LawIT - A Personal Online Legal Consultant

Prajna Bhandary, Renard Calalang, Sam Mendimasa
Computer Science and Electrical Engineering
University of Maryland, Baltimore County
prajnab1@umbc.edu, j168@umbc.edu, sam34@umbc.edu
December 18, 2019

***Abstract*—The LawIT projects aims to provide a web-based chatbot service that acts as a free online legal consultant. Currently in the U.S, any form of legal assistance require some form of monetary compensation. Additionally, there isn't any software that offers legal advice, and in most cases individuals would need to seek advice from lawyers or other law related professionals if there was a need. This software attempts to help fill this need by providing a system where users can ask any questions relating to Maryland laws and are given relevant legal text responses pertaining to their inquiry.**

***Index Terms*—Term-Document Matrix, TF*IDF, SVD, DialogFlow, AzureDB, Intents, Entities**

I. Introduction

The context of this project derives from a means of providing legal assistance. Given a scenario where an individual were to commit a certain crime, he/she may want to know what would be the repercussions based on the law of the current U.S state that they reside in. This scenario can be further execrated to include:

- inquiring if any laws are violated,
- checking if an individual is a victim of a crime,
- checking the legality of a certain deed in a given jurisdiction and
- whether or not individuals rights are being infringed upon.

Our approach to filling the need for legal assistance was to develop a chat bot system that we named 'LawIT'. This system would allow interaction and could simulate a conversation between the end user and the system. The chat bot would prompt the end users for information via questions designed to identify the relevant legal text for the user's inquiry. It would then use the user input and a Natural Language Processing (NLP) technique Singular Value Decomposition (SVD) to determine the most relevant record and law pertaining to the user's inquiry. Currently, to limit the scope of the project, the chat bot that we developed will tentatively only accept inquiries for Maryland state laws.

In other words, the chat bot system LawIT has the following functionality and it's intended users are as follows:

- To simulate Conversation with the system end users
- To utilize NLP techniques to find relevant record based on User's inquiry
- Was designed to help anyone that wants to inquire if they violated any laws, were victims of crimes or had their rights infringed upon, and the legality of certain deeds in a specific jurisdiction.

II. Related Works

This section describes other works that provides a chatbot as a service. Our system differs from all of these as it provides legal assistance with respect to specific laws and it also helps users to know if what their action has severe consequences or not. Some of the applications that are available in the market that are similar are:

A. Laws Related to Parking Tickets

An application called 'DoNotPay' asks a handful of questions and looks for keywords that help it parse the particulars of your legal quandary. Then it uses that information to help guide you through the tangles of forms and clauses that make up our legal system. Its an application that helps users to check if they have violated any parking rule and helps them to resolve the parking ticket. It helps users to fill out forms related to that.

B. Laws Related to Divorce

Hello Divorce is a California based website that that only concentrate on divorce cases as there is a high divorce rate in California. It helps users with the procedure that involves filing for a divorce and helps get in touch with the a divorce lawyer.

C. Availability of Legal Forms

'RocketLawyer' is also an application that has legal forms that might be required for work. It only helps with any kind of forms that might be needed for any legal work. So, the user needs to know what form they are looking for when they use this website.

D. Automio - LawyerBot

There is a bot named Automio that was developed for lawyers to interview clients and create instant, customised advice, contracts and legal documents. It saves lawyers the time in creating an initial case. This application helps lawyers with the initial paperwork that might be needed to proceed further with a case.

Our project compared to all the related work is different in ways that it concentrates on all laws related to Maryland and the query can be related to any topic of the law. It has the potential to give a user a clear view of whether they really need a lawyer and know if they have actually committed crime or not.

III. Data Set

As the LawIT system provides legal advice, all of our data were obtained from U.S. government sites containing Maryland state laws and statues. We used Octoparse, a web scraping tool, to retrieve all of our data that we then tentatively stored as commas-separated-values (.csv). The format for this local storage is described as follows:

- Each file correspond to a single Maryland State Law
- Each Law is divided into subsets that has a title number, title name, subtitle number, subtitle name, a header, and a full description
- The header contains a short summary of the full description of the specific subtitle

To better visualize this data set, imagine a three-level tree, where the root corresponds to a specific Maryland state law, the parent nodes are the titles of the law and the children names correspond to the subtitles, where each also has a header and a full description.

After we had retrieved all of the relevant data, we further cleaned the data set to remove any inaccurate records that were outdated. We proceeded and stored all of the laws in a SQL database on Microsoft Azure Database system.

IV. System Framework

In this section we present both the system architecture and technology stack. This includes the hardware, software, programming languages, frameworks, and tools that was used to create LawIT. This also includes the behavior of various components of the system and how they interact with each out. The system architecture is illustrated in Fig. 1.

A. System Architecture

The system architecture of LawIT combines technologies from Google and Microsoft. Lawit uses Google Dialogflow, a conversational agent building platform, to create the chat bot interface. Dialogflow was configured to communicate with a web application created and deployed with Microsoft Azure Database and cloud services tool. Also, the web page that interface with Dialogfow was also hosted on Microsoft Azure.

When a user access LawIT, they are accessing a web page store on azure that describe the application and how to use it. On the web page is an iframe that pull the created Dialogflow chat bot interface. The interface simulates the conversation with the user, and once enough information is gathered, a query is sent to a web app hosted on azure, that performs the NLP algorithm on our data set and either returns and ask for more information or, returns the corresponding laws and statues that the user is interested in.

B. Technology Stack

The technology stack for LawIT included HTML, CSS, JavaScript, and Bootstrap, which was used to create the front end website. The back-end included DialogFlow with a C-based webhook, MSSQL Database, and Azure Web Applications and Azure's managed MSSQL database service. DialogFlow was used to create the create the chat bot interface, handle user's input, and the C webhook allows integration with azure web applications. Azure web applications was used to create and deploy a web app that handle input and perform the SVD search algorithm using Term Frequency Inverse Document Frequency (TF*IDF). We used Azure's managed MSSQL database service store our data set containing the state of Maryland's laws and statues.

V. Software Development Tools

The technologies used to design the system is given in this section.

A. C#

C is a general-purpose, multi-paradigm programming language encompassing strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. C was the primary language used in the development of the chat bot's response generation given the ease with which it can be used with the SQL database back-end. This language was chosen given the group's experience with development and deployment using the language. [1]

B. Dialogflow

Dialogflow builds conversational applications in a natural language processing (NLP) platform. It enables developers to create text-based and voice conversation interfaces for responding to customer queries in different languages. Customer service artificial intelligence (AI) agents- Interfaces can be programmed to answer questions, make appointments, access orders and take requests. [2]

C. Github

Github was used for the collaborative management and development of the codebase.

D. Microsoft Azure

Microsoft Azure (formerly Windows Azure /ær/) is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers. It provides software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) and supports many different

[1] https://searchenterpriseai.techtarget.com/definition/Dialogflow
[2] https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

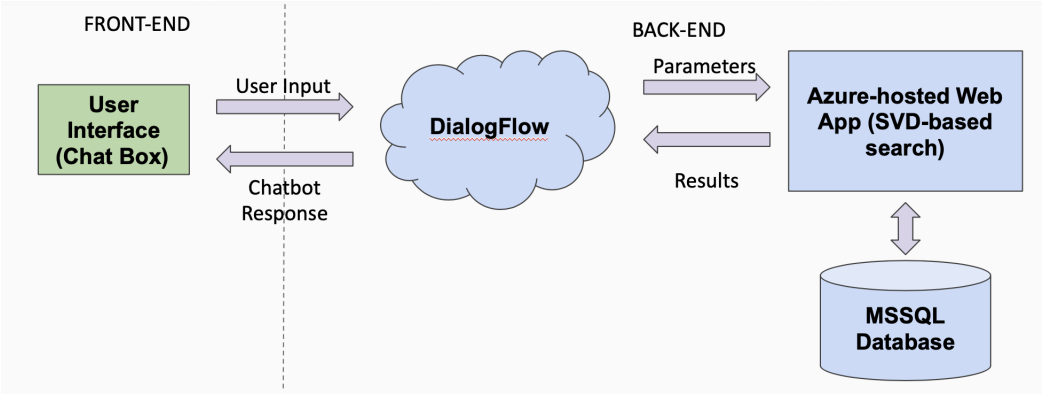

Figure 1. System Architecture

programming languages, tools and frameworks, including both Microsoft-specific and third-party software and systems. [3]

E. Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code. It was the IDE we used to build and deploy to Azure. [4]

VI. IMPLEMENTATION

In this section, we describe how LawIT was implemented. We present and describe how we configure Dialogflow, our chat bot interface, and how we clean and build our data set and document-term matrix. We also describe how the web app was created and how it works.

A. Building the Data Set

Web scraping our data source generated tuples comprising of headers and body texts. We used a web scraping tool called Octoparse on Justia.com, a website focusing on legal information retrieval. The data was then cleaned and organized to generate discrete title, subtitle, and body text tuples.

B. Storing the Data

To create a place to store our data, we initialized an Azure MSSQL database which we could then access using Microsoft SQL Server Management Studio. The Azure database also made it easy to link up with the web application to be discussed later on in this section. We then separated the data out into discrete database tables, namely Titles, Subtitles, and Documents, which we uploaded unto our database. We then created a table to store the individual words, and a table to store our document-term sparse matrix.

We wrote a program that used the Porter stemming algorithm from an imported library(Lucene.Net) to get all the unique word stems from all of the documents in our data set and uploaded that to a Word table in our database. We then used that to generate a set of entities with references to the Document and Word tables and a value ‘Counts’, denoting the instances of each unique word stem in each document.

	term_1	term_2	...	term_n
document_1	#	#	...	#
document_2	#	#	...	#
...	#	#	...	#
document_n	#	#	...	#

Figure 2. Sample Document-Term Matrix

C. User Input and Document Searching

We developed a web application to handle the fulfillment of our chat bot’s queries. The web app was hosted on Azure as well, which allowed for easy deployment and connection to our database. Our web application was designed to take input strings from user queries taken by the chat bot, remove all the stop words, apply the Porter Stemmer to each word to generate a list of tokens with which to conduct our search on. We then used the document-term matrix as a reference to find the most relevant documents. We tried out multiple different approaches, initially with a sum of count values, but immediately discovered that this would then be skewed towards choosing longer documents. Eventually, we decided to use the Text Frequency - Inverse Document Frequency scores, a measure of the importance of words in documents, and leveraging that to find the most relevant document to our

[3]https://en.wikipedia.org/wiki/Microsoft_Azure
[4]https://en.wikipedia.org/wiki/Microsoft_Visual_Studio

search query. We can show the TF*IDF given the following formula:

$$W_{i\ x\ j} = tf_{\ i\ x\ j} \times \log \frac{D}{N_j}$$

Where:

- $W_{i\ x\ j}$ is the weight of term j in document i
- $tf_{\ i\ x\ j}$ is the frequency of term j in document i
- D is the number of documents
- N_j is the number of documents term j occurs in

After finding the most relevant document, we generate a return message with the relevant text and the title and subtitle(if applicable) with the relevant universal citation code. The web application's response is passed back to the Dialogflow instance via a JSON message, which the chat bot receives via webhook fulfillment.

D. Webhook

We used webhooks to create a custom callback from Dialogflow to the Algorithm in the Database. In Dialogflow, a webhook is used to fetch data from the server whenever a certain intent has its webhook enabled. The information from the intent is passed to the webhook service to receive result. The implementation using webhook is as shown in figure 3.

E. User Interface

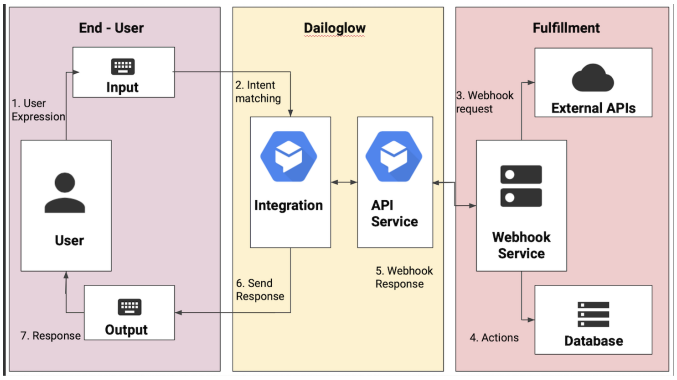

Figure 3. Implementation

The user interface and the overall system works as follows:

1) The end-user types the question they want to ask.
2) Dialogflow matches the end-user expression to an intent and extracts the related parameters from the sentance that the end user has typed.
3) Dialogflow sends a webhook request message to the webhook service. This message contains information about the matched intent, the action, the parameters, and the response defined for the intent.
4) Our service includes the algorithm that matches the the documents by querying the database.
5) The service then matches the right document and sends a webhook response message to Dialogflow. This message contains the document or the exact law pertaining to the query.
6) Dialogflow sends the response to the end-user.
7) The end-user sees the response.

F. Website

As mentioned above, the website for LawIT was built from scratch using HTML, CSS, JavaScript, and Bootstrap. HTML and CSS was used to create the content and styling and javascript and Bootstrap was used to make the site responsive. The Dialogflow interface was reference through an iframe on the site. The website content describe what is LawIT, how to use it, and the intended users of the systems. It also provides documentation and reference for the entire system. Figure 4 displays the website from a user point of view that is interacting with the system.

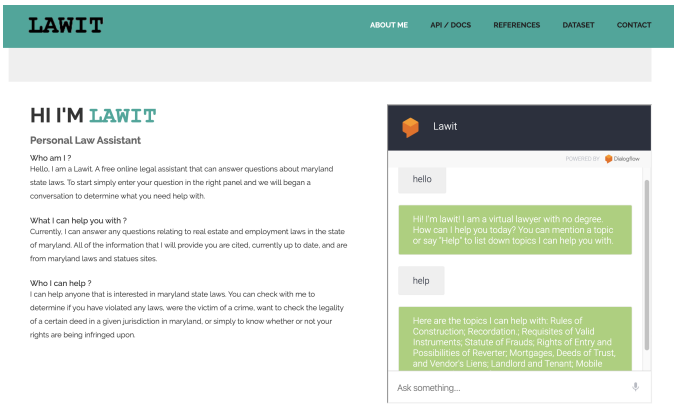

Figure 4. LawIT Website

The test cases mainly follows the following parameters
Uniformity: It should be able to get the right law
Scalability: Multiple users must be able to use the application at a time.
Response time: The time it takes for the service to respond to the request.
Reliability: It should be able to get the closest relevant document related to the user query.
Performance: It should have a good response time.

VII. RESULT ANALYSIS

Our system responds well to the queries that are passed to it. For example, the word 'pet' when typed gives the document most relevant to that search term. The system can take multiple different terms and uses an aggregate of those terms' scores to find a relevant document should one be present. A response

of ”no document was found” is shown when the user fails to enter terms that do not occur in any of the documents.

Initial results would lean heavily towards longer documents, as they had significantly higher counts per term per document, and so we had to look towards other means of allocating weights to each document. After applying the TF*IDF algorithm and giving documents weights based on document length, we managed to improve results significantly, and while longer documents with more words were more likely to be chosen, it did not only choose documents based on length. Documents where words were proportionally more important due to their relative length compared to the length of the text were now more likely to be chosen.

While improvements in the results were seen as less naive methods were used in searching, this also necessitated a slight increase in overhead, which lead to some increase in the delay between responses. Whether this is a product of the systems being used, and whether or not choosing to use a more powerful and proportionally costlier machine, is a topic which may be studied further.

VIII. Conclusion

Our system is a free online utility that users may query to find the most relevant legal text to their concern. The system uses cloud technologies such as Google Dialogflow and Microsoft Azure to provide an accessible and reliable resource for end-users. Natural Language Processing techniques such as stop-word filtering, stemming, and document-term importance measurements have been used to generate responses to a wide range of user inputs.

Certain challenges were faced during the development of the system. Dialogflow's intents and entities proved to be challenging when used with more complex query structures. This was in part due to the active development of the service at the time of this project's lifespan. Legal texts prove overly verbose and would require further research so as to effectively summarize them without losing context that is crucial to understanding legal texts. This project was intended to be able to help everyday users with legal queries and in a sense, by presenting the most relevant laws, it achieves that, but comprehension is a necessary step which should be explored more in future studies.

IX. Future Enhancement

Improvements on the implementation of the search functionality is a possible path for improvement. Adjustments can be made to the search algorithm and methodology to provide a faster and more accurate response. Adding in a component that uses NLP to digest text-heavy responses to generate a summary, which were out of scope due to resource constraints, would be useful in generating more user-friendly responses.
Another possible improvement would be the generation of a library of key terms or phrases, or perhaps a measurement of tone that may denote severity, that may trigger the chat bot to advise the end-user to consult a legal professional. Sentiment analysis may be of use in this improvement.
Optimizations on data access and computation, as well as the search algorithm, may further improve responsiveness of the service. Most queries are handled through the webhook, the the authors would recommend leveraging Dialogflow more for the vetting. Given current system constraints, this would prove difficult but revisions and updates to the services used may change that in the future.

Acknowledgment

The authors would like to thanks Dr. Milton Halem for his guidance and sharing his knowledge on the concepts related to Data Science and Machine Learning, and also providing insightful comments on each research topic. His input was of great importance, and helped the authors at many decision points during the process.

The authors would also like the thank their respective friends and families. This paper would not have been possible without their support.

References

[1] Bayan Abu Shawar *Chatbots: Are they really useful?*, DBLP, December 2006.
[2] Justia US laws *https://law.justia.com*.
[3] K. Sparck Jones. *A statistical interpretation of term specificity and its application in retrieval* Journal of Documentation, 28 (1). 1972.
[4] Jana Bergant*DialogFlow fulfillment —dynamic responses from Google Firestore* https://chatbotslife.com/dialogflow-fulfillment-dynamic-responses-from-google-firestore-20acd19146ee
[5] Fouad Roumieh *Create a C .net core webhook for a Dialogflow chatbot*, https://medium.com/voice-tech-podcast/create-a-c-netcore-webhook-for-a-dialogflow-chatbot-e22d53c40d64.
[6] G. Salton and C. Buckley. *Term-weighting approaches in automatic text retrieval*. Information Processing Management, 24 (5). 1988.
[7] Lucene.net *https://lucenenet.apache.org*
[8] http://tfidf.com