NAPALM SQUAAWKS: A Naïve APproach towards A Linguistic Model for Simplistic QUestion Asking & Answering on WiKipedia Segments

Joseph Petrich (petrich@cmu.edu)
Joseph Zischkau (jmzisckau@cmu.edu)

Introduction

For Natural Language Processing (11-411 at Carnegie Mellon University), we implemented a question asking and answering program trained and tested on Wikipedia articles. These articles came from four categories: People, Cities, Languages, and Instruments. A fifth "mystery" category, Airports, was used to test our system. This category was not disclosed to us until development of our system was completed. Our program was supposed to ask only Easy, Medium, and Hard questions, as defined by the instructors. An easy question was defined as "A yes-or-no question that could be answered fairly easily using string matching." Medium questions "could be answered (if a bit tersely) by exact string matching. The answer is likely a single noun phrase, and certainly no longer than ten words." A hard question "requires some synthesis of information across slightly larger portions of the article; string matching is probably not enough. However, the answer is clearly within a small segment of the text for a human reader who can recognize synonyms and basic propositions."

Throughout development we strove to implement simple solutions through an intelligent and thorough consideration of the problem at hand. We did not attempt to solve the problems of computational linguistics, nor did we design a system to utilize all the state of the art computational linguistic tools on a supercomputer. Instead, we put together a system designed to run on a computer with a quad-core processor and 4GB RAM, using the Natural Language Toolkit (NLTK) within Python and run through a shell script. This system outputs intelligent, natural language questions given any of the four considered types of articles, and outputs the correct answer in a human readable format most of the time.

Methodology

Programmatic Architecture

Our system was implemented using a core of python scripts, which handled the main functionality. We made use of the Natural Language ToolKit (nltk) for sentence and word tokenizers. We used NodeBox for POS-tagging, and we used shell scripts as wrappers for the core python scripts, and also to handle various non-core processing, computation, and analysis. The basic layout of our ask program is as follows: the user runs the command *./ask article.txt nQuestions*. The *ask* shell script first executes a separate python script to convert the article into sentences. Then, it invokes the python *ask.py* program, and outputs the results obtained, with one question per line. The *answer* shell script was invoked with the following command: *./ answer article.txt questions.txt*. Again, the shell script passes the article to the python script for sentence separation, and then passes the sentences and questions to the python answer script *answer.py* which returns answers. These answers are then output as one answer per line. The core python scripts invoked other python helper scripts for certain tasks. These helper scripts included *classify.py* which classified the article into one of the four categories, *evaluate.py* which graded questions for fluency and *pronounce.py* which was used for date manipulation.

Preprocessing

We did a small amount of preprocessing on the article files given to our programs. For a given article text, we would tokenize it into sentences, and output it to a file with one sentence per line, with “empty” sentences being removed. This allowed us to more easily look at each individual sentence for both question asking and question answering.

Classification

In both the question answering and asking portions of this project, we wrote a script for classifying the subject of the given text. Initially, we implemented a Naive Bayes classifier working on unigrams, bigrams, and trigrams trained on articles from the four given subjects (People, Cities, Languages and Instruments). This classifier, however, used valuable processing power and time, and failed to identify the four categories with 100% accuracy. Discarding this method, we wrote a simple decision tree, using hand selected features. Instead of scoring the text using many features, and then picking the category with the highest score, we checked each category, only proceeding if the qualification for that category was not yet met.

The first classification we checked was the language category. All Wikipedia articles concerning languages have the word “language” in the title to avoid ambiguity. Therefore, checking the title to see if "language" was contained in it was a one hundred percent accurate method for classifying the article as a language article. If the word "language" was not present, we proceeded to check if the article was in the class of instrument articles.

We obtained from Wikipedia a list of all instruments with Wikipedia articles. Our test for whether a text concerned an instrument consisted of checking the title of the text against that list of articles. Failing that test, we moved on to the city category.

People and cities are harder to distinguish. We considered looking for a birth date, family words, or even using a list of names. With some experimentation, however, we found that a count of the word "city" within a text worked to distinguish all city articles from all person articles in our training data, and this is what we implemented. Having made it through all these tests, the script would assign the category "person" to a text failing all previous tests.

Clearly this approach only works on the specific domain given: Wikipedia articles on one of four subjects. However, to achieve a better classification system for the specific evaluations performed in the class, we ought to have implemented a specific "person" classifier, and then assigned anything which did not fit into any of the four categories to an "unknown" category. To expand our approach to all Wikipedia articles, a larger decision tree would have to be written, and more advanced techniques (like Naive Bayes classifiers or clustering algorithms) could be used to distinguish less distinct categories like cities and countries.

Question Asking

Predetermined Questions

Knowing what general information is in every Wikipedia article of a certain class, we wrote a selection of questions corresponding to each class. The predetermined questions are enumerated in Table 1.

PERSON	
On what date was <PERSON> born?	On what date did <PERSON> die?
How old was <PERSON> when he died?	Was <PERSON> ever married?
Who was <PERSON>?	Did <PERSON> attend college?
When did <PERSON> come into this world?	Does the article mention that <PERSON> published anything?
LANGUAGE	
Where is the <LANGUAGE> spoken?	What language family is the <LANGUAGE> a part of?
Approximately how many people speak the <LANGUAGE>?	Who speaks the <LANGUAGE>?
What is the word order in <LANGUAGE>?	Does the <LANGUAGE have vowels>?
How many vowels does <LANGUAGE>	

have?	
CITY	
What is the population of <CITY>?	In what country is <CITY>?
What is the population density of <CITY>?	What kind of transportation exists in <CITY>?
Where is <CITY>?	How old is <CITY>?
What kind of climate does <CITY> have?	
INSTRUMENT	
What is the <INSTRUMENT>?	How is the <INSTRUMENT> played?
How does one play the <INSTRUMENT>?	Does the <INSTRUMENT> have strings?
Is the <INSTRUMENT> a wind instrument?	Is the <INSTRUMENT> a percussion instrument?
Where does the <INSTRUMENT> originate?	What kind of music is played on the <INSTRUMENT>?

Table 1

If there are more predetermined questions than the number of questions requested, the script randomly selects from these predetermined questions the number of questions requested, and the task is complete. In case more questions are required, the program proceeds to methods of asking questions based on the specific text.

Copula Movement in Past Tense +Q Sentences

Most Wikipedia articles have a large number of past tense sentences using copula verbs. Sentences are most often past tense for two main reasons. First, articles about people are usually (and for this project, always) about deceased people, and their activities occurred in the past. Second, all Wikipedia articles rely on information to cite; for this reason, sentences are not usually speculative (future tense) and rarely describe the current state of events (present). Wikipedia is also designed to be accessible to a wide audience, so its language is often simple. The most simple sentences in link two nouns or noun phrases in a sort of equality, using a copula verb. In English, this copula is “to be,” which is pronounced “was” in the past tense, “is” in the present tense and “will be” in the future tense.

We utilized this information in order to transform statements ([-Q] sentences) into questions ([+Q] sentences). According to current syntactic theories (X’ Theory and its derivatives) the underlying representation of a sentence does not change in the formation of a question sentence from a declarative sentence. Instead, these theories postulate movement from the place a constituent holds in the declarative sentence to a place that denotes questioning. Sometimes other words have to be inserted to fulfills semantic constraints, to clarify tense, or provide case, but the important thing is that cases do exist in which there is simple movement of one word in order to form a question. One of these simple cases is that of copula movement, in which a copula moves from the tense position to complementizer position, in order to satisfy the [+Q] feature, which can only appear in this position.

Our program utilizes copula movement in past-tense sentences to form questions beginning with "was." The sentence "Jelinek was a recipient of a IEEE Third Millenium Medal in 2000" becomes "Was Jelinek a recipient of a IEEE Third Millenium Medal in 2000?" simply by moving "was" to sentence-initial position. Interestingly, this same method works for past-tense passive-voice sentences as well. "Jelinek was survived by his wife?" becomes "Was Jelinek survived by his wife?" with the same movement. This works because "was" in passive sentences originates in the verb and moves to tense because the verb already has tense (-ed [+past] on survive). We then move it from that tense position to complementizer position as before. Our program does the same kind of movement with the present tense "is."

Adjunct Extraction and Manipulation

In semantic role labeling, adjuncts are often overlooked, as they are not necessary to the understanding of a sentence, and they can move around, making them hard to identify. However, we eschewed the idea of semantic role labeling because current implementations are imprecise and error-prime. We attacked a specific adjunct identification problem directly, using regular expressions to extract adjuncts of the form "is the NP_1 of NP_2." We then transformed this into the question "What is the NP_1 of NP_2?" This approach worked well for City and Language articles, as these types of sentences were relatively common.

Lexical Substitution

Our most commonly used lexical substitution involved forming questions by changing the third person nominative pronoun "he" to "who." The questions formed using this substitution are more difficult to answer than syntactic movement questions, because the answer cannot be solved simply by string matching. As an example, the question "Who was a recipient of a IEEE Third Millenium Medal in 2000" generated by our program matches the sentence "He was a recipient of a IEEE Third Millenium Medal in 2000." Returning this sentence, or the word "he" would not adequately answer the question because "he" is ambiguous.

Many other lexical substitutions of this sort could be performed. The feminine nominative singular pronoun "she" could be substituted, but it occurred substantially less than "he" in our training data. The accusative forms of both these pronouns ("him" and "her") could also be substituted with whom. These were not implemented in the final form of our program to streamline the code and maximize the relevance of our techniques.

Another method of lexical substitution we tested was substituting adjectives. For this, we tagged the text with part of speech tags (POS-tags) and then switched adjectives around in the article. As an example, given the sentences "Pascal was a talented mathematician. He worked on an unwieldy project" this process would output "Pascal was a unwieldy mathematician. He worked on an talented project." These sentences could then be turned into questions through syntactic methods. This example outlines two major problems with this strategy: a/an agreement and semantics. A project cannot be talented, because, semantically, talented can only apply to animate nouns. An alternative approach we tried was to utilize WordNet to substitute synonyms or antonyms for adjectives. However, WordNet is computationally intensive. For these reasons we discarded this portion of the program.

Date Extraction and Manipulation

Dates are abundant in Wikipedia articles, as Wikipedia deals primarily with past events. Random dates are not included in Wikipedia articles. Following this logic, we decided to ask questions about dates, as they relate to significant information within the article. Using regular expressions, we identified dates of the form Day Month Year and Month Day, Year. These are the two forms commonly used in Wikipedia articles. Then, we asked "What is the significance of <DATE>?" We also found years by looking for the word "in" followed by a four digit number. Using this year, we asked "What happened in <YEAR>?"

We knew that our system would be tested against other systems. Having already identified dates, we decided to manipulate them to make our questions harder (if not impossible) to answer using a computational system. We wrote a separate script which transformed dates into their pronounced forms. Day Month Year and Month Day, Year became "the <N>-st/nd/rd/th of <Month> <YEAR>" where <N> was replaced with its pronunciation (24 becomes twenty-fourth" and <YEAR> was replaced by its pronunciation (1978 became Nineteen Seventy-eight). We performed the same year transformations on our "What happened in <YEAR>" questions.

Filtering For Fluency

Part of the training data available for development was a database of all fluent questions asked by systems designed for the same project over the past few years. While these questions were helpful for evaluating the answering portion of the system, we also used them for grading fluency of the questions we asked. First, we separated all the past questions into individual lines and stored them to a dictionary. Then, we ran an implementation of the Brill Tagger from the NodeBox English Linguistics Library on this dictionary of questions. We kept the first two letters of each tag (the general part-of-speech) and concatenated the tags for each question into a string. We wrote these strings, one per line, to a new file, which then contained all of the simple POS-tags for all known fluent questions. This was all done in the preprocessing stage, and the POS-tags file is a dependency of the question asking system.

Excluding the predetermined questions and questions derived from dates, we ran our list of potential questions through a fluency grading algorithm based on the preprocessed file of POS-tags from known fluent questions. This algorithm performed the same POS-tagging procedure on the potential question as on the known fluent questions, concatenating the simple POS-tags into a string. Then, it compared the string of POS-tags to the list of strings of known fluent POS-tags by calculating the minimum edit distance. The algorithm returned the minimum edit distance from the question to the answer. If this minimum edit distance was beyond a threshold set through trial and error, the question was discarded as disfluent.

Verbose and Negative Questions

In order to add to the number of candidate questions, ensure that all easy questions could not be answered "yes," and foil string-matching algorithms for answering, we implemented two algorithms. One appended “According to the information in the article,” to the beginning of every question. The other negated questions generated by copula movement. This algorithm POS-tagged the question, and then inserted “not” after “was” unless a proper noun existed in the sentence before a verb, in which case the “not” was inserted after the proper noun. So, “Was Jelinek a recipient of a IEEE Third Millenium Medal in 2000?” became “Was Jelinek not a recipient of a IEEE Third Millenium Medal in 2000?”

Question Answering

Basic Approach

For question answering, we began by implementing a naive approach to the problem. We tokenized each question into words. Then, for every tokenized sentence in the article, we created an empty counter, and then looped over the question word tokens. If a particular question word token appeared in the sentence, we incremented the counter. After we had checked all the questions words, we stored the counter in a dictionary with the sentence as the key and the final value of the counter as the value. After performing this operation on every sentence, we had obtained a score for each question based on the word overlap that sentence had with the question. The sentence with the maximum score out of all the sentences was chosen as our candidate for the answer.

We did not attempt to convert the sentence into a more concise answer. For example, when given a yes/no question, and a candidate sentence, deciding whether the answer was “no” or “yes” is actually a quite difficult problem. Factors that make this problem difficult include multiple uses of negative words in different forms, including contractions, and also the use of synonyms or antonyms in the question. We decided that the extra computation and processing time needed to turn the sentence into a concise representation of the answer was unwarranted, especially given that our system was able to return a sentence with the information needed to answer the question.

We attempted several improvements on the naive approach. The methods we attempted were ignoring insignificant words, stemming all words, and employing parsing as a basis for semantic relation determination. In ignoring insignificant words, we omitted words such as “the”, “and”, “or”, “who”, and several other similar words. However, we noticed no significant improvement from this method, and therefore left it out of the final implementation. Another method we used was reducing all words into their lemmas to account for tense swapping, plurality, or other differences that could cause exact string matching to fail. However, like the insignificant word ignoring method, we did not see any significant improvement with this functionality, and left it out of the final system. We did some initial testing with the Stanford Parser, and found that it was not suited for parsing Wikipedia articles because of the noisy data they contain from sources such as image captions, or the alt-text of inline images, and was not accurate enough to make it useful for sensible semantic relation resolution.

Common Questions

Since we had decided to generate class-specific questions about information guaranteed to be contained in the articles but not necessarily contained within a single sentence or even explicitly stated, we implemented a system to recognize these “common questions” and answer them based on the class of the article. For classification, we used the system described above. We used our own class-based questions as the starting ground for answering common questions. The method we employed for this purpose used custom functions for recognizing a specific type of question given the class. Generally, the question identification involved using a word overlap similar to the way we implemented our naive method, except we would compare the words of the question to a list of predefined words that we chose to indicate a certain type.

The first type of question we looked to identify was the type of question involving the birth or death of a person. We knew that all Wikipedia articles about people that we could be given would be about a person who is dead, and the article would contain their birth and death dates in a parenthetical in the first sentence of the article following the title. Therefore, upon identifying the birth/death question type, we could extract the dates and then answer any of the following three questions (or slight variants thereof) appropriately: “On what date did <PERSON> die?”, “On what date was <PERSON> born?”, and “How old was <PERSON> when they died?” Simple date subtraction served to answer the “how old” question, and the appropriate date was returned for the other two.

The second type of question we attempted to identify was “population” questions for the “city” class of article. We identified the question in a manner similar to the one stated above. There were two sub-types of this question, the “population density” questions, and the “population” questions. In answering these, we searched through the sentences for ones that mentioned various keywords such as “population” and “density.” Solving the population density questions was easy in most cases, as we were able to extract a sentence, but in some cases there was just a given population and surface area, which would involve performing a calculation. We decided not to do this calculation because it would only be necessary in rare cases, and we were more focused on solving the “population” question subtype. Unfortunately, this proved to be a harder task than anticipated, because many articles listed many populations at different times in their history, and it was often not clear from the immediate sentence what time period that specific population count was from. As such we were unable to determinately yield the correct answer for this question subtype.

Results

Our question asking program was able to generate fluent questions that were reasonable if the article on which the questions were based fell into one of the four specified classifications. A majority of these questions were deemed "hard" as per the evaluation scheme given in the class. While our program did not score among the top five in the class, we believe this is because of the questions generated for the mystery classification (Airport), which incorrectly assumed those articles (in most cases) to be of the Person class. It is interesting to note that in the pool of example questions displayed during the final evaluation and disclosure of results, a large majority of the questions were easily recognizable as output of our system. Furthermore, it is worth noting that our testing process involved generating a larger pool of questions from each article, and therefore the three-question limit on each article used in the evaluation did not allow for an appropriate representation of the breadth and depth of our system.

Our question answering program was able to find the sentence containing the relevant answer a majority of the time. The specifications for the project deemed the return of such a sentence both fluent and correct. We have reason to believe that, given the evaluation specifications, the entire evaluation process was prone to subjective variations. We believe that our accuracy scores would have been higher had the evaluations of fluency and correctness been standardized and carried out by an unbiased third party.

Our system won the award for "most creative" system.

Conclusions

Given a narrow scope and with various computational linguistic tools at hand, a robust question asking and answering system can be developed, as our work has shown. Our system does not handle the English language with the alacrity of Watson or ask the kinds of difficult questions an anthropology professor might ask about motives. Our system was also not designed to do so. With more development, greater computational resources, and an infinite (or at least much larger) training set, it would be possible to integrate, into our simple architecture, Named Entity Recognition, Semantic Relation information, Syntactic Tree-Parsing, and other state of the art NLP and machine learning tools. Using these tools, articles could be parsed into semantic logic statements, and questions could be evaluated by querying this semantic database. However, the creative methods we employ in our system could still be used to grade fluency and increase efficiency.

We discovered that naïve approaches can produce a very good baseline for a system that attempts to solve or employ NLP methodology for a specific task. Using very simple methods, we were able to perform the given tasks well enough, and further attempted small improvements produced no or insignificant increases in the performance of the system. To achieve a significant improvement, methods mentioned in the preceding paragraph would need to be applied to the system.

Acknowledgments

We would like to thank Dr. Alan Black and Dr. Chris Dyer, our professors for Natural Language Processing at Carnegie Mellon University, for their help throughout the semester introducing us to the big ideas of computational linguistics. We would also like to thank Google for hosting the demonstration of our system on May 1, 2012 at their Bakery Square offices in Pittsburgh.

References

Bird, Steven, Edward Loper and Ewan Klein (2009). Natural Language Processing with Python. O'Reilly Media Inc.

Carnie, Andrew (2010). *Syntax: A Generative Introduction*. Blackwell Publishing.

De Bleser, Frederik, Tom De Smedt, Lucas Nijs (2002). NodeBox version 1.9.5 for Mac OS X. Retrieved March 2012, from: http://nodebox.net/code/index.php/Linguistics

Jurafsky, Daniel, and James H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition - 2nd ed*. Pearson Education, Inc.

Smith, Noah A, Michael Heilman, and Rebecca Hwa (September 2008). Question Generation as a Competitive Undergraduate Course Project In Proceedings of the NSF Workshop on the Question Generation Shared Task and Evaluation Challenge, Arlington, VA. http://www.cs.cmu.edu/~nasmith/papers/smith+heilman+hwa.nsf08.pdf