.](image)

**Figure 3.** Diagram of how AJAX SOLR is used as a bridge between the front-end user interface and the back-end processes.

We modified the AJAX SOLR framework to provide certain search engine features such as pagination of our legal search results and filters. Given that tags were added to the legal questions and answers in the crawling and indexing stage, we created a front-end interface that allows for users to filter their search results by state or practice area. Thus, a user can select the “California” filter to only see the legal questions & answers from California.

**IV. CONCLUSION**

We believe that building a consumer-oriented legal search engine that helps an individual find answers to their legal questions will create real value. There are several legal Q&A web services that push questions to actual lawyers, but none of these services provide instantaneous results. We hope to leverage the currently available legal information in order to provide a unique solution to a problem that people face. We have been able to build a basic prototype legal search engine using open-source tools that are freely available. The preliminary results are promising and provide some validation to the tool that we are building. We plan on further pursuing this project by building out and publishing online a fully-featured search engine to help answer legal queries.
V. REFERENCES

hool%20%282%29.ppt

consumers-go-online-to-look-for-an-attorney.aspx

