Pawlicki Collective: Advisor Dialogue Agent

NIKOLA DANEVSKI, BARTLOMIEJ JEZIERSKI, VLADIMIR MAKSIMOVSKI, and JUSTIN FARGNOLI, University of Rochester

CCS Concepts: • **Dialogue Agents**; • **Natural Language Processing**; • **Ontology Interpretation**; • **Data Mining**;

ACM Reference Format:
Nikola Danevski, Bartlomiej Jezierski, Vladimir Maksimovski, and Justin Fargnoli. 2020. Pawlicki Collective: Advisor Dialogue Agent. 1, 1 (July 2020), 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

We describe our advising dialogue agent, Pawlicki-in-a-box. This agent aims to compute accurate, precise, and actionable answers to an advisee's question related to choosing which courses to take during an upcoming semester. By mapping an ontology representation of the user input to SQL queries the agent is able to produce the desired output.

2 PROBLEM STATEMENT

We have aimed to develop an agent which can answer an advisee's questions about which courses to choose to take during the upcoming semester. One of our guiding principle throughout the chatbot's development has been to produce accurate, precise, and most importantly actionable results.

3 DESIGN

Our design consists of three independent modules; a query generator, a database, and a response generator. The query generator generates the fundamental elements of a specific SQL query which will answer the users' question. This SQL query is then executed on the database. The response from the database is reduced to its fundamental elements and given to the response generator. The response generator then generates a response and outputs it to the user.

3.1 Query Generator

3.1.1 Parser. In order to extract the most important information from the user's input, spaCy - a Python package was used. spaCy parses a sentence into a dependency tree where every token (word of the sentence) is tagged with a part of speech tag, lemma (basic form of the word) and its relationship with other tokens in the tree. This enables efficient extraction of the most important information from the sentence, such as its root, subject or object. This information is then used to create the database queries.

Authors' address: Nikola Danevski; Bartlomiej Jezierski; Vladimir Maksimovski; Justin Fargnoli, University of Rochester, Rochester, New York.

XXXX-XXXX/2020/7-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Most commonly extracted information from the input is ACTION which is a verb such as "describe" or "teach". Each sentence is marked with a question or declarative statement type and is usually assigned with subject and object(s). An example of

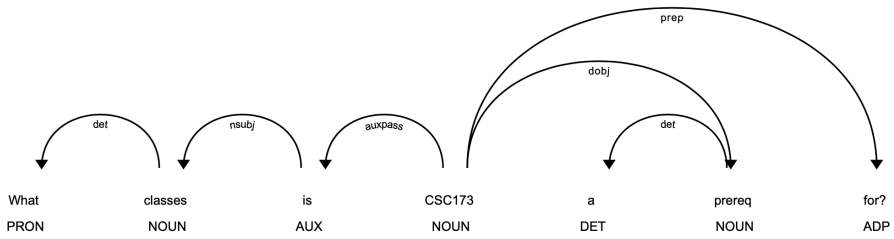

Fig. 1. What classes is CSC173 a prereq for? [(ACTION, describe), (SENTENCETYPE, q), (OBJECT, prereq), (SUBJECT, CSC173)]

After receiving the spaCy modified sentence, the next step we do is creating a query to pass to our database. This is done with sample sentences that students usually ask their advisors. We analyze the sentences according to the structure and their output from spaCy and generate queries accordingly. There are multiple queries that answer questions such as course prerequisites or classes taught by a professor.

3.2 The Database

Our database stores course offerings for the University of Rochester's Fall 2019 and Spring 2020 semester and students' ratings of University of Rochester professors. The former is constructed by mining University of Rochester's Course Description and Course Scheduler (CDCS) and cleaning the data such that it can be stored in a SQL table. A real strength of the Pawlicki bot is that it has information about every subject and every course in the last 2 semesters of the University of Rochester, which is many more courses than even Pawlicki can compare to. The latter is constructed similarly by mining https://www.ratemyprofessors.com to obtain every rating it stores for University of Rochester professors. These two tables are then stored into a persistent, read-only SQLite instance which is used to answer the previously generated queries. The rust program, chatbot, then interacts with this database via the rusqlite crate.

3.3 Response Generator

The response generator translates the resulting query from the database to human-like language. Similar to how we handle an advisee's input, we have map a our response to the user to a series of format strings which provide a natural and human like response to the user.

4 EVALUATION

In the case of the parser, the main difficulty was accounting for incorrect parses. In some cases, a token which is not a verb would be parsed as the root of the sentence. In other cases, more detailed relationship markers were determined for objects and subjects. In order to resolve this, we had to check for these edge cases. In building the working model, we first parsed an input into a dependency tree graph which helped us analyze the relationships between tokens. We worked on a limited set of input sentences to ensure that the increased number of incorrect or unexpected relationships encountered in parses for larger input set would not affect the functionality of the parser in relation to the most common inputs.

5 RELATED WORK

This work builds off of the work of many other related projects. Many other groups employed learning techniques to generate to answer a users question. This results in a model which is challenging to reason about. By using the ontology of the users input we are able to see deep into the meaning of the sentence and generate actionable responses. However, their work has inspired us to look at the efficacy of applying learning techniques to the job of mapping a users ontology to a query. One technique that other teams used was using GPT-2 to full-text-search over CDCS and the major websites. However, we felt that this wouldn't deliver actionable insight for the user, for something that they previously didn't know. It would probably have a better user experience, but our goal was for the user to learn something valuable, rather than create a chatbot of no practical use.

6 FUTURE WORK

This project can be improved by applying certain natural language processing and database design techniques.

6.1 Natural Language Processing

One drawback to our approach is that different verbs need to be handled individually when generating a query for the database. We believe that learning techniques could address this by predicting what query would best answer the users question, given the user's input. However, in order to do this we would need training data. Thus, future work could be done to generate training data for a model that would map an ontology to one of our concise representations of a SQL query. Another possible path would be to integrate standard NLP techniques to find and use existing code written for certain verbs to also be used when the users inputs a synonym of a verb we are able to handle. Finally, work can be done to evaluate the efficacy of using natural language processing techniques to produce less structured, yet coherent output for the user. Since the technique chosen for semantic parsing is not very robust under syntactic reordering (for example, "What is Gildea's rating" and "What is the rating of professor Gildea" are different in the eyes of the NLP parser), the user unfortunately won't be free to do many changes to the expected query formats. However, one easy feature that helped user interaction is the freedom for the user to use multiple synonyms for the same query, depending on how they prefer to express themselves. Since automated synonym finding is a difficult problem to handle, and due to the limit query formats, each word had its synonyms computed manually. So for example, "teaches", "educated", and "instructs" are nearly interchangeable for any of the user's inputs.

6.2 Database Design

Currently the database's structure is closely related to the structure of the JSON file which holes the raw content. Future work could clean the data further, such that the data is able to be represented by SQL types. Also, work can be done to design a concise and flexible BCNF database schema.

7 REFERENCES

- The Rust Book - https://doc.rust-lang.org/book/
- Nomicon - https://doc.rust-lang.org/nomicon/
- SpaCy 101 tutorial - https://spacy.io/usage/spacy-101#annotations
- Gildea's presentation - Slides on Blackboard
- SQLite Documentation - https://www.sqlite.org/docs.html
- ELIZA - https://en.wikipedia.org/wiki/ELIZA