Machine Learning Decal Hackathon - NLP Track Spec

1 Project Description

For this particular hackathon track, you will work to implement a chatbot for note-taking. This chatbot will be given text input, will return text responses, and should have a persistent memory such that you can take and retrieve an arbitrary number of notes. We have subdivided the functions that the chatbot must support into two groups: basic and advanced. You should first focus on ensuring that the chatbot can correctly respond to basic commands, and then try to implement as many of the advanced commands as possible. These are the basic commands that the chatbot must support:

- **Start to take a note**. This corresponds to a command that indicates that the user would like to take a note. For instance, the first user command in the following example corresponds to a "start to take a note" command:

  User: Hey, please take a note for me.
  Agent: OK, what would you like me to say?
  User: Remind me to buy oranges tomorrow.
  Agent: Alright, I've noted that.
  User: What was my last note?
  Agent: Your last note was: Remind me to buy oranges tomorrow.

- **Take a note.** This corresponds to a command immediately after a "start to take a note" commmand. For instance, the second user command in the above conversation is an example of this.
- **Retrieve the last note**. This corresponds to a command that indicates that the user would like the previously-written note to be read back. For instance, the third user command in the above conversation is an example of this. Your system should also be able to respond correctly when there is no last note.
- **Delete the last note**. For example, "Hey, go ahead and delete my previous note."
- **Total number of notes**. For example, "What's the total number of notes that I have?"

Now, for the advanced commands (roughly ordered in increasing difficulty):

- **Retrieve / delete a note by number**. This corresponds to a retrieval / deletion of a particular note (not necessarily the last one), where the number of the note is given. For example, "Please delete the second note that I took."
- **Retrieve / delete a note by facet**. This corresponds to a retrieval / deletion of a particular note, where we reference a note by its content. For instance, the second user command in the following conversation is an example of this.

  User: Remind me to buy three oranges tomorrow.
  Agent: OK, I've noted that.
  User: What note did I take about oranges?
  Agent: That note was: Remind me to buy three oranges tomorrow.

- **Take a note with content**. In the basic commands, we told the agent that we wanted to take a note, then we told the agent the content of the note. Here, we do both steps in a single user query, as in the first user command in the above conversation.

2 Starter Code

The starter code is located on the class GitHub at `www.github.com/mlberkeley/Machine-Learning-Decal-Fall-2018`. There are two files: *conversation.py* and *main.py*. You will only need to edit *conversation.py* in order to have a working solution.

The *conversation.py* file defines the `Conversation` class; every group of back-and-forth interactions between the user and the agent will happen after a new `Conversation` is created. This allows you to save the past user queries / notes, such that

your agent can properly retrieve notes when asked. The only interface that your `Conversation` class needs to support is the *respond* method. This method accepts a user sentence, and should return the corresponding response from the agent. Currently, the method simply repeats the input back, but you should change this to support a proper response.

The other file, *main.py*, is used to test your agent. To do so, simply run *main.py* from the command line – This will create a new conversation and allow you to input a sentence. The corresponding output from the chatbot will be printed to the command line. Optionally, you may supply a *--dir* argument to *main.py*: This allows you to test a specific conversation without manually inputting the sequence of queries. Try running *python main.py --dir exampledir* to see what it does on the example conversations that we've given.

3 Hints, Tips, and Ideas to Explore

The project is pretty free-form; we will simply score you on the quality of your responses to a held-out test set that we have prepared. You may employ any methods and libraries you like. That being said, we outline here a few different tips and approaches that may be interesting to explore in order to help you get started.

A common approach to this chatbot problem is to recognize a particular *intent* when given a user query. For example, the query "Please take a note for me" might have a *start_note* intent, while the query "What was my last note?" might have a *retrieve_last_note* intent. This allows us to view the problem as a classification problem, the exact same sort of thinking that we would use when classifying digits in the MNIST dataset, for example. Hence, we can use many of the classification tools that we've explored in this class, such as SVMs and word embeddings.

Note also that certain intents require *entities*: Additional objects in the input query that are relevant to the chatbot. For example, in the query "What was the second note I took?", the word *second* might be an important entity that we care about. You should think about how we might recognize these entities. This is a good way of thinking about the problem, but here are a few more practical tips to help:

Note that the user can add words such as "hey" and "please" that don't really change the meaning of the input. In the NLP world these are called *stop words*: Words that are not semantically meaningful. You should try to curate a list of stop words and remove them from the user input before classifying the intent.

A helpful package will be *nltk*. This package includes lots of helpful tools, such as a list of common stop words. You can also explore (by looking at the NLTK docs) more advanced preprocessing techniques such as *stemming* and *lemmatization*. In addition, NLTK provides access to *WordNet*, a thesaurus-like map of every word in the English language that will be helpful for looking up synonyms / other word relationships.

Good luck! Remember to start with the basic commands, and only work on the advanced commands afterwards. You won't need fancy trained models for the majority of the project (most everything can be done with WordNet / word embeddings).