LING 550 - HermioneBot Final Report

Victoria Feere - 260405806

April 10, 2015

Introduction & Brief Description

For the final project I wrote a domain-specific chatbot that takes as input a Harry Potter related question and replies with a predicted best response. In order to do this I had to leverage articles from an online wiki to formulate a response based on bulk textual data. When interfacing with the wiki I focused on simply narrowing search topics given the user's question and forwarding the topic on to the wiki to obtain which articles are most likely to contain pertinent information. This information formed the minimal answer and underwent a series filters to obtain a more refined response.

When running the application it is advised to follow the *Happy Path* in the README for an optimal demonstration of the application. However, you should feel free to play around with various syntactically structured questions, spellings and general Harry Potter questions.

You can checkout my repository on github here:

https://github.com/vfeere/HermioneBot

Motivation

The reason I chose this particular project was because it seemed to be an interesting cumulative exercise applying my knowledge from computer science, linguistics and what I know of NLP from this course and my work experience at Nuance Communications. I was primarily interested in applying linguistic techniques to analyze queries and determine user intent. This included performing POS-tagging, parsing parts of an English language syntax tree and predicting relevant response phrases given loosely structured questions.

Survey of Existing Work

A lot of research has been done on part of speech tagging, natural language grammars, parsers and programmatic spell checking. Tools already exist in nltk for many of these tasks. References exist on the web, including papers such as this one: http://nlp.stanford.edu/ manning/papers/tagging.pdf. Likewise, there are many useful tutorials online like this one written by the head of research at Google, Peter Norvig: www.norvig.com/spell-correct.html.

Data & Tools Used

This application is in the form of a desktop application. It was written in python. I used nltk to perform word tokenization, POS-tagging, NP-chunking and parsing on all user input and part of the system formulated response. I used the built-in python urlib2 & json modules paired with an external wiki API (whose documentation can be found here: http://www.wikia.com/api/v1/) to fetch results from the remote Harry Potter wiki.

Implementation

General Infrastructure

The first task was to implement the basic setup necessary to perform API queries using the python json module and retrieve data from the online wiki. Next, I built the basic GUI components for the desktop application. For this I used Tkinter, python's built-in GUI builder. This created an interactive text area for user input and an output text area for the system's response.

Understanding User Input

When a user inputs a question the first task is to perform word tokenization and POS-tagging. This was achieved using nltk's built in data libraries, particularly the Penn Treebank POS Tagger. Upon transforming what was first a string into a list comprised of (word, POS) pairs it was then necessary to identify the type of user input phrase based on it's syntactic structure. If the user asked a question then a list of identified key queries would then be selected to be sent to the wiki and obtain relevant article ids. To achieve this I wrote a short grammar using regular expressions defining simple NP, VP and PP's based on the Treebank parts of speech. This was then used in combination with nltk's RegExParser to "chunk" NP, VP and PP's into a less robust syntax tree than one would expect given an explicit English language grammar. The loss of strictness was one of the biggest trade-offs of the system but it turned out to be useful as it allowed for the most concise parts of the question to be sent to query the wiki. This improved the chances of obtaining matching articles.

Once a tree had been generated it was decided, based on where the WH-NP fell (if it existed), whether the user had asked a question or not. The system additionally accepts queries starting with an auxiliary verb, 'Is Gryffindor the bravest house?', for example.

In general, I chose NPs for querying and retrieving relevant articles. I identified particular verbs and prepositional phrases as keywords, helpful for ranking relevant sentences in an article. When the article text is returned from the wiki each sentence is scored based on many different measures such as the number of occurrences of a particular NP normalized over the number of words in the sentence. A sentence's relevance score is boosted when it is found to contain particular keywords. The system returns the 1-2 sentences (depending on length) with the highest score.

Additional Features

In addition to identifying questions, querying the wiki and obtaining a "best" response, the system performs a spell check and is self-aware. If you ask Hermione a question about herself, for instance, 'Who was the love of your life?' she will reply in first person. I also defined default responses and response prefixes so that the answer would seem more authentic, as if it were actually Hermione you were speaking with.

As for the spell checker, I used Peter Norvig's online tutorial as a general guideline and incorporated my own compiled .txt file as the vocabulary to compare against. The spell checker makes use of the minimum edit-distance algorithm given the transpose, insert, delete and substitute operators.

Future Work & Improvements

The most important aspect to note is that the system as a whole is extensible and could be applied to a variety of other wikis to create chatbots specific to that particular domain. The few features which were rooted in Harry Potter knowledge are easily interchangeable and would require simply replacing the vocabulary text file required for spell checking. Also, the few lines of code which create a "self-aware" bot by replacing phrases in 3rd person with their respective 1st person pronouns would need to be replaced with the new bot's persona. Additionally, all pre-defined strings included in the system response would need to be tweaked to be specific to the new domain. However, all these aforementioned aspects of the software application are modular and therefore easy to refactor.

With respect to the linguistic computations, such as parsing user input and identifying relevant keywords, more work could be done to probabilistically predict the most important branches in the question parse tree. It would be interesting to explore the impact of preserving word-order when ranking the most relevant response strings. Something which I did not have time to implement but would have liked to was a word-similarity mapping in order to create a more competitive ranking amongst response candidates.

Conclusion

In conclusion, I am satisfied with what I was able to accomplish given the time and resource constraints. I feel the solution is general enough to attain decent results and any additional work would be to optimize and improve upon what has already been sketched out by the current solution. Were I to continue development my focus would be on creating word-similarity mappings and understanding the topology of question phrases so as to better understand the most important selectors when looking for a response in the text.