Quackbot: Your Coding Wingman

Jared Webb

College of Computing, Michigan Technological University

HU3120 Technical and Professional Communication

4/28/2023

Executive Summary

This report presents the design and implementation of a rubber duck AI chatbot, named Quackbot. The goal of the project is to create an artificial intelligence chatbot that is capable of helping software developers and software students debug code when they encounter an error. This software should use models taught during class and/or described in the Russell and Norvig textbook, *Artificial Intelligence: A Modern Approach*. The main idea behind the rubber duck method is the manifestation of the Self-Explanation effect. It works by explaining a problem in simple terms to eventually break through a debugging roadblock. The act of explaining a problem in simple terms can reveal solutions that were previously obscured. The chatbot uses natural language processing techniques to understand the user's problem and provide relevant feedback.

Contents

Program Description

The program is a chatbot with a command-line interface that is capable of understanding and responding to natural English language input from users, designed to provide simple solutions to frequent programming bugs/problems and assist with Python programming by leveraging existing CoNaLa training data and documentation.

The program is composed of four Python scripts, each with a specific purpose. The first script, parseIntentAndSnippet.py, preprocesses the CoNaLa training data and tokenizes the intents and rewritten_intents. Then, the script creates a bag of words representation for the intents and rewritten_intents using the Natural Language Toolkit (NLTK) and defines a function that determines the intent, keywords, and snippet of a sentence provided by the user. A bag of words interpretation treats collections of words as unordered lists. A bag of words is different from a set because frequency does matter.

The second script, dictResponse.py, contains a dictionary of regular expressions and their corresponding responses. When the user inputs a statement, the program uses regular expressions to match it to one of the patterns in the dictionary. If a match is found, the corresponding response is returned. If the response is a callable function, it is called with any captured groups as arguments. If the response is a string, any captured groups are formatted into the string using the .format() method.

The third script is formatPydoc.py defines the function format_pydoc. This function will take two parameters. The first is the name of the documentation module that the function will import. The second will select a specific function or subsection of that page to provide more specific information if possible. The second parameter is optional. The function creates, formats, and returns a string with relevant information from Pydocs if it's able. However, this function handles errors occurring from not finding the proper Pydoc page and handles them accordingly.

The next script, codeSummary.py defines functions that are capable of being fed Python code and summarizing it. Hopefully, by explaining misconceptions the AI has with the given code the programmer can understand better the techniques and methods they are using. Also, the program can explain and hopefully identify things that are occurring in the code that the programmer has a misunderstanding about or just does not understand. This second set of eyes is vital. These formatted strings of summaries are returned.

Finally, the last script, main.py, is responsible for interfacing with the user. It takes in user input, uses the determine_intent_and_keywords() function from parseIntentAndSnippet.py to determine the intent, keywords, and snippet of the input, and then passes the input to the respond() function from dictResponse.py to obtain a response. The program continues to prompt the user for input until the user enters the "EXIT" command.

One notable feature of the program is the use of CoNaLa training data to provide solutions to programming problems. The program can determine the intent of a user's input and use that intent to find the corresponding snippet from the training data. It can also use the training data to identify keywords in the user's input and use those keywords to provide a more specific response.

In summary, the program is composed of four Python scripts: parseIntentAndSnippet.py, dictResponse.py, formatPydoc.py, and main.py that use the Natural Language Toolkit (NLTK) to tokenize and create a bag-of-words representation of the intents and rewritten_intents, and regular expressions to match user input to predefined patterns.

Methods Used

The code consists of a collection of methods and practices from the field of natural language processing (NLP) and artificial intelligence (AI) that have been used to develop a conversational agent capable of understanding and responding to user input.

The core AI method used in the code is the use of a bag-of-words model to represent natural language text. In this model, a document is represented as a vector of its words, ignoring grammar, and word order. The use of bag-of-words allows the program to compute the probability of a sentence being related to a particular intent or topic (Juluru, 2021). The code uses the Natural Language Toolkit (NLTK) library to tokenize and preprocess the input sentences, which are then used to create the bag-of-words representation (Maeda, 2023).

The use of a machine learning algorithm determines the best matching intent for a given input. Specifically, the code uses a probabilistic approach similar to multinomial Naive Bayes, as opposed to Bernoulli or Gaussian, to determine the most likely intent by computing the probability that each intent and rewritten_intent is a match. However, instead of using the probabilities of keywords that are in the intent's bag of words, the method instead just uses the number of occurrences of a word. This means that more long-winded questions won't be punished by having lower probabilities because that would increase the size of the denominator of the probability. This also means the program does not need to use smoothing and take into consideration the words that were not included in the document. These changes make the classification process more efficient. The program identifies the intent behind user input in order to provide the most appropriate response.

Another important aspect of the code is the use of regular expressions to match user input with specific patterns. Regular expressions are a powerful tool in NLP and allow the program to identify specific keywords and phrases in user input. The program uses regular expressions to match user input to specific intent templates, which are then used to identify the best matching intent (Rul, 2019).

The program also makes use of pre-trained language models to perform specific tasks, such as formatting Python documentation. Specifically, the code uses the format doc library to format Python documentation strings into a more readable format (*Pydoc,* 2023). This library uses a pre-trained language model to extract and format the relevant information from the documentation string, allowing the program to provide a more helpful response to user input.

Finally, the program makes use of a simple rule-based approach to generate responses based on the identified intent. Specifically, the program uses a dictionary to map specific intent templates to appropriate responses. The program also uses a set of predefined responses to handle common user inputs, such as greetings or simple questions (*Rule-based Chatbots: Rule-based Chatbots Cheatsheet,* 2023*)*.

All in all, a range of AI and NLP methods were used to develop a conversational agent capable of understanding and responding to user input. These methods include the use of bag-of-words models, probabilistic algorithms, regular expressions, pre-trained language models, and rule-based approaches. The program is designed to be modular and extensible, allowing new intent templates and responses to be easily added as needed.

Conclusion

The program has significant implications for the field of artificial intelligence and software development. The ability to automatically debug code using natural language processing and machine learning techniques is an exciting advancement in the field and has the potential to significantly reduce development time and improve the quality of software produced. This program is simply an example of a broader trend in AI research, which involves developing systems that can learn from large amounts of data and use that knowledge to make predictions or provide solutions.

In the context of software development, machine learning techniques can be used to identify common programming errors, suggest improvements to code, and even generate code automatically. The program uses machine learning to analyze large amounts of code and associated natural language descriptions and uses that information to identify common patterns and errors. This approach has the potential to significantly reduce the time and effort required for manual debugging and to help developers produce higher-quality software more quickly.

The use of natural language processing is particularly important in this context, as it enables the program to understand the intentions and needs of the user. By analyzing natural language descriptions of code, the program can identify the user's goals and suggest solutions that are tailored to their needs. This can help to reduce the time and effort required to identify and fix bugs, and can also improve the overall quality of the software produced. While it has significant potential to improve the efficiency and quality of code development, there are also many other areas where these techniques can be applied. As such, it is important for researchers and developers to continue exploring new applications and approaches to machine learning to fully realize its potential.

A necessity, in this exploration, is the use of open-source data sets and machine-learning libraries. By building on existing resources, the program can leverage the knowledge and expertise of the wider AI community and benefit from ongoing research and development in the field.

By leveraging the power of machine learning and natural language processing, the program can analyze large amounts of code and natural language descriptions and provide solutions that are tailored to the needs of the user. While there are limitations and challenges to be overcome, the potential benefits of this approach are significant, and it is likely that we will continue to see rapid progress in this area in the coming years.

Works Cited

Juluru, K., Shih, H.-H., Keshava Murthy, K. N., & Elnajjar, P. (2021). Bag-of-words technique in natural language processing: A Primer for radiologists. *RadioGraphics*, *41*(5), 1420–1426. https://doi.org/10.1148/rg.2021210025

Maeda, J. (2023, March 16). *LLM ai tokens*. LLM AI Tokens | Microsoft Learn. Retrieved April 21, 2023, from https://learn.microsoft.com/en-us/semantic-kernel/concepts-ai/tokens

NLTK. (2023, January 2). Retrieved April 21, 2023, from https://www.nltk.org/

Pydoc - Documentation Generator and online help system. Python documentation. (2023, April 21). Retrieved April 21, 2023, from https://docs.python.org/3/library/pydoc.html

Rul, C. V. den. (2019, November 30). *[NLP] Basics: Understanding regular expressions*. Medium. Retrieved April 21, 2023, from https://towardsdatascience.com/nlp-basics-understanding-regular-expressions-fc7c7746bc70

Rule-based Chatbots: Rule-based Chatbots Cheatsheet. Codecademy. (2023). Retrieved April 21, 2023, from https://www.codecademy.com/learn/paths/build-chatbots-with-python/tracks/rule-based-chatbots/modules/rule-based-chatbots/cheatsheet

Russell, S. J., & Norvig, P. (2016). *Artificial Intelligence: A modern approach*. Pearson.