11-411 NLP Semester Project

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

February 2, 2015

1 Introduction & Timeline

This proposal outlines team Hydra's plan for the 11-411 semester project: building a question asking and answering system for Wikipedia articles within specific domains. Team Hydra is composed of Jemmin Chang, Eileen Jiang, Ashley Lai, and Rohan Varma.

We decided to split the development of the ./ask and ./answer programs into separate stages. We plan to develop the ./ask program first, the techniques and challenges of which will inform our development of the ./answer program. We will divide the time remaining into three stages as follows:

Stage	**Task**	**Due Date**
./ask	Develop an initial implementation of the ./ask program	2/17/15
	Test, evaluate performance, and plan for improvements; Progress Report 1	2/24/15
./answer	Develop an initial implementation of the ./answer program	3/10/15
	Test, evaluate performance, and plan for improvements; Progress Report 2	3/19/15
Refine system	Rapidly test/evaluate and implement improvements on both programs	4/7/15
	Test/debug on GHC machine; tie up loose ends; prepare final system	4/14/15
	Final Report	4/21/15

Of course, we recognize the high likelihood of unforeseen challenges/bugs. This is why we've planned to complete development of each of the parts at least a week before the report or final system is due, so that we have ample time to address issues, adjust our schedule as necessary, and plan for improvements in our refinement stage. The next two sections detail the approach we will take to solve the asking and answering problems.

2 Asking

We have split the implementation of the ./ask program into four roughly independent tasks, so that each of us can work on one task in parallel, putting the parts together as we complete their development. These tasks are:

1. Parsing. Parse the HTML and extract the text of the article. Split this text into sentences using a sentence segmenter. Parse these sentences into syntax trees using a PCFG parser.

2. Pattern extraction. Using a list of hand-generated, hard-coded syntax patterns for phrases from which we can easily produce questions, find instances of these patterns in the syntax trees.

3. Question generation. Using the phrases extracted in 2, transform these phrases into questions according to hand-generated, hard-coded rules. These include rules for generating both binary and wh-questions. Make the questions harder by substituting hypernyms; adding adjectives, prepositions, or comparatives/superlatives; and changing numerical values.

4. Question ranking. Evaluate and rank questions according to a formula which includes factors such as:

   (a) Grammaticality (as evaluated by a trained question grammar or language model)

(b) Probability of the sentence parse from which the question was derived

(c) Ability of ./answer to answer the question (after ./answer is developed)

Return the n best questions according to this ranking.

This is the general approach we will follow to implement our initial ./ask program. When we've finished this initial implementation, we will test and evaluate it, noting challenges and possible solutions to implement in our refinement stage.

3 Answering

We have also split the implementation of the ./answer program into tasks for the same reason. Here we have five tasks, but the first is exactly the same as the first part of ./ask, so it will already be implemented. They are:

1. Parsing. See the same in Asking.
2. Question processing. Determine the expected answer type, either by hardcoded rules about wh-words and binary questions, headword identification, or supervised machine learning trained on questions with labeled answer types.
3. Sentence selection. POS tag and then lemmatize the question and sentences in the article. Using a vector model, weight matching lemmas using tf-idf and then compare each of the sentence vectors to the question vector for similarity, and select the highest ranked sentence.
4. For wh-questions, identify the smallest phrase within the selected sentence that is of the expected answer type and contains a lemma not in the question, and return this as the answer.
5. For binary questions, check that all lemmas in the question are matched in the selected sentence (without negation or other dangerous complications), considering hyponym/hypernym relations of lemmas, and return yes if everything looks good; otherwise, return no.

Again, this describes the approach for our initial implementation of ./answer. After implementing this, we will test and evaluate it, noting challenges and possible solutions for our refinement stage.

4 Refinement

The refinement stage will consist of rapid cycles of experimental changes and testing on the sample data to evaluate the performance of improvements to our initial implementations of ./ask and ./answer. Since it is impossible to anticipate now what this stage will entail, we have reserved almost three weeks for it to ensure ample time to address issues and make as many improvements to the system as we can.

5 Tools

We will use several external, freely-available tools for both solving the problems and for managing the project. These include NLP tools:

- Python - a great programming language for rapid development
- NLTK - a Python toolkit useful for many NLP tasks. We will use NLTK for sentence segmentation, sentence parsing, POS tagging, WordNet lemmatizing, and exploring WordNet relations (such as hyponymity, hypernymity, synsets, etc.).
- WordNet - a hand-generated database of English words encoding many useful semantic relations. We will use WordNet indirectly via NLTK.

As well as project management tools:

- Google Drive and Gmail: for communication and collaboration on reports
- Git/GitHub: for revision control

As we develop our system and (inevitably) make changes to our design, we may very well add other tools to this list.