Training a Virtual Assistant to Return Relevant NBA Information

Timothy Flannagan
Student at UML
Lowell, Massachusetts
timothy_flannagan@student.uml.edu

Danny Abou-Chakra
Student at UML
Lowell, Massachusetts
danny_chakra@student.uml.edu

Rushabh Doshi
Student at UML
Lowell, Massachusetts
rushabh_doshi@student.uml.edu

ABSTRACT

Virtual assistants, or chatbots, continue to grow in popularity in the past decade or so. This can be attributed to both textual communications becoming the preferred way of social communication and large companies such as Microsoft, Facebook, Google, IBM, and more creating frameworks for others to use. The main purpose of this project was to create a chatbot that's able to return applicable, real-time information while being able to adequately interpret the intent in a user's question. Using the RASA open-sourced artificial intelligence framework, we were able to manage the dialogue flow between a user and the chatbot, while incorporating machine-learning and natural language processing techniques.

AUTHOR KEYWORDS

Virtual Assistant, Artificial Intelligence, Chatbot, Natural Language Understanding, Natural Language Processing

INTRODUCTION

The purpose of this paper is to walk through the development of our NBA Chatbot, providing an explanation of the design decisions and an overview of natural language processing, and the challenges that come with it. The RASA AI framework allows a developer to expand a chatbot's capabilities from answering simple questions, to more complex ones due to the separation of its core engine and its natural language understanding (context) engine. More specifically, the core engine deals with dialogue management, allowing the developer to explicitly specify how a user may interact with the chatbot in the form of "stories."

LITERATURE REVIEW

METHODOLOGY

Initially, we used the NLTK (Natural Language Toolkit) Python API to help manage the natural language understanding portion of this project. This library encapsulates text processing libraries, large data corpus', and different interfaces for the developer to chose from. When using this API, we were able to have more control over how the user's input was lemmatized, tokenized, tagged, and parsed, eventually feeding those results into NLTK's text processing libraries. This was proved to be a poor design decision from the start as we had an inadequate natural language processing background and had to adapt to a new design. After researching different engines and frameworks, we were able to settle down on the RASA framework, which abstracted the lower level details into easier to use libraries. As mentioned earlier, the RASA AI framework is divided into two components: the dialogue management engine, and the NLU engine. In Figure 1, we see a high-level overview of how the RASA components interact with each other in order to process a user's input. One component listed in Figure 1 is the policy and that determines which action to take at any point in the conversation (cite rasa core docs policies). For the interpreter (NLU) component, we use the spacy-sklearn pipeline, which abstracts some of the difficulties with using spaCy, and the KerasPolicy and MemoizationPolicy that are integrated within the RASA AI engine. We also wanted to use 200 epochs (one full training cycle on the training vector to update the weights) and a max history (how far into the dialogue model training histories we look into order to determine which action to take next) of 3. We defined all of these values in the policies.yml file.

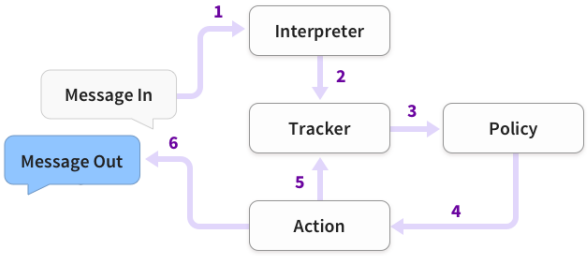

Fig. 1. Overview of RASA Components

Now our main focus in the developmental stage was to map out different intents with examples of how a user might phrase the question. For example, if we had the intent: querying Wikipedia for player information, we would then need specific examples of that intent such as "who is Kyrie Irving", or "Kyrie Irving", and place all of those values into a markdown file to use later in the training phase. Furthermore, we may define entities, such as player, team, conference, greeting, and more, and specify those entities in different intents in that same markdown file. So now the example of "who is Kyrie Irving" now becomes "who is [Kyrie Irving](player)",

Fig. 2. Interpreter returning JSON output

where the square bracket notation is an example of the player entity. This helps the NLU training policy extract entities from a user's input. After populating an intent with multiple examples of entities (we used the NER_CRF entity extractor), we were able to train the NLU component (both unsupervised and supervised) and run the interpreter to ensure the intent extraction was performing well enough to start training the dialogue portion of the project.

As we can see in Figure 2, when the user inputs "who is Kyrie Irving", we get a JSON formatted dictionary containing entities, intent, the user's text input, and an intent ranking. First, we can see that the NER_CRF entities extractor correctly extracted the entity "player" and associated that with the value "Kyrie Irving". Further down in the output, we can see the intent ranking key with a list of confidence and specific intent mappings, where the text input "who is Kyrie Irving" was correctly associated with the intent of querying the Wikipedia API with a confidence rating of 84%. It's also important to note that we decided on a confidence threshold of 60% or better before taking a non-text, custom action (i.e. if the user's input was "hello" or some form of greeting, we would reply with a text response instead of defining an action class in Python.)

Once we populated the chatbot domain with more intents, such as finding out when a team plays their next game, getting the eastern conference standing information, etc., and intent/entities extraction was above our confidence threshold, we started the development on the dialogue management component. Here, we used the RASA core engine, and now needed to create a markdown file of "stories", which essentially tells the dialogue model how to behave by defining possible flows of conversation. This is a very powerful feature, as there can be missing "slots" in a user's input that are needed in order to take some sort of action. For example, if the user wanted to information about a player, but didn't specify which player, the chatbot would need to ask the user for the player's name (the slot) before proceeding. Once the dialogue system is built up, we can assign an intent with either an "utterance", which is a pre-formatted response (usually text, but can include thing such as images), or a custom action defined in Python. In order to define custom actions, we need to make use of the Tracker object as seen in Figure 1, which keeps track of conversation state, receiving the information from a new user message (cite this. rasa core docs arch.) We needed this Tracker object is order to get the user's last text input or the associated intent of that input in order to use the objects in the NBA_api library.

RESULTS

Currently, the chatbot is able to do a couple of things fairly well. For starters, we were able to implement many different intents in the chatbot's domain. When the training stage ended, our model was able to correctly identify these intents with a confidence above our threshold, which was exciting. When a user asked a question where the entity wasn't specified in the domain, it was still able to determine the intent correctly after both supervised and unsupervised training. Additionally, the chatbot was able to run the correct action associated with the user's intent. The problems that arose during development were more geared towards configuring the response back to the user, rather than the overall infrastructure of the chatbot.

In the project proposal, we intended to use this Python library called "NBA_py", which configured the stats.nba.com endpoint and returned that information in different formats (JSON, panda arrays, and more.) When it came to the developmental stage of defining our custom actions to take, we didn't realize that the GET requests from this API would mainly result in hanging (never timing out) requests. This was clearly problematic as the library was well managed until the summer of 2017, but since that time period, it was poorly maintained. In conjunction with that time period, the NBA started changing their internal API and the NBA_py endpoints weren't updated accordingly. We then needed to amend our design decisions, and find another Python library that handled these endpoint configurations or risk doing them ourselves, which would prove to be too time-consuming. After finding another Python library called "NBA_api", we were able to resume the implementation of these custom actions. At the time this project was due, we were able to get JSON information back from the stats.nba.com, and return that block of JSON back to the user, but failed to parse the JSON correctly and output a more appropriate response.

On another note, we planned on hosting our chatbot infrastructure on Facebook Messenger, allowing the user to interact with the chatbot in a more polished interface. We were able to correctly construct the webhook to the Facebook Messenger API using a Node.js application, and get the user's text input when they sent a message to the chatbot on Facebook Messenger. The problem was that we needed to find a way to return the chatbot's response to the user's question back to the web-hook, but weren't able to find a way for the Python and Node.js programs to communicate with one another. We

felt as though this was an important feature to implement as now the chatbot is able to interact with users around the world and therefore would encounter different inputs than what we might have asked it during the testing phase.

Discussions

Improvements

For the future, we could implement many different features that could expand the capabilities of this chatbot, allowing it to interact with the user more fluidly. First, before implementing any new intents, we would need to get more comfortable with the NBA_api, which would allow us to use the internal methods that formatted the data in a way that we could use. After parsing that JSON, we could extract the information we needed, and return to the user the information that they requested, making the chatbot have actual use. After that, we could continue implementing new intents, while continuing training our models to handle user input that's not yet specified in its domain.

Another big feature we would eventually like to implement is finding a way for the Python application to communicate with our Facebook Messenger page. Additionally, once that it's implemented, we would also need to host our application on a server or instantiating an EC2 instance on Amazon Web Services, instead of hosting it locally using ngrok.

The interaction between the software and user should flow naturally, similar to other artificial intelligence systems, such as Alexa, Siri, or Google Voice Assistant. With a firm foundation, we would expand the variety of functionality. Similar to querying past NBA statistics, a query of predictions could be possible. With past player data, such as points, shooting percentages, playing minutes, it is very much possible to train a logistical model that could predict future stats. Another expansion would be querying dates for future games that could be relayed to the user at the appropriate times. This could translate to being imported into a portable application on the phone in the future. For the project, it was meant to run on the terminal. Hosting on a social media site such as Facebook would have been appropriate to show off during the presentation, but in the future for any alpha beta releases, a release of a compact form of phone application is ideal. Additionally, to fully emulate an artificial intelligence system like Alexa, Siri, or the Google Virtual Assistant, we could also add the ability to communicate with the chatbot verbally, implementing this feature using pre-existing voice-to-text software and dumping that text input into either stdin in the terminal or whatever medium we decide is applicable.

Conclusion

Our task was to create a chatbot that interacts with the user in a closed-domain environment, returning relevant NBA information based on the user's input and correctly identifying the intent or context of that input. Most of our effect during this project was looking through RASA AI framework and the different NBA Python libraries that were used at some point during this project. While moving towards using the full RASA AI framework proved to be a good adaption in the development process, due to its vast capabilities and abstraction of NLP techniques, this meant dedicating most of our effort towards reading the source code of RASA and figuring out how to best use its components. This, in turn, took away time that could have been dedicated to working with the NBA_api and having a more polished response from the output of the custom actions back to the user.

Overall, we were able to successfully create the foundation of this chatbot, while still allowing room to grow in adding more features, such as expanding the list of defined intents. We were unsuccessful in returning structured responses back to the user and needed more time in order to fully understand the NBA_api library and the Tracker and Domain objects in RASA. In further releases, we specified some of the things that we want to tackle in the improvements section, most importantly interacting with this chatbot in a different medium, like Facebook Messenger or a mobile application.

Acknowledgment

The work described in this paper was conducted as part of a Fall 2018 Artificial Intelligence course, taught in the Computer Science department of the University of Massachusetts Lowell by Prof. Jonathon Mwaura.

References

Please number citations consecutively within brackets [**?**]. The sentence punctuation follows the bracket [**?**]. Refer simply to the reference number, as in [**?**]—do not use "Ref. [**?**]" or "reference [**?**]" except at the beginning of a sentence: "Reference [**?**] was the first . . ."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [**?**]. Papers that have been accepted for publication should be cited as "in press" [**?**]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [**?**].