Clarity: a Natural Language Question-Answering Engine

Mike Klein, Advay Mengle

May 17, 2009

Abstract

Despite its high price tag, Powerset doesn’t seem to use very advanced natural language technology. This project aimed to reproduce Powerset in a clean-room setting, then improve upon it. We determined that Powerset uses two distinct forms of search: it performs a gross, text-based snippet search making light use of synonyms, and in parallel matches queries against simple Subject-Verb-Object frames extracted from its corpus. We replicated both the text snippet search and the frame search, and enriched our frames with more grammatical information. In addition to reasoning about verbs, subjects, and direct objects, our enhanced frames allow our system to answer questions about indirect objects, predicate adjectives, cause, and the sources and destinations of motion. Our system, named Clarity, answers natural language questions about Wikipedia’s Vital 100 articles [4] through a web interface at http://clarity.xvm.mit.edu:8808/.

Introduction

The motivation for a project like ours can be found in the familiar maxim “Knowledge is power.” Knowledge of a topic gives us power over it, so programs that increase our knowledge are power-bearing tools.

One useful piece of insight is that our knowledge can be factored into, roughly, how many things we know, and how well we know them. This factoring leads to at least two obvious routes to increase the power of a system: cast a very wide, shallow coverage net, found in most text search engines; or understand a small area very well, as expert systems tried to do with specialized representations.

But ultimately the best thing is to do both: determine valuable representations that are both generally applicable and usefully meaningful, and extract these representations from the entire body of human work, as far as possible. Programs like Powerset (http://www.powerset.com/) make tenative steps in this direction; while still clinging to the current (extremely successful) text-based search model, Powerset can extract simple semantic—that is, meaningful—relations from natural language text. As a product, it uses these relations to answer natural language questions.

Our aim is to improve upon Powerset. To do this, we first attempted to determine what Powerset currently does from its external behavior. Then we reimplemented what we deemed its core facilities. Finally, and most importantly, we demonstrated that Powerset’s representations can
be enhanced in simple ways. Each enhancement expanded the types of questions that our system can answer.

Step 1: Understand Powerset

Our first step was an admittedly shallow inquiry into what Powerset does. We wanted to avoid patent, trademark, copyright, and other painful issues, so this section is entirely speculative. We want to note emphatically that our work, as far as it mimics Powerset, was developed in a clean-room setting. Though we made external observations, we know nothing definitive of Powerset’s internal details.

Snippet Search

By exploring Powerset’s world-visible web frontend, we were able to determine that Powerset presents results to a query in two primary forms. Most visibly, it returns snippets from Wikipedia, with words and phrases corresponding to the key search terms highlighted. (It appears that Powerset does drop stop words, like “the”, “in”, “is”, and question words.)

The highlighted words are not necessarily the exact search term; it is obvious that Powerset uses stemming (as you’d expect) to better match variants of a search term. But Powerset goes beyond just stemming: often the highlighted words are not the original search terms, but rather synonyms. A search for “bovine disease” shows matches for not only “bovine”, “disease”, and “diseases”, but also “cattle”, “cow”, “pleuropneumonia”, and “AIDS”. It appears that rather than matching against stems, Powerset matches against the synonym sets implied by those stems.

Powerset seems to employ a simple heuristic to choose which snippets to display: snippets are better when the key search terms are found closer together. Thus to find the best snippet, you look for the shortest snippet containing the search terms of the query. Of course, we cannot be certain that this is the approach Powerset takes, but it is reasonable: empirically, short snippets provide good results, and the shortest snippet is efficient to calculate; we show our polynomial implementation in Step 2.

Frame Search

Powerset augments its somewhat-standard snippet search with a database of dorkily-named “Factz”. (We reckon the word “Facts” must not be trademarkable when spelled correctly.) Factz are frames of a simple sort: a single Fact comprises a transitive verb, its subject, and its direct object. It seems the entire triple must be present; an intransitive query like “Who runs?” does not evoke any Factz, but a full, transitive query like “Who runs congress?” works fine.
As far as we can tell, Powerset has run Wikipedia fully through a natural language parser. From the parse results, it extracts what S-V-O triples it can find and stores them as its Factz. At query time, Powerset probably extracts a partial fact from the sentence and matches it against its database. Notably, it seems to only extract a partial fact if exactly one of the three slots is filled with a question word. Two-slot questions like “Who killed whom?” match nothing, while “Who killed Kennedy?” or “Who did Lee Harvey Oswald kill?” work fine.

Powerset’s natural language parser could use some work. While the questions “Who did Lee Harvey Oswald kill?” and “Who killed Kennedy?” match Factz, the seemingly equivalent questions “Lee Harvey Oswald killed whom?” and “Who was Kennedy killed by?” give no results. The code that extracts verb-subject and verb-object relationships from the parse must be rather brittle.

**Step 2: Reproduce Powerset’s Snippet Search**

We attempted to reproduce Powerset’s snippet search by imitating its two salient features: matching search terms with synonyms, and finding short snippets to return.

To match synonyms, we employ WordNet. In the preprocessing displayed in Figure 1, we step token by token through the corpus, looking up all possible Synsets in WordNet, and build an index mapping these Synsets to the articles in which they are found. This builds a large mapping that lets us put in an arbitrary Synset, like **dog.n.01**, and find all the locations that words falling into that synset were found—e.g. “dog” here, “canine” there, etc.

In total we index 100 Wikipedia pages, but given a query, we filter the possible relevant articles to just those containing at least one match for each search term. That is, we consider the intersection of the articles matching each search term. This guarantees the article has at least some snippet containing synonyms of our search terms. For each article, we find the shortest matching snippet,
then display several of these snippets in ascending order of length.

To find the shortest snippet in an article, we use a dynamic programming algorithm. Consider this brief, made-up example, where our query words are “bovine”, “mad”, and “cow”. The following index shows the locations these words appear in the article:

bovine 1 20 75 99
mad 6 34 39
cow 21 35 67 248

The algorithm proceeds row-by-row in arbitrary order (here first “bovine”, then “mad”, then “cow”). At each location on the table, we store the shortest interval containing the words up to the current row. So for the first row, we have 4 degenerate intervals: “bovine” is found ranging from 1 to 1, 20 to 20, 75 to 75, and 99 to 99. To extend the table to the next row, we consider each location and find the minimum way to add it to one of the intervals in the previous row.

Considering “mad” at 6, we’d look to the four “bovine” intervals and minimize the size of the new interval. In this case, the smallest “bovine-mad” interval for the “mad” at location 6 is 1-6, and the other two smallest intervals are 20-34 and 20-39. Note that we’ve already determined locations 75 and 99 are useless.

It gets interesting when you consider the next row; the locations now may fall either inside or outside the partial intervals. If they fall inside, they are effectively zero cost! When we move on to the “cow” row, we consider each of its four locations against the previous three partial intervals 1-6, 20-34, and 20-39. Location 21 fits for free into 20-34; locations 35, 67, and 248 all make larger intervals. As this is the final row, we can return that the shortest interval covering “bovine”, “mad” and “cow” is the range from 20-34.

This algorithm steps through each row once, and in each row compares every location in that row against the optimal intervals up to the previous row. So if we have $N$ rows with a maximum number of locations $k$, this algorithm takes $O(Nk^2)$ time. (More exactly, $\Theta(\sum_{i=2}^{N} k_i k_{i-1})$ where $k_i$ is the number of locations for word $i$.)

**Step 3: Reproduce Powerset’s Frame Search**

To reproduce Powerset’s Factz frame search, we also put natural language technology to use: we pre-parse the same 100 Wikipedia articles, extracting relations as frames as displayed in Figure 1. Initially, we mimicked Powerset’s simple S-V-O frames; in Step 4 we’ll talk specifically about how we enriched the frames with more information.

We chose to use RelEx [2] to extract our frames. RelEx builds on the Link Grammar Parser [3],
and uses heuristic, human-coded rules to give back even more information than the Link Grammar Parser. These two tools together serve as a robust natural language foundation for our system. Using these tools, we don’t even have to think about issues like passive- versus active-voicing. The sentences “Lee Harvey Oswald killed Kennedy.” and “Kennedy was killed by Lee Harvey Oswald” return the same representation! RelEx also handles questions elegantly. “Who was Kennedy killed by?” returns the same representation as “Lee Harvey Oswald killed Kennedy”, but with Oswald replaced with a standard question-word marker. RelEx and the Link Grammar Parser seem designed exactly for our purposes: they take a wide range of natural text and give back a standard representation of the relations between words. We highly recommend these tools.

Running RelEx over our 100 Wikipedia article corpus gives us back about 60,000 S-V-O frames. In our frames, all slots are optional except the verb, so we have many intransitive frames, like “She runs.” and sometimes even just verb frames, which may be bugs. This preprocessing is slow; on a dual-core 2 GHz machine, it took about 12 hours to process the 100 Wikipedia articles. It would takes several processor-years to trawl all of Wikipedia. Of course, this preprocessing is embarrassingly parallel, and with a cluster of 1000 or so servers, the task becomes quotidian. It usually takes very little time at all (< 1s) to parse the users’ queries.

We match frames by using another prebuilt index, which lets us filter the 60,000 frames down to only those that might be relevant, the union of frames containing one of the search terms. For each slot (subject, verb, object, etc.) in the query frame, we find any the frames containing that term. This filtering often gives WVVx to WVVVx speedups when matching.

Given a filtered list of possible matching frames, we then apply a custom distance metric between the query frame and each of those frames, displaying the best few to the user. We find this two-phase matching process reminiscent of the Forbus/Gentner MAC/FAC (“many are called but few are chosen” [1]) paradigm.

Our custom distance metric is not very good. We did not spend much time tuning this, or even really thinking about it too much; the current implementation serves more as a place-holder than even a proof-of-concept.

The current implementation works by combining several heuristics. For each slot in the question frame, if a question word is in the slot, penalize a potential matching frame if it does not have that slot. This penalizes matching “What do lobbyists run?” with “Cheetahs run.”; we decided that extra information is fine: we say “Who runs?” is adequately answered by “Lobbyists run Congress.”

When considering non-question words, we also penalize missing slots. But when both question and possible answer have words to compare in a slot, we assign finer-grained penalties by using a measure of similarity between words: words are more similar if their possible Synsets overlap. Dissimilar words in the same slot proportionally penalize the possibility of the match. We tried
using standard WordNet-based similarity measures, but found them a little too slow for our liking; our overlap measure approximates these more expensive algorithms.

Additionally, we allow modifiers to feature in the distance metric; “Hegel developed Transcendental Logic,” doesn’t match “Who developed Syllogistic logic?” quite as well as “Aristotle developed Syllogistic logic.” Both equally match “Who developed logic?”.

**Step 4: Enriching the Frame Representation**

We found Powerset’s S-V-O Factz to be fairly lacking in representational power. Given that RelEx makes it so easy to extract information about sentences, we thought it might be productive to see what other kinds of information we could represent, what other slots we might add to our frames. We settled on a hodge-podge of additional bits of information.

The following sections should serve as a taste of what is possible to extract from natural text today using off-the-shelf parsers. With each new slot, we gain the power to both ask and answer a new kind of question.
Trajectories

Inspired by Jackendoff, we added slots to hold the source and destination of a trajectory, and some hacky code to extract these slots from the output of RelEx.

We added a slot for indirect objects, in hope of extracting the beneficiary of an action. However, it seems that most of the time RelEx says it’s found an indirect object, it’s not quite what we intended. In the sentence “He called the dog Snoopy.”, “the dog” is apparently an indirect object. This seems grammatically suspicious, though admittedly it does have the same grammatical form as “He gave the dog a bone”, in which the dog is certainly the indirect object. Wikipedia is full of phrases of calling, believing, etc., which makes finding true beneficiaries of an action a difficult task.

Cause

RelEx is quite good at finding phrases and the relations between them, making it surprisingly easy to extract causal relationships from phrases linked by “because”, “since”, or the like. We extended our frame system to allow causal links between frames, which allows answering the all-important question “why?”.

Modifiers

We realized adjectives and adverbs typically play a minor but useful role in most sentences. Rather than adding new slots per se, we added the ability for modifiers to decorate existing slots; the subject, verb, object, etc. can all be modified by grammatically appropriate modifiers. The modifiers mostly serve to make slight distinctions in our distance metric.

Step 5: Putting it all Together

We combined these features into an end-user accessible system with a web front-end. Clarity’s runtime procedures are displayed in Figure 3. The web front-end sends plain-text questions to the assertion– and article/snippet–finder servers in parallel. The assertion-finder server returns natural language sentences reconstructed from the best frames found and the parts of those sentences that it believes answer the question asked (along with the original sentence and article URL from Wikipedia); the article/snippet-finder server returns the original sentences comprising the best snippets (along with the context the sentences appear in, to aid a human pronoun disambiguation). The two search types run completely independently.

The assertion-finder, article/snippet-finder, and parser proxy servers may all be duplicated by deploying multiple identical instances of each. The system provides simplistic load-balancing by
routing queries to a different random server of the appropriate type for each question asked. A more intelligent load-balancing algorithm would split the synset-indices amongst multiple servers, with all servers being utilized per question, but each server finishing faster since article/frame search is the most time-consuming part of generating answer.

Testing and Results

We admit that we have done no empirical testing of our system beyond playing with it while developing it. Given more time, we’d like to compare our frame coverage of Wikipedia with Powerset’s, but as a final project, we decided this was out of scope. Scientifically comparing the quality of our results with another search engine’s would most likely require recruiting unbiased subjects and following the standard COUHES–based study procedures. In addition, our ability to search only one hundred articles from Wikipedia (due to memory constraints) gives us a significant disadvantage in terms of the scope of our results, regardless of the quality of our search and answer algorithms. We do note, however, that while Clarity can answer questions regarding cause and trajectories, Powerset simply cannot do so.
Contributions

In this section we list the specific contributions of this project.

- Inferred specifications for Powerset’s main two forms of search: snippet- and frame-based.
- Re-implemented both forms of search in a system that answers queries about Wikipedia’s 100 Vital articles.
- Demonstrated that Powerset’s simple S-V-O frames can be improved by extracting more semantic relations.

A Using our system

Clarity was implemented in an odd combination of mostly Python for ease of coding and a little Java to interoperate with certain Java-only libraries like RelEx and to provide a web front-end. This bilingualism complicates running the code; ultimately the system comprises four servers interacting over HTTP as indicated in Figure 3. It’s a bit tricky to get running correctly, so we have deployed a demonstration at http://clarity.xvm.mit.edu:8808/. You can download an archived snapshot of our code, however we recommend using the demonstration due to the number of third-party dependencies and complexity of server deployment.

Loading the demo webpage should present you with a web-interface consisting of a text field to enter a question. Using proper capitalization, grammar, and punctuation helps RelEx generate a good parse for the question, and thus helps retrieve better answers. Please be patient, as queries can take anywhere from 3 to 30 seconds or more, depending on both the length of the question and the number of senses each word has.

Our system indexes the Wikipedia Vital 100 articles [4], so it may help to first browse those titles to see what sorts of things it might know about. But the courageous user simply types an English question into the text box, and presses Enter.

We list here some functioning, and perhaps impressive, questions that our system can answer well:

- “What do people eat?”
- “Why do trainees devote years to research?”
- “Who invented logic?”
- “What did Hegel do?”
- “What did Hegel invent?”
Figure 4: Excerpt of a web interface depicting Clarity’s answers to the question “What leads from a premise to a true conclusion?”. Assertions are displayed on the left, and article snippets are displayed on the right.

- “What leads from a premise to a true conclusion?” (depicted in Figure 4)
- “Patterns lead from a premise to what?”
- “Where did homo sapiens move from?”
- “Where was the capital moved from Nanjing to?”
- “Who does parliament give opinions to?”

Many other questions will give embarrassingly poor results. We believe this is not so much an indication of a bad approach, but our relatively little time spent tuning the metrics judging the quality of a match, and perhaps some misunderstanding of the relationships given to us by our parser.

References


