11-411 NLP Semester Project
Progress Report 1

Team Hydra: Jemmin Chang, Eileen Jiang, Ashley Lai, Rohan Varma

February 24, 2015

1 Introduction & Work Completed

This progress report details the status of team Hydra's semester project, with frequent reference to the plans and schedules outlined in our initial proposal submitted on February 2. Since then, we have closely followed the timeline and accomplished the scheduled tasks proposed therein.

Specifically, we have completed a prototype of our ./ask program. The program takes a local .html file representing a Wikipedia article as input, parses the text of the article into sentences, parses the sentences into syntax trees, and transforms these trees to return a list of binary questions about the content of the article.

This prototype is limited in the following ways:

- Only two patterns are identified within the sentences for transformation into questions. These are "NP VP" (a simple predicate sentence) and "NP, NP," (apposition). We chose these two to start with since they can be reliably transformed into questions in predictable ways.
- The system only generates binary (yes/no) questions.
- The system directly transforms the aforementioned patterns into questions, so the answer to all of its questions is (i.e. should be) "yes."

In the next section we discuss how we will address these limitations.

2 Future Work

As noted in our initial proposal, we decided to spend roughly one third of our time since February 2 working first on the ./ask program, then one third on ./answer, then the last third on improvements and refinements of both. At this point, we have completed the prototype of the ./ask program. However, there is still some work to be done to flesh out ./ask to an acceptable level that we can move on to working on the ./answer program; namely, the three limitations cited in the previous section. We will address these issues through the following:

- As we run the prototype on several test articles, we will identify more syntax patterns which can be transformed into questions, and add these.
- We will implement transforming extracted patterns into wh-questions.
- We will implement various confounding techniques (NP/adjective substitution, negation, etc.) to make some of our questions have "no" as the correct answer.

Once this is completed, we will begin working on the initial implementation of the ./answer program. We will follow the steps outlined in our initial proposal to implement this.

After we have completed these initial implementations of the ./ask and ./answer programs, we will spend the last third of our time testing, evaluating, and improving them. We discuss our methodology for this in the Testing & Evaluation section.

3 Division of Labor

As noted in our initial proposal, we divided the implementation of the ./ask program into four discrete tasks, so that our four members could develop more-or-less in parallel. Specifically, the assignments were:

1. Ashley: parse the HTML, split the text of the article into sentences, and parse the sentences into trees using a PCFG parser.
2. Eileen: find instances of the two hardcoded syntax patterns in the sentence trees.
3. Jemmin: transform instances of the syntax patterns into well-formed, grammatical questions.
4. Rohan: evaluate and rank the questions according to various criteria

After each of us had completed our assigned task, we met to write a script which imports each of our modules and strings together the tasks to perform the ./ask functionality.

As noted in our initial proposal, we plan to divide the tasks for the ./answer program in a similar manner.

4 Tools & Changes

The tools we planned to use were listed in the Tools section of our initial proposal. So far we have used all of the tools listed there in some way. In addition, during the course of development we found and used a few new, useful tools:

- pattern.en: a Python NLP toolkit with several useful functions. We use it to lemmatize and conjugate verbs in the sentence transformation stage.
- The Stanford parser: a state-of-the-art parser from Stanford. We use it (via NLTK's interface) to parse sentences into syntax trees.

The approach outlined in our initial proposal for the ./ask program remained, at the high-level description that it was, unchanged. One major change in tool usage was our decision to use the Stanford parser rather than NLTK's built-in parsers. This came about as a result of reading various materials on NLTK usage which noted that NLTK's parser performed poorly compared to state-of-the-art tools like the Berkeley and Stanford parsers.

5 Testing & Evaluation

At this stage in our project development, we have just completed the prototype of ./ask and have not tested it on more than one article. From here on as we flesh out the initial implementation of ./ask, work on ./answer, and enter our refinement stage in the last third of our project timeline, our general testing and evaluation philosophy is this: start simple, test and manually evaluate on real data, and adjust/expand our approach accordingly.

We have started simple by implementing a simple version of ./ask, and we will do the same for ./answer. We believe that it's difficult (i.e. impossible) to anticipate every problem that we'll encounter in trying to implement our system, so instead our approach is to start with relatively simple, even naive, solutions and then test them on sample articles. We believe testing on just a few sample articles and manually inspecting the output will quickly reveal problems which we can then think about how to solve and incorporate into our system. This methodology also ensures that we always have a working (albeit perhaps not very well) product at every point after our initial prototype implementations, so we have little risk of struggling to get a large, complicated system to run for the first time right before the final deadline.

As for manually evaluating the output of our ./ask and ./answer programs, we will simply use real Wikipedia articles from the target domains to generate test cases. It will be an iterative process of noting the things that don't work (e.g. ungrammatical/illogical questions, answers that are ungrammatical or incorrect), identifying the underlying assumptions/design flaws that cause our program to produce the incorrect output, and changing the design and/or implementing new features to cover the cases which we missed.