Nick Visalli
Sean Donnelly
Nick Winner

1. ## The overview of your chat bot

   The chatbot is composed of an Angular frontend and a Python Flask backend. When a user types a query in the chat input and presses the enter key, the client sends a request to the Flask endpoint. The backend uses RASA to interpret the query. After the query is interpreted, our service sends the query to the Twitter api. Our api returns returns 10 tweets to the client. The client displays the tweet text and photo is applicable.

2. ## What approach your group took to analyze/vectorize the query?

   First, we composed our own training data based on the use cases for our chat bot. This training data was written into a simple markdown file, where we identified intents that the user might have, entities that may exist in a message from a user, and datetime parsing rules.

   Then, we fed this file into a script to define RASA-based JSON documents that define our data and query relations. From here we were able to kick off a model training pipeline that took the constraints we defined and worked with existing models, analyzers, and tokenizers that are integrated with RASA to generate a model ready to analyze messages from the user. In particular, we leveraged the english version of the SpacyEntityExtractor.

   At this point we had a functional data pipeline that we plugged in as an intermediary data processing step in our APIs request-response lifecycle. The message from the user comes in, goes though our analysis pipeline, and then is formatted into a query to the Twitter API.

3. **What approach your group took to collect the data**

   After running the user's query through our model to extract the entities and other relevant aspects of the query, we sent a request to the Twitter API over HTTP. This request originated on our server application. After receiving a response from Twitter, the response payload is forwarded down to the client application running in the user's

browser.

4. **What method was used to compose the response?**

   We defined a JSON schema in our server application that our client was built to parse and create UI elements from.

5. **The integration and/or limitations of your chat bot**

   One limitation that we faced during the development of this application was fitting all of the tweets inside of a chat window. In order to overcome this issue, we decided to only display the text of the first 10 tweets found. Tweet photos are also displayed if they exist.

   We also found that with such a generalized chat domain, we weren't able to train our model well enough to go too far off script. In order to extend this and add functionality we would need to retrain our model.

   However, the largest complication that we faced was the use of the Twitter API itself. The querying mechanisms that the API exposes do not allow for robust filtering and sorting on a specific user. When using the free tier of the API, this becomes even more detrimental because the default will return very few tweets - this makes post-processing almost useless.

6. **Anything else you would like to add**

   It would be much more helpful to the user if they could chat with the bot before receiving a response. A fun feature to add would be the bot asking the user some questions to understand their intent and context more clearly.

   Entering the word "clear" in the input field clears all of the messages.

   It would also be helpful if the chatbot displayed information about the tweet author. The chatbot should also display a link to the original tweet.