Project 3: StaCIA

Team H1 | CSC Courses

Brett Nelson, Michael Pangburn, Austin Silveria, Garrett Wayne

Data Sources

Cal Poly Course Catalog | http://catalog.calpoly.edu/coursesaz/csc/

- Provides a full up-to-date listing of CSC courses, including:
  - Course code (e.g. CSC 101)
  - Course name
  - Unit value
  - Terms typically offered
  - Prerequisites
  - Grading schema (e.g. credit/no credit only)
  - Description

Data Sustainer

Parsing

We use BeautifulSoup to first parse the Cal Poly Course Catalog into a Python data model (i.e. a Course class) for easier manipulation in preparing the database.

Database Schema

Our strategy for storing CSC course data in our SQL database was using two types of tables, one for relating a course code to its non-repeating information and multiple association tables for relating a course code to its repeating information in a lightweight way. This strategy allowed us to store the bulk of a course's information only one time in the main table, then using small tables to store listed data while duplicating the least

Table 1. Main course table containing core information.

Class Code	Class Name	Units	CRNC	Prerequisites	Description
466	Knowledge Discovery from Data	4	No	CSC 349 and one of the following: STAT 302, STAT 312, STAT 321 or STAT 350.	Overview of modern knowledge discovery from data (KDD) ...

Code	Department
357	CSC
357	CPE

Code	Term
466	Fall
466	Spring

Code	GE Area
302	B7
302	F

Code	Topic
466	data mining
466	classification

Fig 2. Lightweight association tables to store course information that may have different numbers of items.

amount of information. The main table shown in Table 1 contained core information that had the same structure across all classes, and the association tables shown in Figure 2 stored information that may have different number of items for different classes.

Topic Mining

A main goal of our bot was to answer questions related to class topics such as “What topics are covered in CSC 466?” and “What classes discuss data mining?”. Our strategy for this topic mining was to preprocess the text of course descriptions, then use rule-based selection to choose relevant topics across classes. We began by using NLTK to perform tokenization, stop words filtering (with custom stop words added such as “lecture”, “laboratory”, etc.), and lemmatization. This normalized text allowed us consolidate topics that may have slightly different wordings in different class descriptions. An inverted index was then built to map topics to classes they matched as shown in Table 2. We then removed topics with the following characteristics:

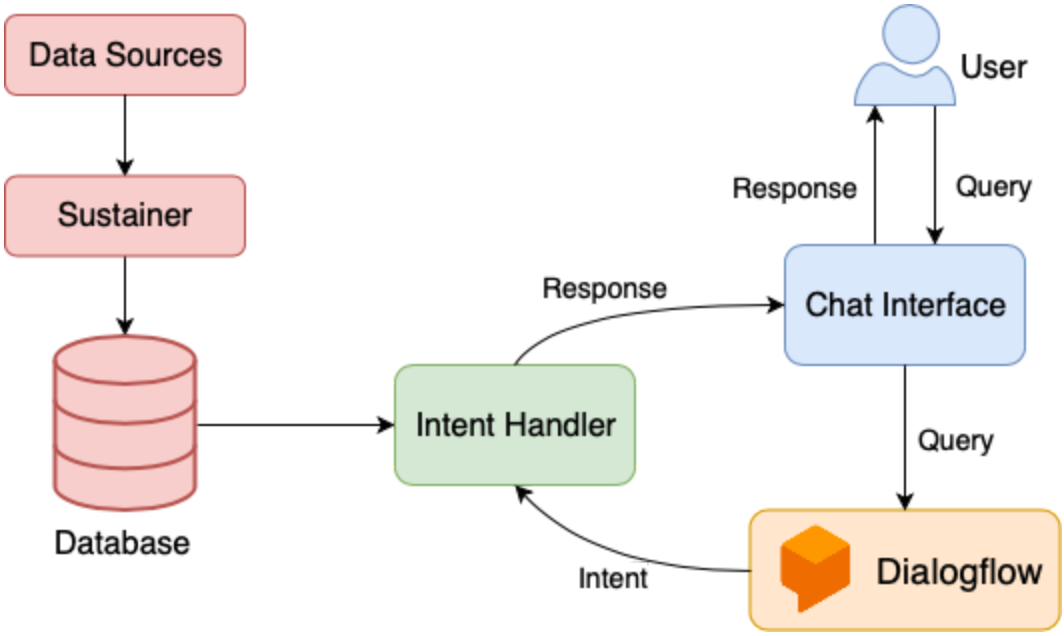
1. Mentioned only 1 time
2. Mentioned more than 8 times

Topic	Classes
data mining	466, 482, 566

Fig 2. Inverted index mapping topic to classes they matched.

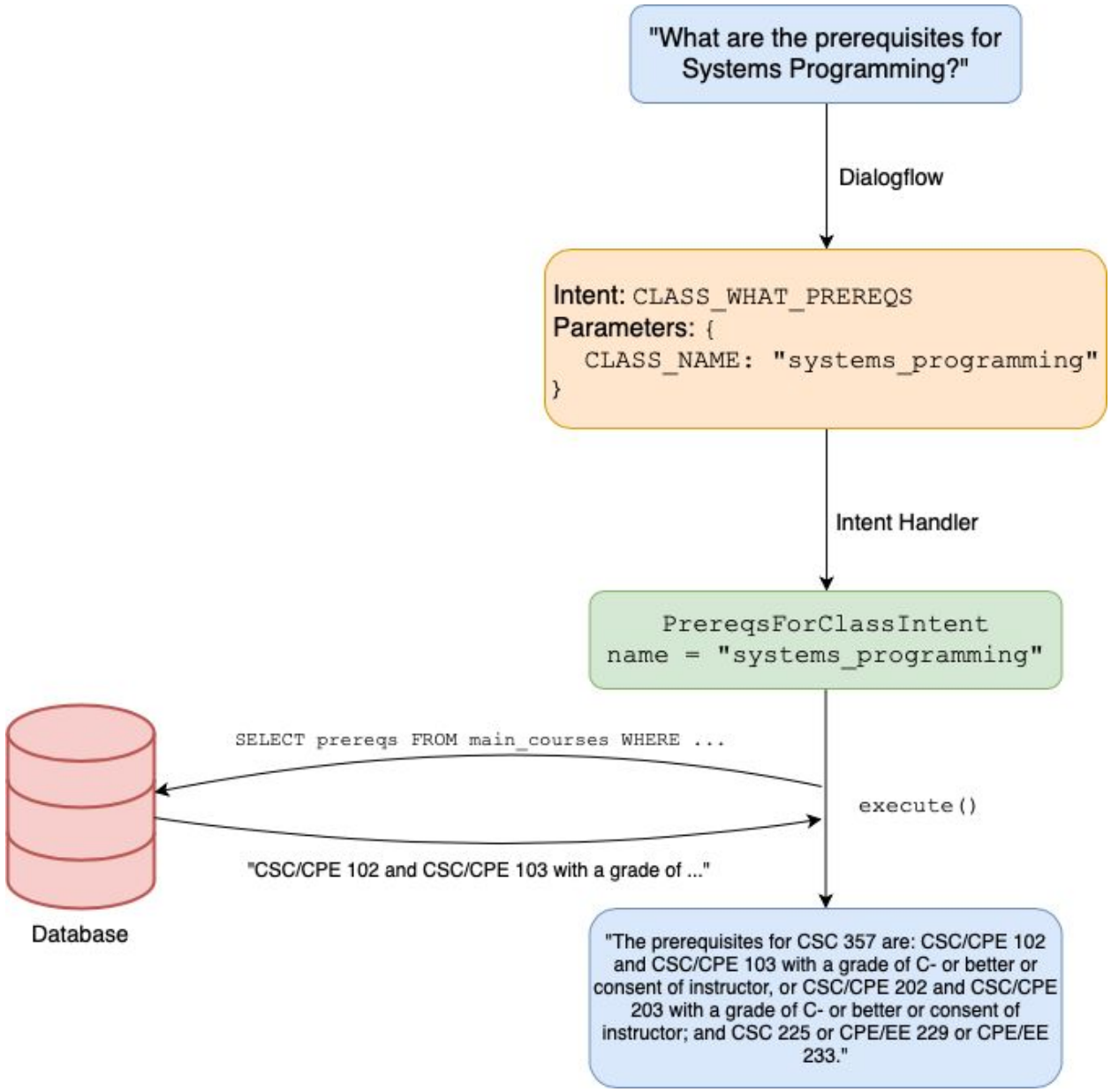
These rules served as an effective proxy for removing “topics” (unigrams, bigrams, and trigrams from descriptions) that were grammatically incorrect or simply common phrases that appeared in most of the descriptions. After tuning these rules and filtering the topic list, we ended up with topic lists for each course that were fairly accurate.

Strategy



0. Prior to use of the chat interface, the data sources are parsed and mined for information via the sustainer to populate the database.
1. The user enters a query into the chat interface.
2. Via an HTTP POST request, the query is passed to Dialogflow, a web-based Google service that uses machine learning to parse the user’s intent from the query, along with any pertinent parameters.
3. The returned, parsed intent is passed to the intent handler, which maps the intent to its corresponding Python class.
    a. The intent instance performs a SQL query to retrieve information from the database based on the intent’s parameters.
    b. The retrieved information is formulated into a response string.
4. The response string is displayed back to the user via the chat interface.

The following diagram presents an example of the transformations applied to the user’s query to produce the response string:



Tools, Packages & Libraries

Beautiful Soup

BeautifulSoup is used to parse the HTML of the courses web page into a Python data model.

NLTK

This package provided a number of text preprocessing tools to facilitate topic mining from the course descriptions including tokenization, stop words filtering, and lemmatization.

PyMySQL

MySQL's Python client allowed us to programmatically store all parsed information in our database.

Dialogflow

We used Google's machine-learning-powered chat bot building platform "Dialogflow" to handle intent recognition and entity extraction. When a user submits a query through one of our chat interfaces the query is first sent to our bot on Dialogflow in a web request. Using the knowledge gained from being trained on user query examples, our bot is able to recognize the user intent and appropriate query parameters (aka entities) then send these back to us in the web response.

Bing Spell Check

The first thing that takes place when a user queries our stacia bot is a spell-check of the query. Using the Bing Spell Check api from Microsoft Azure, we can go through each word in the query, see if that word is likely to be a typo (using a set threshold for typo-likelihood), and change the word to what the user was trying to say. The new string, free of typos, is henceforth used as the query.

Irc.bot

The Irc.bot library was used to create an IRC bot that would exist on a freenode channel and take in (and respond to) queries from users on that channel. We used this library in a manner similar to how we used it in Lab6; to create a running instance of an Irc bot and provide it with responses to specific user queries. Other than help, usage, and greeting queries, all of this bot's response content comes from pinging our Stacia bot.

Testing & Performance

Using Dialogflow's web interface allowed us to iterate rapidly while training the model. If a phrasing of a particular question failed to parse, we could add additional rephrasings to the model and retrain quickly on their servers. Meanwhile, we each kept a terminal

window with our chatbot open on Frank, and we could continue asking questions after retraining without needing to exit the process. This workflow allowed us to develop our Dialogflow model quickly while retaining fast response times, as the bulk of processing is done on Dialogflow servers.

Weaknesses & Unsolved Problems

Intent Mapping

One shortcoming we had with the intent mapping of the user question was the original choice we had of using wit.ai to parse the intents. The original reason for using wit.ai was because of the simplicity that it offered us, as we can simply train the bot on questions and rephrasings mapped to specific intents and it would act as a classifier as so. The software that is wit.ai has since been outgrown by newer machine learning technologies and for that reason, we began using Google's Dialogflow and found much more success.

Second Interface

One shortcoming with the implementation of the second interface to host the Stacia chatbot was not being able to fully use the extent of Dialogflow's Fulfillment feature. This would have enabled our model to fully give responses from our live database back to the user in the web app console or on our google assistant. It would have required a web server such as apache to be hosted with our source code on Frank, allowing for POST requests from Dialogflow with the intent and corresponding parameters, returning a response that would be generated locally from our intent handler and sent back to Dialogflow.

Topic Mining

One shortcoming of our topic mining implementation was the wide reach of topics that remained in our topic lists even after filtering. Even with text normalization and rule based selection we still ended with topic lists that included some grammatically incorrect and irrelevant "topics". To improve the relevance of our mined topics in the future, we could introduce more natural language rules to ensure more grammatically incorrect "topics" are removed, as well as part of speech tagging to find topics in the context of full sentences.