Spoken Language Processing - Voice Twitter App

System Description Report

May 9, 2012

Yufei Liu, Aneem Talukder

Functionality of Domain

Our domain functions on a set of Twitter-based queries. One can ask the system to read a specific set of tweets by giving the system a number of parameters such as time, users, and topic.

There are four different modes for our system. They are main menu, read, write, and interact with a tweet. For this, we implemented four parallel grammars. This helped reduce the errors of the system because each state is insulated from the others. There is less confusion that could possibly come from the ASR interpreter.

In main menu state, one simply chooses which state they want to move to, or if they want help or to exit. They can say something like "HELP ME" and “EXIT”, respectively.

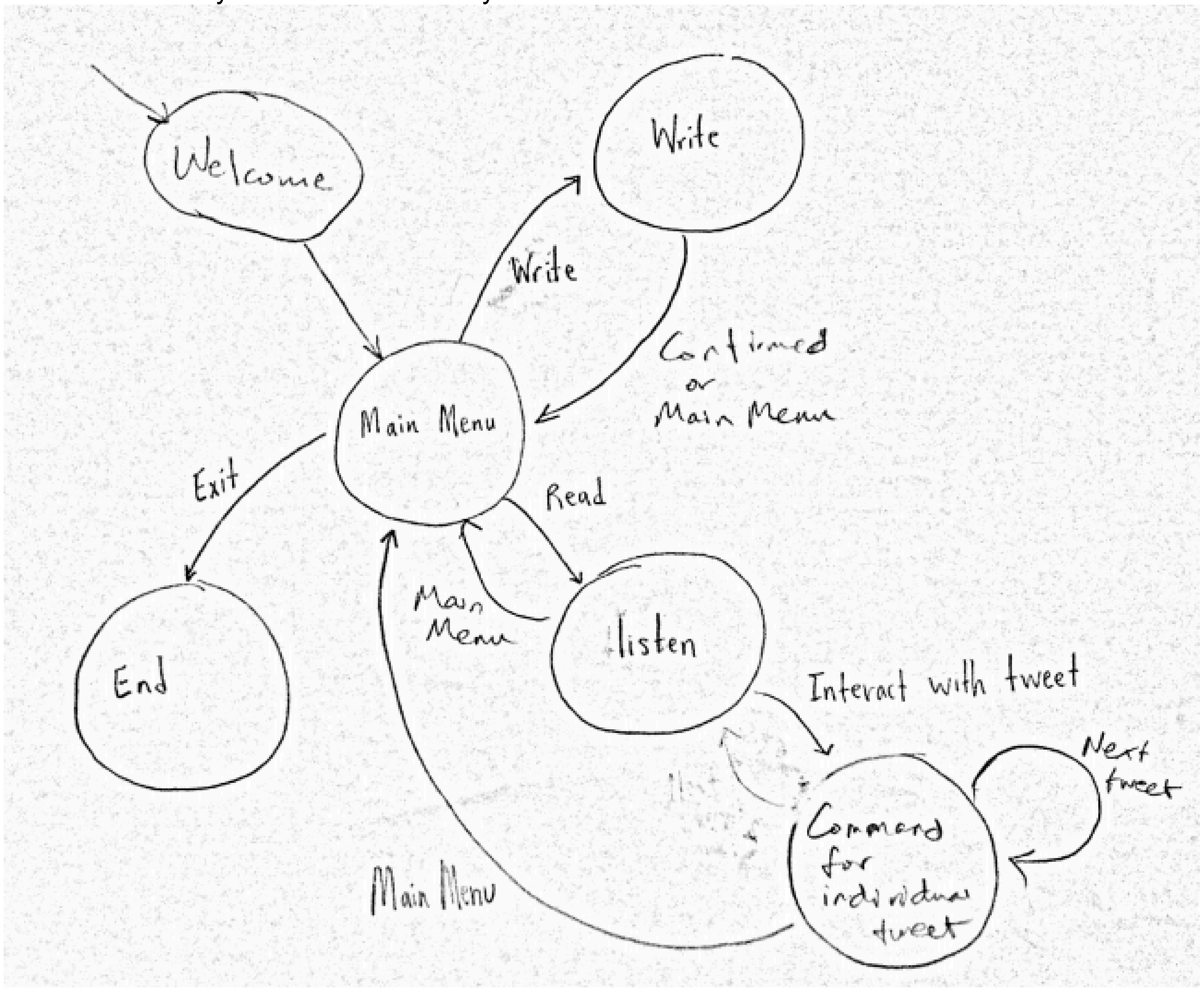
In the read state, one can ask the system to read a set of tweets. The user can preconfigure the account they follow into different groups such as "friends", "family", and "colleagues". Then the user can ask to be read tweets specifically from users from those groups. Moreover, the user can specify the time of those tweets- he or she can ask to be read tweets from an hour ago, for example. An example of a read query: "READ tweets IN MY TIMELINE in FRIENDS in the LAST HOUR." Instead of reading tweets in timeline, user can also request tweets with a search term. We recognize the following five: POLITICS, ENTERTAINMENT, SPORTS, NEWS, and TRENDS. TRENDS is a special term because our system will actually look up trending terms, and search these terms instead.

After the system accepts the user’s query, the system first gives an overall summary of query results, generated by the TTS. This outlines roughly how many tweets are in the query set, and when is the most recent tweet. For instance, the system may say “there are more than 10 new tweets since you last checked, the most recent is from a few seconds ago”. After this summary message, the system proceeds to read each individual tweet.

After a tweet has been read, the SDS moves into the tweet interaction state. In this state, a user can perform up to two actions on the tweet that has just been read by the system. For example, after a tweet has been read, a user can both FAVORITE and RETWEET. The second action is optional. The user can also say “follow this user” or “unfollow this user” as actions.

The final state is the write state. In this state, the user can tweet a preconfigured set of tweets. These include emotion tweets such as "I am happy! :)". The user, for this, can say something like "tweet that I am HAPPY." Other possible tweets include a random fact from reddit.com or the weather, which we pull from Google's weather API.

A summary of the states of our system is illustrated below:



Modifications to ASR and TTS

ASR:

Initially we had three parallel grammars in our ASR for the three different states of Twitter interaction. When it came to the SDS system, however, we needed a way for the user to get to each of those specific states and so we implemented the main menu mode which got its own grammar. In this state, user is able to navigate to other states.

Also, we added the option for the user to go back to main menu from every other state. This is done by making the sentence “MAIN MENU” an acceptable sentence in all grammars.

Likewise, we also added “HELP ME” as valid sentence to all grammars, such that user can request the help message to be played at anytime. This is useful because our system by default only plays help messages once.

TTS:

Our TTS has largely remained the same. We did, however make some more prerecorded prompts to smooth out the user experience with the SDS. Originally, these static prompts were part of the TTS. We factored them out into static wav files to ease the development process (so re-recording a prompt does not require re-recording the entire TTS system). Some examples of these include "Welcome to Voice Twitter, what would you like to do?"

We also realized that most of the system output in a typical dialog is actually done in the default voice because they involve arbitrary tweet text. While there is no trivial way to amend this, we did do some preprocessing to tweet texts to make the speech as natural-sounding as possible. Examples include replace the hashtag symbol # with the word “hashtag”, and normalizing URLs to be more pronounceable.

We also recorded prompts such as "tweeted/retweeted/favorited successfully!" This is a confirmation for the user to know that his or her action was successfully executed.

Trade-offs

One tradeoff we made was making our help recordings very detailed. We felt that we wanted to give instructions and then an example of how to use those instructions to form a query. We compromised this by allowing the user to use the system without any help. All they have to do is to add the flag "--nohelp" as a command line argument when starting the program.

Even when the system is in the mode of playing help messages, we opted to play help messages in each state only once. Namely, if user leaves state 1 and revisits it later, no help will be provided. This is done to save user time so they don’t have to sit through long messages again. Of course, we provided the flexibility for user to request help messages anytime by saying “HELP ME”.

Another tradeoff that we made was to not use confirmation methods. We felt that it would make the dialog too clunky and slow. Instead, by splitting the system into four grammars, we tried to reduce the amount of error as much as we could. We also decided to use help messages instead. That way, if a user becomes lost, he or she could ask for help and an appropriate set of instructions would be read. Also, since our system consists of several states and transition between states is easy and readily available, we feel that user can easily recover from errors, without the need of explicit confirmations.