Question Generation: Using NLP to Solve “Inverse” Task

Alec Kretch
Jan 8, 2018 · 4 min read

The future of customer service may be solved through the finessing of well-trained artificially intelligent (AI) bots. Facebook has opened up their Messenger bot platform for developers, and question answering has been a major area of research in Natural Language Processing (NLP) for years.

However, the opposite task — and depending on your perspective, more important task — has received far less widespread attention. While question answering can provide powerful insights and instant, humanless responses, question generation can *teach*.

That is where my system *Text2Questions* (T2Q) comes into play. Tasked with generating a set of questions based on some given input text, the target demographic of T2Q is

academic institutions — high schools and colleges — and, more specifically, educators at these institutions.

College professors, for instance, could save hours of time by automating the process of test and assignment creation. These hours could instead be spent on preparing content for upcoming lectures or labs. The benefit then falls directly upon the students, who would have better learning materials *and* a more-focused teacher as a result of T2Q.

Of course, NLP systems such as T2Q are only powerful resources if they work as advertised. What use is a system that generates irrelevant — or worse, incoherent — questions?

In prioritizing a great system for a subset of the problem over a generalized solution for the grand problem, I focused my efforts with T2Q on US history. That is, the training data sets and generated question types — in which year, on which date, fill-in-the-blank, true/false, and how many — were developed with only US history in mind.

Let's jump right in!

Approach

The goal of T2Q is to take input paragraphs of text from a textbook, perform some magic, and output a set of relevant questions based on the information. Said magic is not actually magic at all, but rather a pragmatic, step-by-step NLP process.

My approach is as follows:

1. **Tokenization:** Tokenize the input text into sentences then words using a tokenizer such as *NLTK*.
2. **POS-tagging:** Assign POS tags to each individual token using a POS-tagger such as *NLTK* or my own lightweight Markov Model.
3. **True-casing determination:** Assign tokens their true capitalization casing style using a true-caser such as Stanford CoreNLP's TrueCaseAnnotator. This is especially useful for textbooks that include important vocabulary terms in a caps-lock style.
4. **Subject/phrase determination:** Build a list of subjects and corresponding phrases from the tokens and POS tags using text chunking, Stanford Dependencies, or a multiclass classifer. Universal Dependencies can also be extracted such as from using Stanford CoreNLP's UDFeatureAnnotator. A multiclass classifier or trivial

determiner can then be used to determine the best subject/phrase matches to be used for question generation.

5. **Synonym replacement:** Convert some individual words to synonymous words using *Princeton WordNet* to find words with synonymous lemmas or *Stanford GloVe* to find words with similar word embeddings. This step aims to prevent students from matching words in order to answer questions.

6. **Additional information extraction:** Associate additional information with each subject/phrase match that could be useful for question generation using regular expressions. T2Q, for instance, matches common patterns that identify the date or year of a given event.

7. **Question formation:** Attempt to form a question of each question type for each subject/phrase match using basic patterns of each question type as a guide.

8. **False answer formation:** Form similar, but false, answers for each question based on the question type using patterns of good false answers for each question as a guide. For instance, true/false questions can be negated to determine the correct and incorrect answers, while word embeddings or a system scoring similarity between words and POS tags of other subjects can be applied for fill-in-the-blank questions.

9. **Shuffle and return questions.**

Further Research

Question generation is an under-researched topic in NLP, and one with the potential to make a powerful impact on the future of education. While I am excited about the current stage of T2Q, it is still only a work-in-progress and comes with its own shortcomings.

Ideas that warrant consideration for future implementations include:

- Using information from previous sentences or previously inputed texts to form questions and false answers.

- Expanding question types, including types for more abstract subjects such as calculus, which may require processing similar to *Wolfram Alpha*.

- Building a stable platform around the system such as one that allows the scanning of textbooks to trigger question generation, has features to allow a professor to dismiss, edit, or add questions, and learns of which questions cause struggles for different students.

Sample Command Line Output

The following is an actual output from testing an input chapter of a US history textbook with T2Q:

> ***$ python3 text2questions.py -tf us-history-chapter-sample.txt -max 3***
> *[{'type': 'TF', 'question': 'True or false: 750,000 family farms disappeared through foreclosure or bankruptcy.', 'answers': ['True', 'False'], 'correct': 0}, {'type': 'HMA', 'question': 'The Dow Jones Industrial Average peaked at a value of how many points?', 'answers': ['978', '319', '711', '381'], 'correct': 3}, {'type': 'IWY', 'question': 'In which year did a major strike at the Ford Motor Company factory near Detroit result in over sixty injuries and four deaths?', 'answers': ['1931', '1928', '1933', '1932'], 'correct': 3}]*