Lea Frost

Project Report

CSCE 240H - Spring 2024

Requirements

For this project, we were asked to create a chatbot that can answer questions about a company based on information from their most recent 10-k form. The project was done in five parts – a data extractor, a processor, a user interface, a user query handler, and a statistics calculator. These smaller projects were assembled to create a larger, functioning chatbot with multiple features. We could choose any two businesses and could choose between C++, Java, and Python for our programming language.

Specification

I chose Python as my programming language because I have never used it on a large scale, and it has many libraries and built-in features that would make this project easier and lessen development time. Additionally, many other classmates chose this language as well, making it easier to reuse people's code.

I chose Apple and Amazon as my companies because their 10-k structure was very similar, allowing me to use the same regex patterns for both without error. I tried a few other companies, but differences in formatting made the regex matching difficult. The regex patterns I used are hardcoded. In the future, I would improve my program to adapt to any 10-k, regardless of formatting. This would require a substantial amount of additional code to achieve, so I chose not to add that feature given the limited time to build the project.

Development Highlights

There is a main class that takes the user input and hands it to the back-end module to be processed. The back-end module utilizes the word matcher and session logger modules to process and log information. The main class also records statistics like time as well as system and user utterances that are then used to calculate statistics.

The 10-k forms are composed of four parts, each with multiple items. Each item contains a class of information, e.g. Item 1 contains information on the business, Item 2 contains information on properties, etc. For the processor assignment (PA2), I matched the user query to each of the parts via keywords. For example, if a user query contains the keyword “business”, it would match to “Item 1. Business”. Additionally, the program would know which file to read based on which company name appeared in the user query. If no business name and/or keyword is found, the user is prompted to ask again with the required information.

After the user’s information is mapped to the keyword, the item contents are read from the text file via regex patterns. The program starts reading when the beginning of the item is found and stops reading when the next item is reached. The contents of the item are then displayed in an output file for the user to access. This part was the most difficult, as it forms the bulk of the chatbot. It composes most of the back-end module.

Additional features were built upon this base in the following programming assignments. In PA4, I improved the user query matcher to make it more functional. It can handle minor misspellings from the user, e.g. if a user inputs “What is the comensatin of amzn's excutivs?”, it can understand that the user is asking about Amazon’s executive compensation. To achieve this, I calculate the edit distance between each of the words in the user query and the keywords. The edit distance for each word is divided by the length of the keyword it is being compared to to

calculate a percentage match. If this percentage is at least 70%, it is considered a match. If a keyword match is found in a user utterance, a match between the query and the item is made.

The word matcher functionality is in its own module for simplicity and organization. Because of this, it is easy to reuse. All you must do is import the module. I ran into some problems with the word matcher, specifically it would generate matches when it should not have. After some modification, I was able to prevent this from happening.

For the last programming assignment, we created functionality that could record statistics on the chatbot's performance. For each interaction, we must record the time taken and the number of system and user utterances. The chatbot can answer questions about the statistics. For example, a user can ask for a summary of a specific chat session, a summary of all chat sessions, or ask for the chatbot to display the contents of an entire interaction.

This functionality is contained in its own module. It contains methods for logging information in a CSV file, calculating statistics, accessing data from the CSV file, and outputting the information at the user's request. Because of this modularity, my statistics calculator is easy to reuse.

My final chatbot is a combination of these individual assignments. The processor uses the word matcher functionality to understand a user's query. The statistics calculator is an additional feature if a user is curious about the chatbot's performance.

I tested my chatbot by asking a combination of questions that relate to each part to see if it was able to generate the correct output. I also asked questions that were not related to supported keywords to see how the chatbot would handle them. I found that my chatbot was able to handle all user input as it was intended to.

Reuse

I reused Tarun Ramkumar's web scraper from the first programming assignment. I modified the web scraper so my chatbot will only web scrap the 10-k if the company's text file does not exist or is empty. This is to prevent an unnecessary increase in runtime. I attempted to build a web scraper at the beginning of the project but could not get it to work. Reusing Tarun's code helped me understand web scraping and made it easier to implement it in my program.

My code is very reusable as each major element of its functionality is divided into separate modules. Each feature can be reused with little modification, and each module is very generalized so it can be reused for many purposes.

Future work

In the future, I would modify my program to make it adapt to the 10-k itself instead of hardcoding the supported items and regex patterns. This could be done by reading the 10-k and retrieving the contained items and their individual formatting style, then generating regex to match. This way my chatbot could answer questions about any company regardless of their 10-k's formatting.