Final Report: Calpass

I. Team

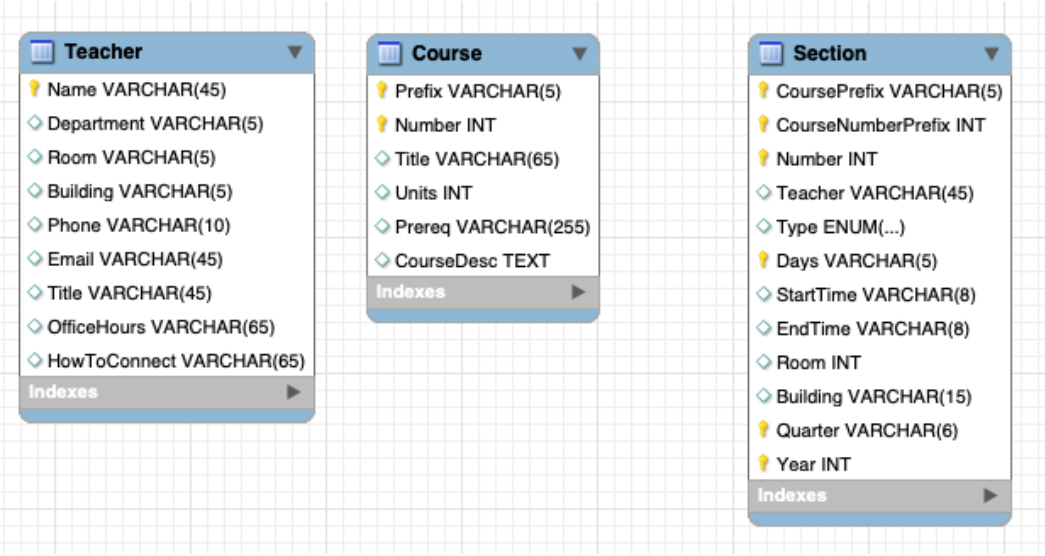
Anastasia Arsky (STAT) , Nicholas Wachter (CSC) and Joao Cavalcanti (CSC)

II. Work Split

The team tried to work together and collaborate for every phase of the calpass project. Below is a summarized description of each member's contributions. There is also our GitHub history that shows individual contributions.

A. Data Sustainer

Joao created the following database design, along with SQL files for creation and drop of tables. The database was implemented with pymysql[10], using MySQL 8 syntax.



Anastasia populated the “Teacher” table. She first scraped the office hours information for STAT professors from the Cal Poly Statistics Department Directory and Office Hours page[1] using BeautifulSoup[8] and stored them in a pandas[12] dataframe. Then, she scraped the office hours information for the CSSE professors in the link[2] provided using pandas[12] to both scrape (using the read_html() method) and store the data (in a dataframe). Finally, she scraped the List of College Instructors Teaching by Subject page for both the STAT[3] and CSSE[4] instructors using BeautifulSoup[8] in order to get their job

titles. These job titles were added as the column “Title” in both the STAT professors dataframe and the CSSE professors dataframe. She used requests[9] to fetch all of the aforementioned pages. The STAT professors dataframe and the CSSE professors dataframe were then cleaned using numpy[13] and merged. This resulting dataframe was used to populate the SQL database using pymysql[10].

Nick scraped the CSSE course catalog[6] and STAT course catalog[7] to populate the “Course” table in our database. This was done using the python BeautifulSoup[8], requests[9], and pymysql[10] libraries.

Joao populated the “Section” table by scraping the class schedule page[5] in order to retrieve data from different section offerings. He did this by using requests[9] (to fetch the pages), BeautifulSoup[8] (to scrape), re[11] (to parse data while scraping), numpy[13] (to perform operations on scraped data), and pymysql[10] (to connect to the database). Additionally, he reviewed code changes from other peers.

B. Bot

Nicholas created the tagging functionality to pull out keywords from the user input and put them into a map. Additionally, he transformed the original input by replacing key words with their variables. This was the preprocessing step before the questions were given to the classifier. For this, Nick used nltk[14] to tokenize the user’s input and then relied on pymysql[10] to fetch the data from the DB so the bot could match against it.

Joao worked on the implementation of the bot interface. In addition, he constructed the initial data pipeline, classification models and bagging implementation. For all of the mentioned tasks, the only package used was sklearn[15], where the TF-IDF vectorizer, Gaussian/Categorial Naive Bayes, K-Nearest Neighbors and Bootstrap Resampling algorithms were used. Also, he reviewed code changes from other peers.

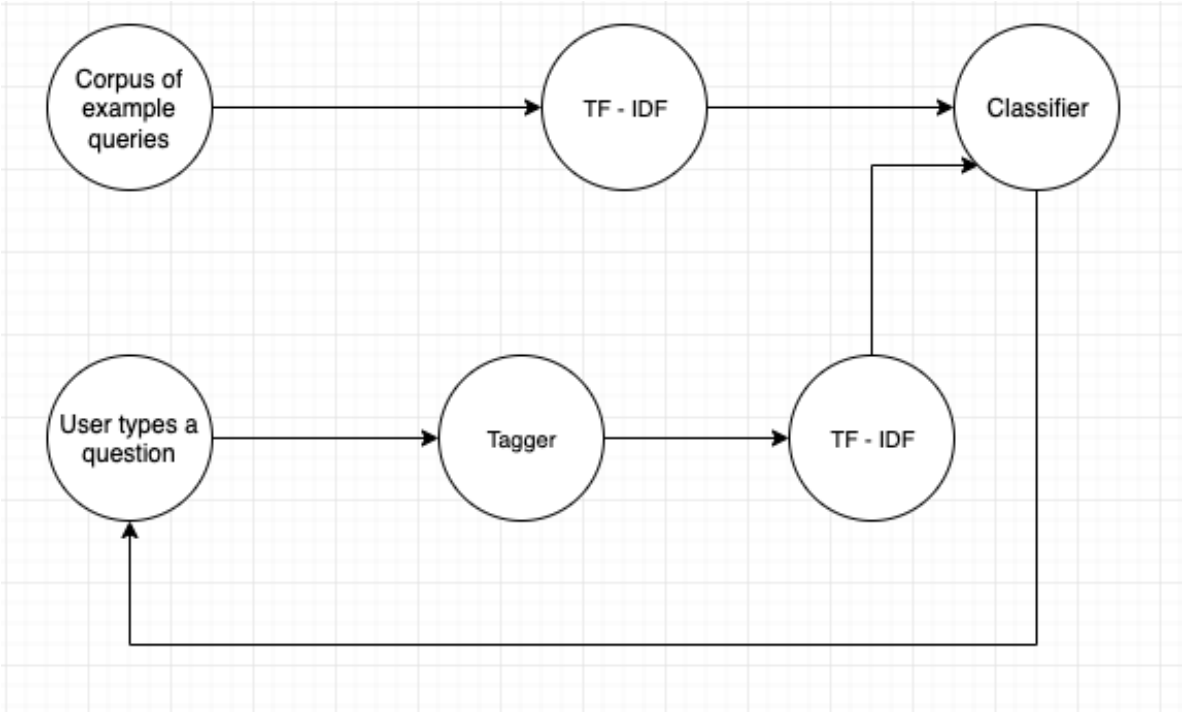
Anastasia worked on fetch_answer.py, which, given the intent class of the user’s question, as well as the list of tagger variables, returned the appropriate responses to the user’s question. It did this by using pymysql[10] to run different SQL statements depending on the intent class. The SQL statements used the tagger variables in their WHERE clauses.

III. Strategy

For the data sustainer, the strategy was first to design a database, along with the creation of SQL files that manage our DB (create and drop tables). Then, the team scraped all the

mentioned links and used the information gathered to create insert statements for respective tables in order to populate the DB.

For the bot, the strategy was to first create questions intents with their respective example queries. For instance, a question intent can be asking for a teacher's office hours location when a user gives a teacher's name. Then, when the bot spins up, it fetches data from the DB to get all variables, like teacher names. Additionally, the bot performs TF-IDF for the corpus of questions, where each document is a question intent and it includes many example questions from users. With that step, the bot feeds TF-IDF results into an implementation of Bagging, which, in turn, uses classification models seen in class, such as Naive Bayes and K-Nearest Neighbors, to classify a question into a question intent. In order to classify a question, we first process the query by tagging and replacing variable examples (like Fooad Khosmood) with their variable names (like [CSSE-FACULTY]), and then feed the processed string to a classifier. With both question intent and tagged variables, the bot calls a specific function that selects data from the DB and prints out a response to the user.



All development was done locally on each member's machine. The only difficulty was using Frank's MySQL engine while developing locally, but this problem was solved by using port forwarding when running the following command on another terminal shell:

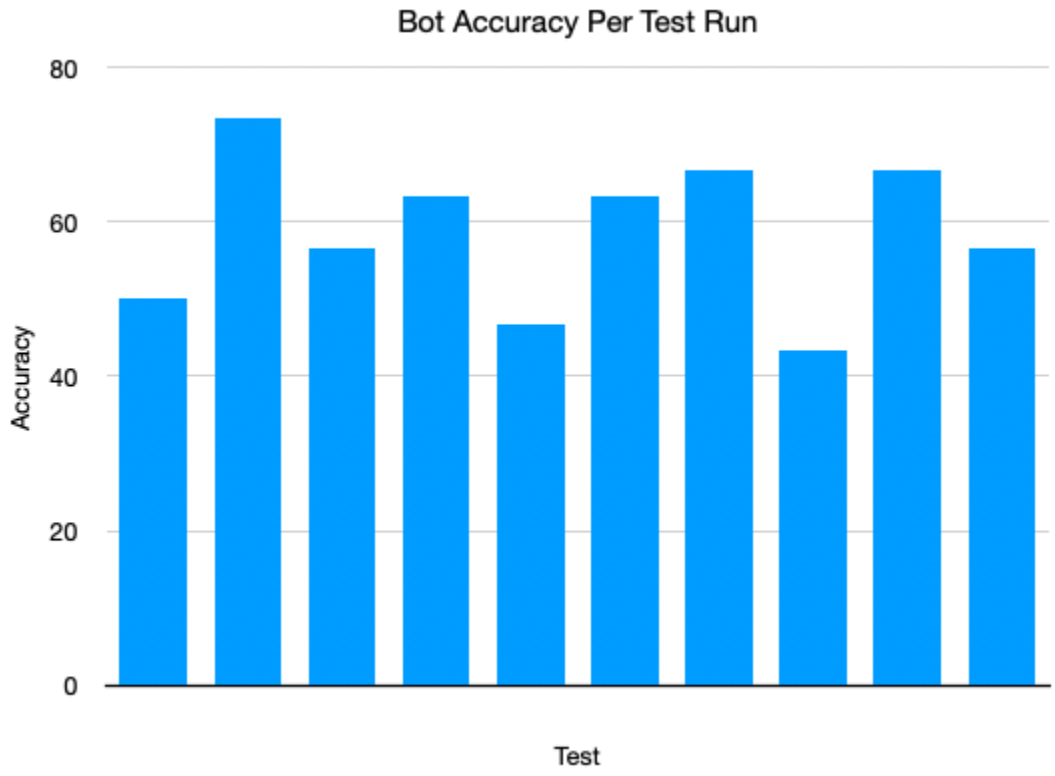
ssh -i ~/.ssh/466key.pem -L 9090:localhost:3306 user466@frank.ored.calpoly.edu

IV. Testing and Performance

We decided to write a total of 30 example user queries to test our bot's performance. Such queries were intended to cover all question intents and different phrasings covered by our bot. In addition, the queries also test the tagging of variable names and classification of question intents. Each test case consists of an answer and expected output from the bot. Since our bot's behavior isn't deterministic (i.e. sometimes it misclassified questions that were successfully answered in previous runs), we decided to run all test cases 10 times, and average the performance.

For a more detailed view of each test case, please refer to our GitHub repo. Here are the results:

15/30, 22/30, 17/30, 19/30, 14/30, 19/30, 20/30, 13/30, 20/30, 17/30
For an average score of 59%



V. Weaknesses

This section aims to describe weaknesses that couldn't be tackled due the time constraint of this project. Firstly, we are not supporting misspellings on our bot. Although we have the

implementation of levenshtein distance ready, we didn't get to actually integrating such a feature in our bot. Therefore, we rely on correct spellings to match variables against our database variables.

Furthermore, we expect teacher names to be completely typed. I.e., our tagger can't recognize just first or last names from user queries but it relies on having an exact matching against our DB. In addition, another shortcoming is the lack of optimization for different phrasings of questions with the same intent and similar question phrasings for different intents. Our TF-IDF implementation is structured as each document pertains to a single question intent string, which may include many different phrasings. We believe there is likely a strategy to optimize this procedure to avoid misclassifications, but we weren't able to tackle this task.

VI. References

[1]https://statistics.calpoly.edu/content/directory#:~:text=F%204%3A10%2D6pm%20Zoom,See%20Canvas%20for%20Zoom%20Link
[2] http://frank.ored.calpoly.edu/CSSESpring2022.htm
[3] https://schedules.calpoly.edu/all_person_76-CSM_curr.htm
[4] https://schedules.calpoly.edu/all_person_52-CENG_curr.htm
[5] https://schedules.calpoly.edu/
[6] https://catalog.calpoly.edu/coursesaz/csc/
[7] https://catalog.calpoly.edu/coursesaz/stat/
[8] BeautifulSoup package
[9] requests package
[10] pymysql package
[11] re package
[12] pandas package
[13] numpy package
[14] nltk package
[15] sklearn package