Eddie Aung
Kalen Goo
Kenny Lau

Project CalPass Report

Data Sources

All data was scraped using BeautifulSoup from Cal Poly's scheduling site. This website has the schedules of all professors who have taught and who will teach in each academic quarter. The data is grouped by their teaching courses. For this project, our focus was on the Statistics and Computer Science department for the Fall 2021 quarter.

Data Sustainer

Once the data has been scraped from the website, they were added to our Professor and Course SQL database hosted on the Frank servers. For each professor in the Professor database, we had the following columns: id (primary key), full name, alias, title, phone number, office, and course id. The course id is the primary key for a course within the Course database which also has the following columns: course code, section, type, days, start time, end time, and location. For both databases, there was some missing information for specific professors and courses which are represented with a "NULL". With both SQL databases created and filled, we can run the following query to get all the data into one database:

```
SELECT * FROM Professors INNER JOIN Courses ON Professors.courseID =
Courses.courseID;
```

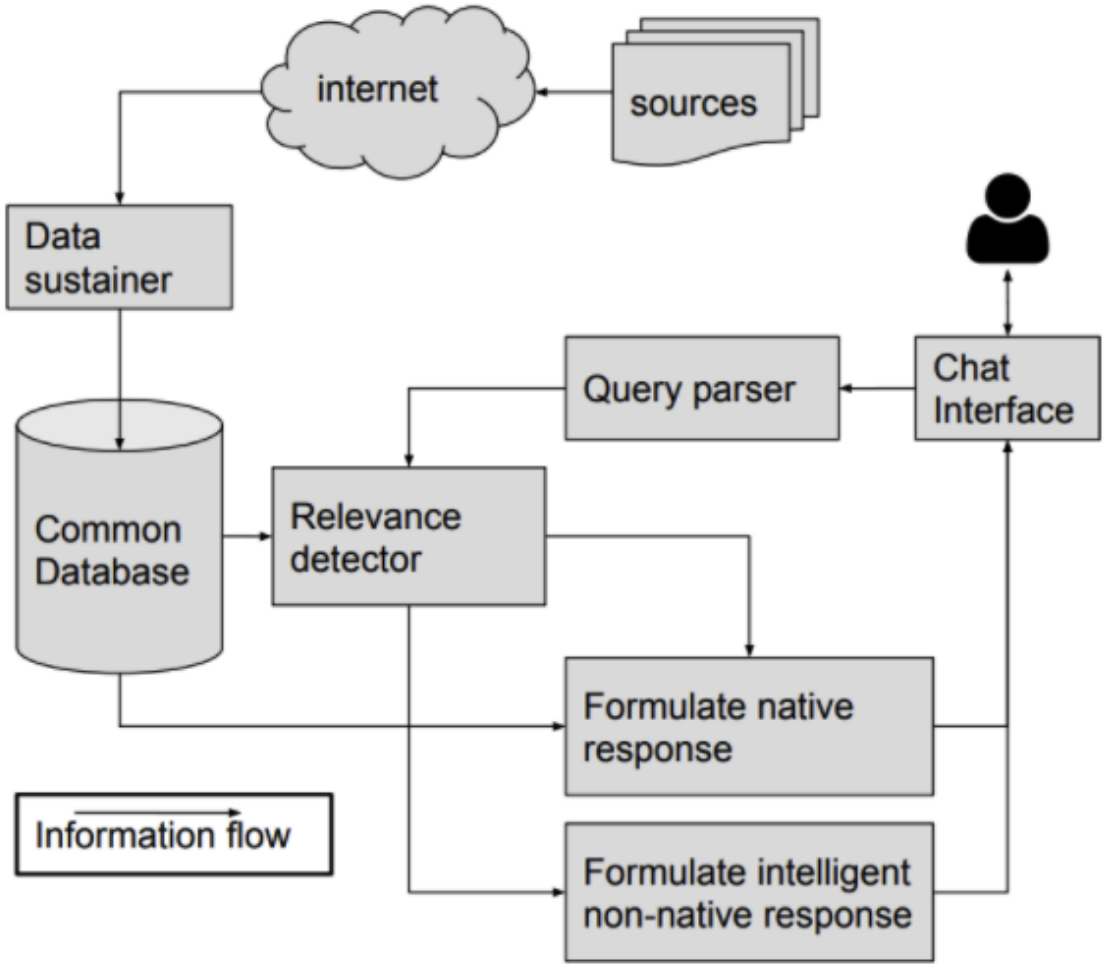
Here are a few example rows of the query output:

id	first	last	alias	title	phone	office	courseID	courseID	code	section	type	days	start	end	location
26	Christopher	Lupo	clupo	Dept. Chair Comp Sci & Soft Eng	+1.805.756.5659	014-0254B	26	26	CSC 596	18	Ind	NULL	NULL	NULL	NULL
27	Paul	Anderson	pander14	Instr Fac AY	+1.805.756.7178	014-0222	27	27	CSC 466	06	Lab	TR	03:10 PM	04:30 PM	999
28	Paul	Anderson	pander14	Instr Fac AY	+1.805.756.7178	014-0222	28	28	CSC 466	05	Lec	TR	01:40 PM	03:00 PM	999

Normalizing Data

We wanted to normalize the 863 queries that were combined from all the project groups. This was necessary because the format of each query was not consistent, decreasing our accuracy when training and predicting queries with our classifiers. By hand, we searched for each variable in the queries and replaced them with a consistent set of variables. Some queries were more complex so their variables were not in the scope of our variable set, but we still reformatted and kept them in our dataset.

Approach and Strategy



Here's a quick summary of how the chatbot works. Users start by asking a question through our command-line interface. Every word of the question is run through our `Entity Classifier`. Words that are classified as an `Entity` are substituted for a standardized variable and kept in a dictionary for the future query. Once all `Entities` are substituted, we pass the question into the `Query Classifier` to give the most likely answer format. Based on the answer format and the previously extracted `Entities`, a query is made to our SQL database. If the query is successful, a formatted answer is returned, otherwise, the user is asked to ask their question in a different way.

Example Entities are listed below:

- PROF: Aaron Keen, Dr.Keen
- COURSE: CSC 101, STAT 312

Let's dive into an example here:

An example question is listed below.

```
Who teaches CSC 101?
```

Running through the Entity Classifier returns:

```
entities = {
   COURSE: CSC 101
}
```

The entities are substituted with variables resulting in a question like this:

```
Who teaches [COURSE]?
```

Running this question through the Answer Classifier returns an answer format like:

```
[PROF] teaches [COURSE]
```

Using the answer format and entities, a SQL query is generated which looks like:

```
SELECT Prof FROM Professors WHERE Prof.course == entities[“COURSE”];
```

An answer is formatted with the entities and query result, which is returned to the user

```
Aaron Keen teaches CSC 101
```

Tools, Packages, and Libraries

In the building of the chatbot, several tools, packages, and libraries were used. They are described below.

Beautiful Soup:

Used for scraping data from the Cal Poly schedules website

Pandas

Used for data lookup and processing

Pymysql:

Used to interface with the SQL database from Python

Scikit-learn

Used for the two classifiers and some feature extraction:

- CountVectorizer
- TfidfTransformer
- NaiveBayes Classifier
- Cosine Similarity

Spacy

Used for Natural Language Processing and variable extraction

- Tokenizing
- Noun Chunking

Testing and Performance

Testing was separately done for two different parts of the chatbot:

- `Entity Classifier`
- `Query Classifier`

We based our `Entity Classifier` on the Multinomial Naive Bayes classifier that comes with the sklearn library, finding that our `Entity Classifier` performs very well when classifying Professors, Courses, and non-entities. The prediction accuracy for this classifier is 95%.

```
predicted = clf.predict(vectorizer.transform(df["Word"].iloc[175:]))
np.mean(predicted == df["Entity"].iloc[175:])
```

```
0.9545454545454546
```

The `Query Classifier` uses TF-IDF vectorizer to convert literal strings to numeric values, which are then compared with the cosine_similarity method to get related responses from the crowdsourced sample questions and responses. While some random questions are matched with a cosine similarity score of < 0.5, we found that most of the relevant questions get similarity scores of more than 0.7 and sometimes 1.0.

Examples of Queries That Work:

- What is Aaron Keen's alias?
- When are Dr.Keen's office hours?
- What time does CSC 202 start?
- Where is CSC 466?

Examples of Queries That Don't Work:

- Who teaches CSC 466?
- What classes is Hatalsky teaching?
- How many sections of CSC 202 are there?

Weaknesses or Problems Still Not Solved

While our chatbot can handle simple queries, there is much room for improvement. If there were more time given, here are some aspects of the chatbot to improve.

Standardizing Answer Formats in the Question/Answer Pairs

While we standardized variables in the aggregated questions, the answer formats could have been standardized as well. This would have allowed the chatbot to use a classifier with the label being the answer type instead of the current cosine similarity comparison. This change would reduce overfitting and provide recognizable responses to the person using the chatbot.

Handling of Multi-Entity Queries

Currently, the chatbot can only handle one entity or variable at a time. For example, if someone asked, “When does Professor Clements teach CSC 430?”, the chatbot wouldn’t be able to provide a meaningful answer. With variable substitution, the question would become “When does [PROF] teach [COURSE]. The SQL query would require a JOIN between two tables. To handle this, the SQL query module would have to be refactored to account for multiple entities.

Difficulty with Variable Extraction

Although there was an entity classifier, the potential variables still had to be extracted to pass into it. The challenge was that an entity could be a single word or a phrase. Simply tokenizing the question wasn’t enough. Several techniques were used including regular expressions and noun chunking, but there were still some edge cases that resulted in errors.