Nik Kirstein

ISTA 355

5/2/19

**Pedal Writeup**

Note this requires spacy to work, is also requires both the en\_core\_web\_sm and en\_core\_web\_lg models from spacy to work. Vectors only come with lg, but I used sm elsewhere to reduce space and runtime. Also requires a library called jsonlines to parse jsonl files. Other libraries should be ok.

Task

The task of this project was to use grammar rules to parse questions into forms that were then able to be evaluated against existing triples using cosine similarity for the most likely right answer. I believe my program does that to a simple level. There are three sets of questions and you can pick a question to attempt to answer from each set. It does not automatically evaluate at this time but is functional and you can insert in triples and see if it outputs the right answer manually.

Data

The data used is a few development sets from googles natural questions data set. This data set is comprised of “natural” questions and answers from users; manually annotated by hand from google. It was heavily parsed in a few ways. First, not all questions were annotated, so questions that were not annotated were tossed. Secondly, using some negligible grammar rules and one string keyword search, I further narrowed down the questions to a list of 126 which I then split into 50, 50, 26 for my dev, training, and test sets. I also further annotated the list of 126 questions by hand into hand made triples for checking. That took quite a bit longer than expected due to answers being very ambiguous at times. It was not an easy annotation task.

Process

This program required a large amount of data processing in many ways. I had to first download the data from google and parse it using jsonlines. Next, I had to write grammar rules using regex and extract the noun phrase/subject of the sentence and extract the verb phrase of the sentence and combine them with the question word to create a triple. I did this with straight regex on tagged sentences which then required me to take out the tags. There might have been an easier way to do this, but I found spacy’s matcher to be lacking and difficult to use. I did attempt a parser using it but it’s just legacy unused broken code. After getting the triple, we run spacy’s vector builder on each of the two meaningful parts of the triple and generate a score for the triple. Then we go through a list of other triples and combine cosine similarities (spacy’s built-in similarity method) of the parts of the triple and create a list of scores. We find the max score from that list and its index should be the index of our answer. Most of the difficulties in the project and overall were dealing with the various data formats and the large amount of data. I almost always had to save files as I went in the forms I wanted and read them in again because doing the data parsing each time would take too long. (Hence why we check for the existence of the annotated TSV files rather than parse the jsonlines files every time). It also doesn’t make sense to turn in massive data files to D2L so I had to be aware of how I could format the script to be usable and quick.

Evaluation

I’ll be honest here, this is my project’s current lacking component. I did not code an automatic evaluator, but it does allow you to ask questions from the list and it will tell you if the system it uses to determine an answer is right or wrong. It however, pulls from the gold triple annotations I created because pulling from the overall list of triples from the triple database of 15 million (but parsed down to 28000) triples still takes too long to run (15 minutes to check against 28000 triples). It does seem to usually get the correct answer by taking vectors of both triple lists. Do note that the triples my grammar parser creates aren’t an exact match to the triples I annotated by hand because I’m really bad at regex so it matching is a good sign for future development on a large free list of triples.

Problems

There were so many problems in various ways, I’ve listed some of the notable ones in bullet points.

* Parsing files is slow and doing it every time caused multiple problems throughout the project, some fixed, some not. Saving files every time we do large computation on them or massively go through their formats is the only way seemingly to get around this and even then, I ran into problems with it. Solved the jsonfiles version of this problem but didn’t solve the problem where looking through the 15 million triples parsed down to 28000, doing vector similarity for each and every single triple in the 28000 list taking 15 minutes.
* Data formatting nightmares: bytes that had to be read in as utf-8. Annotations from google that didn’t really answer their questions making my own annotations difficult. Spacy matcher not allowing regex on their tags, only on text so either creating tagged strings or doing my own regex without spacy (what I ended up doing).
* Doing vectors of entire sentences potentially loses meaning and causes problems compared to doing vectors of just words. There’s an entire field of research on sentence vectors and it’s still not clear what the best method is. I mostly ignored this problem as spacy allowed me to use sentence vectors, but it potentially is a bad system.

Next Steps

I will also do this in bullet point form since it will be clearer some of the list items to do with this project.

* Better regex grammar parsing. I am familiar with regex but writing it is still hard and an art to me. I’m sure perfect regex for noun phrase and verb phrase parsing in these ways can done and my method of making tagged strings and then stripping out the tags is ugly and expensive code wise.
* More questions. I was supposed to tackle more questions, but it was difficult to get my project working end to end, I didn’t end up even doing an automatic evaluator and different questions would require a whole different question parser and potentially a different grammar parser too if my regex was badly written.
* Code in an automatic evaluator: one of the reasons I didn’t manage to get around to this is because it took so long to calculate vectors for the list of reverb triples off the web. Doing it on the gold standard annotations I have isn’t the same since it’s just checking against a small number of other triples instead of thousands and is obviously likely to match the correct triple based off cosine similarity. I need to fix the problem of my program taking too long to evaluate vectors against the large list of triples in order to do proper evaluation anyway.
* Sometimes my regex produces an empty triple which in turn produces an empty vector. Spacy and python will complain about this, but it won’t crash the program. This could be fixed in a variety of ways, either with better regex or have error handling to handle empty vectors in the future.
* Implement allowing the user to ask any question one day. This was the end goal, but it allows so much more ambiguity and would benefit from more questions being handled, possible question classification.

Closing thoughts

While my project doesn’t evaluate automatically, it can still be verified that it works, and the pipeline is basically all there. You can see in the function run\_dev\_set() the process to run it against the main triple list from reverb instead of the gold standards. The fact that it usually returns the correct answer by taking cosine similarities of triples it generates from grammar and triples from the gold standard is a good sign. The triples generated from my grammar don’t always match the triples in the gold standard and I’m still doing similarities from vectors within spacy to find this answer and it’s returning the correct one most of the time meaning my code should be sound. All in all, this project was interesting but also the data formats and dealing with computation time due to data size and dealing with large files (especially since I have almost no space on my hard drives and had to actually clear up space multiple times) were new challenges I didn’t have to deal with much before natural language processing tasks.