Lucas Aust, Rae Jones, Christine Steege, Caleb Brunson, Jamel Chouarfia, Azariah Laulusa

Biplav Srivastava

CSCE 240

May 2, 2023

Disease Chatbot Project Report

INTRODUCTION

For our course project, our class had to work collectively to develop a collaborative assistant that offers useful information about diseases. Specifically, the program extracts data about a disease from WebMD and the CDC, processes that data, makes content available in a through a GUI, handles any user query, and reports on interaction statistics. A model that completes similar tasks is ChatGPT, which is “trained to follow an instruction in a prompt and provide a detailed response”, according to the OpenAi website. However, while our model provides the source of the information it uses, ChatGPT does not.

SOLUTION

The disease chatbot was coded in Java and currently supports 9 diseases: HIV, Scabies, Rabies, Rubella, Avian Flu, Malaria, Measles, Polio, and Mumps. The chatbot can detect and answer questions related to the information sections on the CDC and WebMD webpages for the supported diseases, requests for all the information found for a disease, multiple questions in a single prompt, and common small talk questions. It can also use the context of previous

questions to infer a disease if no disease name is provided and detects and corrects for simple spelling errors. To use the chatbot, the program can be downloaded and run as a .jar application on any operating system.

To complete this project, the main tasks and capabilities were divided into five separate components: data extraction, data processing, UI, user intent detection, and session and statistics logging. Once completed, each of these components were integrated to create the final disease chatbot.

To scrape the webpages of the diseases and extract the section information, the class DiseaseWebScraper utilizes an external library called HtmlUnit which provides an API that can invoke the webpages for diseases and parse the information by reading the HTML elements. This class parses the information from the page of any disease from CDC or WebMD and detects different information sections and prints these sections to a formatted text file.

To process the disease information, the class DiseaseDataProcessor reads the text files generated by the HTML parser To achieve this, I relied on the fact that my HTML parser from the previous assignment and checks each detected section against a set of regex patterns that are meant to capture the common information sections on CDC and WebMD pages for these diseases. The section information is stored in a map with the regex pattern serving as the key to make this information easy to access. This class can also provide an answer to a prompt using this stored information by simply checking the prompt against the same regex patterns to see if it could be answered with any of the information sections processed for the disease. Currently, if the same information section is identified from both CDC and WebMD, the text from the webpage that contains more information for that section is selected.

To handle user interaction, the chatbot uses a simple Java Swing GUI window to receive prompts and display chatbot responses. The window, ChatBotGui, extends JFrame and combines a text field for users to enter their message, a send button to send the message, and a text pane to display the user's message and the Chatbot's response. ChatBotGui also uses an instance of an interface, ChatBot, which simply takes in a String user prompt and provides a String response so anyone can easily supply their own ChatBot implementation to this class.

For this project, the implementation of the ChatBot interface was the DiseaseChatBot class. This class creates instances of DiseaseWebScraper and DiseaseDataProcessor upon instantiation and uses these instances to parse all the disease webpages and process the disease information. To generate responses to user prompts, it first checks if the entry matches some common small talk prompts and provides a response, and if the entry doesn't, then it checks for a supported disease name or uses context from the most recently mentioned disease to have the DiseaseDataProcessor attempt to generate a response to the user prompt with the processed disease information.

In order to detect user intent, the program relies on an established set of keywords that are related to each query and first "spell checks" the user input against these keywords. After correcting any words that were misspelled within a certain tolerance, regex patterns were used to check for these keywords and their order to determine if the user entry matched a query.

For spell checking and correcting user input, the class SpellingCorrector utilizes an external library called Jazzy to correct text. The Jazzy library relies on Lawrence Philips' Metaphone Algorithm and a form of the Near-Miss Algorithm to give the nearest suggestions within a certain tolerance of error for misspelled words based on a provided dictionary of words.

The “dictionary” used was filled with keywords from the prompts, so any keywords the user included in their utterance that were slightly misspelled could be corrected and replaced.

Session logging and data collection operates through a CSV file. When a user begins a session with the chatbot, system utterances, user utterances, date, and duration are all recorded. At the conclusion of their session, the data is then stored in its own line of the CSV file and marked with a unique serial number. A log of the chat is then stored in a txt file with the exact date and time it was taken as the title. After the session has been stored, information about the session is made available to the user through the GUI. The system can also use the columns of the CSV file to return data across multiple sessions back to the user.

When selecting components to include in the final product, we chose to base our project around a completed chatbot that we could make minor upgrades to. When deciding on a method, it was incredibly important for us to weigh the options with our strict timeline in mind. After taking this into consideration, we reasoned that the single-base strategy would allow us to complete our chatbot in the most efficient way while still implementing some new features. After concluding this, our group agreed that Christine’s chatbot had the best potential to be used as a base. From there, we were able to integrate user intent determination advancements into the Jar file to update the user experience.

EVALUATION OF SOLUTION

The fully integrated chatbot is currently able to handle user queries about nine diseases, each with two data sources. The chatbot can support approximately fourteen different question types for each disease. In order to add functionality for additional diseases, manual changes to

the DiseaseWebScraper class are required. Though this is fairly simple to do on the developers side, the code could be streamlined into an automatic process in future editions of the project. Additionally, reliable data scraping is only possible for data retrieval from the CDC's website. Our DiseaseWebScraper class often encounters issues when faced with the changing formatting of WebMD pages. Though it will almost always be able to process some WebMD data, the section delimitation that determines system responses is easily thrown off. This means that WebMD data may not always correctly match a question and answer pair.

After an additional disease's information has been correctly processed, information about it is made available to users via a GUI. In order to gain access to the GUI, users must download code from the project GitHub and open a jar file that contains the application. Though the integration of the GUI is an improvement from our system's original terminal output, requiring users to download vast amounts of code and data onto their devices is not ideal. If we were to create an updated version of the system, we would want to host the chatbot on a website to improve user experience.

DISCUSSION

The advent of ChatGPT has brought the usefulness of chatbots to the forefront of public discussion. Furthermore, the technology has a wide range of applications to both the private and public sectors. One of these applications is the dissemination of information regarding diseases. The COVID-19 pandemic illustrated the dangers of widespread misinformation regarding the spread of a disease and its treatment. The CDC and other online sources of medical information were carefully regulated for the health and safety of the public. Likewise, the chatbot we created,

combined with data from credible sources, serves as a powerful tool for informing the public about various diseases.

CONCLUSION

Our team has collectively built an interactive chatbot which responds to user input and provides valuable information on a variety of diseases including HIV, scabies, rabies, rubella, avian flu, malaria, measles, polio, and mumps. Furthermore, the program, following the design principle of being closed for modification and open for extension, allows coverage to be extended such that additional diseases can be supported. The sources of the data are CDC and WebMD articles for each of the aforementioned diseases. The stages of development which our team coordinated into a final program are data extraction, data processing, GUI implementation, user intent detection, and session logging with statistics.

Works Cited

OpenAI. “Introducing Chatgpt.” *Introducing ChatGPT*, 2023, https://openai.com/blog/chatgpt.