Jake Muir
Karl Bell
Toshihiro Kuboi
CSC 580

Chatbot Project

For this project, we implemented a chatbot based on the “Sheldon” personality. The bot is able to handle a typical greeting exchange, and responds to messages by talking about related subjects (he is not interested in what anyone else has to say!). He also interrupts others’ conversations to tell them related facts.

The Main Loop

Our bot has two main functions: dealing with people talking to it, and interrupting the conversations of others. In the case of interrupting others, the bot looks at any conversation that other people in the channel are having, attempts to find some subjects they are talking about, looks up a subject related to one of those, and begins stating random facts about it. If it can’t find anything interesting to say, it moves on to the next message it sees between others. In the case of someone talking directly to the bot, it will respond with a simple greeting exchange, and then wait for the other person to mention anything in particular. Once they do mention something, the bot uses the same procedure as the interruption case to determine some related subject to talk about. If it unable to find something interesting to say, it generally makes fun of the chatter for being interested in the subject at all.

Performing Greetings

We implemented the complex greeting functionality of our chatbot as a finite state automaton. The state transition table was implemented as a dictionary of dictionaries each of which contained at least one pair of an input and a transition state. We searched certain words pertaining to greeting using regular expressions in the inputs. All phrases that we think normally used in greetings were expressed in the regular expressions.

One of the challenges we faced with the implementation of the chatbot was how to express greeting words in as few regular expressions as possible. For example, there are many ways to ask how someone is doing: “How are you?”, “How are you doing?”, and “How is it going?” All starts from “How” and end with “?”, but not all of them include you in them and verbs used are different. We ended up creating one regular expression for “How are you?” and another regular expression for “How are you doing?”, and “How is it going?” Creating regular expressions for the reply to the inquiry “How are you?” was more mind boggling because a person can reply in any way he or she wants. He or she may reply as “I am good”, “Pretty good”, “Ok”, “Fine”, or “I am doing all right.” We could not find one regular expression that covers all, so we ended up hard coding possible replies like “good|fine|ok|all right”. Looking back we could have tried part of speech tagging and classified all adjectives as a reply.

One trap we fell in with the regular expression is that the regular expression “hi” matches every word that has “hi” in it. When some bot replied, “I do not know anything about that.”, our bot took it as a greeting because the regular expression matched with “hi” in “anything”. So, we had to make sure that the regular expression was written so that the position “hi” occurs was specified correctly.

Another difficulty we faced with the implementation was with the functionality of the outreach. It took some time to figure out how to create an event to trigger the outreach. The problem with the timer is that when testing the bot, waiting several seconds before the bot makes an outreach seems very long, but to make it look natural incorporating some delay between chats is very important.

Picking Out Subjects

In order to determine the subjects in the given message, the bot attempts to pull noun phrases out of the message it is looking at (dropping determiners and adjectives). This is done by POS-tagging the message using a NLTK tagger trained on the Brown corpus, then passing this on to a NLTK Regexp chunker which uses a chunking grammar we wrote. This chunker only pulls out the noun phrases from the sentence. We can then look up these noun phrases on wikipedia to find a related subject to talk about.

Finding Related Topics

We implemented a function to find related topics from the Wikipedia article of the subject. The function retrieves the data of article from Wikipedia using its API, and picks first 50 links from it. Then we randomly pick one of the links and check if there is a link to the original subject page in the first 100 links in the page. This is to make sure that we pick an article that is related to the original subject. If the linked page has a link back to the page of the original subject, the function returns the link with the sentence that includes the link. The number of links picked was chosen mainly because we think that links appear at the beginning of the article are more related to the subject than the links found at the bottom of the article. And also we want to limit the number of links we pick for the performance reason.

Once we have found a related subject (and the sentence which connects it to our original subject), we pass these on to the fact-finding function.

Fetching Facts

The fact-finding function takes two inputs to correctly produce a list of facts. It needs the original subject and the new subject. The new subject is used to pull the plaintext data from Wikipedia. Once the new subject wikipedia page is pulled, that data along with the original subject are used to generate a list of facts that appear on the new subject’s Wikipedia page but are related to the original subject. In order to generate the list of facts the fact finding function takes these steps:

1. Pull plaintext from new subject’s Wikipedia page
2. Parse the page’s data and split it up by section

3. Store each section with its data in a dictionary
4. Iterate through the dictionary by section adding sentences to a fact list if the original subject appears in the sentence
5. Return the fact list.

There is some extra processing involved to disregard sections that contain useless data in terms of facts such as “External Links” and to only return sentence facts that end in a period or an exclamation point so the facts aren’t questions or phrases.

The actual identification of determining whether sentence facts are related to the original subject is pretty simple because it just checks for the presence of the original subject within that sentence. We attempted to do a more complicated way of finding facts related to the original subject through parsing and chunking but it performed worse than our simpler solution. We tried using NLTK’s named entity chunker as well as NLTK’s Regexp chunker for noun phrases but we were unable to identify more subject related facts than our simpler solution. As a result, we decided to stick with just checking for the appearance of the subject within the sentence for producing our list of facts.

What to Say?

Once we’ve got our related subject, sentence connecting that subject to the original subject mentioned in the message, and a list of facts about the related subject, we just need to decide what the bot should say. First, we have him say the sentence connecting the original subject to the related subject, preceded by some connecting phrase like “Did you know that ” or “Interesting fact:“. This makes the transition to a related subject seem natural, and, in the case of a direct conversation, makes the bot seem uninterested in what the chatter actually has to say. After that, the bot randomly selects half of the facts retrieved for the subject (up to a maximum of 5) to say. It enters these into chat at a rate directly proportional to their character length. It then assumes the chatter doesn’t actually care, outputting a disparaging message like “Oh, it’s lost on you, anyway.”

In every case above where we used hard-coded text (besides the greeting), we provided a number of choices of what the bot should say. It randomly chooses from the set of whatever kind of thing it needs to say, in order to reduce repetition and make the bot seem more human.