QA System

Executive Summary:

Business Insider is a business and financial news website. The goal is to build a question and answer system given Business Insider articles from 2013 and 2014. The requirements for the system is that it should be able to answer the following types of questions:

- Which companies went bankrupt in month X of year Y?
- What affects GDP? What percentage of drop or increase is associated with this property?
- Who is the CEO of company X?

Due mostly to limitations in computer memory or RAM, only the 2013 corpus is used. Regardless, the QA system is able to answer CEO type questions with ease, company type questions with some difficulty, and percentage type questions with more difficulty. In short, the QA system queries the user, processes the question, retrieves relevant documents to the question, extracts relevant sentences from those documents, and finally extracts the answer from the highest ranked sentence. At the core, term frequency–inverse document frequency (tf-idf) is the information retrieval method used for both documents and sentences from those documents. The sample outputs can be found in 'sampleout.txt'.

Methodology:

In order to optimize the speed of the QA system, I thought it would be good idea to pre-process a database of documents. Instead of having to parse documents from the corpus each time a question is processed, I sentence tokenize, word tokenize (removing stop words and utilizing stemming), part of speech (pos) tag, and named-entity recognize (NER) every sentence in each document beforehand. The NER classifier is used from homework 3 which classifies if something is a 'CEO', 'COMP', or 'PCT'. Then, I build out the tf-idf matrix for the entire corpus. This initial step is time consuming, but significantly increases the processing time of the QA system later on.

Transforming the corpus into a tf-idf is essential for retrieving relevant documents as it provides scores for each word in the corpus for each document. However, the tf-idf used isn't the same standard textbook notation:

$$\mathrm{idf}(t) = \log \frac{n}{1+\mathrm{df}(t)}$$

Instead, a slight modification or smoothing term is added to regularize the idf value. A 1 is added to both the numerator and the denominator:

$$\mathrm{idf}(t) = \log \frac{1+n}{1+\mathrm{df}(t)} + 1$$

The tf-idf scores of the words are then L2-normalized document-wise so that the score are scaled appropriately between documents (no one document should have disproportionately higher scores than another as it would hurt the document retrieval process).

These are the default parameters used by the sklearn package. There is no universally agreed single formula for computing tf-idf. Other packages for other methods are also available such as the okapi formula.

Next, I build the QA system as a function. When the QA function is called, it prompts the user for a question, processes the question, retrieves the most relevant documents, retrieves the most relevant sentences, and extracts the answer from the most relevant sentences.

Step 0) Prompt user for a question.

Step 1) Question processing

I classify our question so that our system knows what type of answers to look for. I use a rule based approach because the questions that we intend to answer are more specific\localized (i.e. we know what kinds of questions to expect). A classification based approach is of course another possible approach but would require a dataset of questions and their labels. In this case, we do not have that unless we made our own which is possible, but in my opinion unnecessary for the problem we are solving. For my rule based approach, I try to extend it to more general cases by adding a miscellaneous class. Thus, there are 4 different question types for the system: CEO, company, percentage, and miscellaneous for anything else. Below are the main rules/ideas used to classify a question.

If a question follows these rules, classify as type 'CEO':

- Question word is 'who'
- Some synonym of 'CEO' like 'corporate executive' is in the question

If a question follows these rules, classify as type 'COMP':

- Question word is 'which' or 'what'
- Some synonym of 'company' like 'firm' is in the question

If a question follows these rules, classify as type 'PCT':

- Question word is 'what'
- Some form of 'percent' like 'percentage' is in the question

If a question is not classified as anything after these rules, classify as type 'MISC'.

The question processer classifies the question, but it also returns the just question content words by taking out the question word itself.

Step 2) Extract key words

I extract the key words from the content words by removing the content words that aren't needed. If a word does not exist in the document tf-idf matrix such as stop words, remove it. If the question is of type 'COMP', I remove the word 'company' or some synonym of it. Having 'company' as a key word and not as a key word were both tested, and the latter performed better. This is likely because if we are looking for companies as an answer type, many sentences containing an actual company name won't include the word 'company' in it and we don't want it sentences that have the word 'company' but no actual company name to be a candidate answer. The same goes for 'PCT' type questions as well. I remove the word 'percent' or some form of it. For 'CEO' questions, leaving 'CEO' in the key words actually performed better again likely because sentences with 'CEO' in them actually include their name. We are then left with the key words. The key words are stemmed since the words in the corpus tf-idf matrix are stemmed. If querying about the CEO of the company ‘Apple’, it is stemmed to ‘appl’ which is how it is in the matrix.

Step 3) Document retrieval

With the corpus tf-idf matrix, I give each document a score by summing the tf-idf score of each key word for each document. I pick the top 50 documents that have the highest score to be our candidate documents.

Step 4) Sentence retrieval

I break down all the candidate documents into an amalgamation of sentences. Instead of a corpus of documents, I create a corpus of sentences. From this corpus of sentences, I make a new tf-idf matrix. I also maintain a copy of the raw sentences and the NER chunked sentences. This is easy because I have already made a database of documents that are sentence tokenized and NER chunked from the very beginning (pre-processing).

I can now score and rank the sentences, but I do some filtering first. If the question type is ‘CEO’, I only keep sentences that have a ‘CEO’ chunk tag. The same goes if the question type is ‘COMP’ or ‘PCT’ but with their respective tag. If the question type is ‘MISC’, I keep all sentences. I am left with my candidate sentences. I follow the same process as the document retrieval step: I sum the tf-idf scores of each key word for each document.

Step 5) Answer

I pick the top sentence with the highest score. If the question is type ‘CEO’, I use my NER classifier to extract the string from the sentence that is tagged as ‘CEO’ which should include the name of the CEO. I do the same question type ‘COMP’ but extract the string tagged as ‘COMP’. For question type ‘PCT’, I just extract the percent myself using regex because it’s more accurate. For questions of type ‘MISC’ like (what affects GDP?), I had difficulty handling for a single direct answer. Instead, I return the top 7 ranked sentences entirely because I don’t have a sense of what my answer should be. Furthermore, I would rather the system output sentences as the answer, and the user can verify if it actually makes sense.

Analysis and Future Improvements:

The QA system can for the most part answer the types of questions we are interested in. It answers ‘CEO’ type questions very well. For ‘COMP’ type questions, it is indeed very specific to the question being asked. For example, when asking which companies went bankrupt in September 2008 and October 2013 gives the result F/F (Fannie and Freddie) and OGX which are actually correct. However, when asking which companies went bankrupt in March 2009, the result it First Republic. First Republic Group Realty LLC did actually go bankrupt in 2009, but not in March (it was June). While there are limitations to the system, there are also limitations to the corpus given to us. If the corpus doesn’t have a sentence that matches the key words being asking, then the QA system can’t give the right answer unsurprisingly. The more information we have the better so it’s likely that using the 2014 corpus as well could give better results. Furthermore, my NER classifier seems to classify CEO’s and percentages much better than it can companies as seen in homework 3. Initially, I had my QA system extract companies from the top 3 sentences as the question word ‘which’ implies one or more results. While the first result (highest ranked) would usually give sensible results (an actual company name), the subsequent results almost never made sense (not company names). Because of this I only extract from the highest ranked sentence which usually, if not always, only gives one company name. I try to value the quality of the output over the quantity when I can. For ‘MISC’ questions, I struggle to get quality answers so instead I choose quantity by outputting the highest ranked sentences meaning the user himself has to parse the output for the answer that makes the most sense which is not very QA like. Asking what affects GDP, the user has to read through the output and infer the answer. In this case it seemed to be exchange rate and inflation rate. The output for ‘PCT’ type questions is always a percentage, but never correct in this case. For example, when asking what percentage of drop or increase in GDP is associated with inflation, the answer is 2% which sounds reasonable until we output the sentence that it came from which is ‘Inflation below 2%’. Again, this is likely due to limitations in both the QA system itself along with the information in the corpus.

A rule based approach is used to classify the question due to the scope of the problem but if given a dataset of questions with labels, we would generalize better and expand the set of questions we could ask the system which would be a nice enhancement. Furthermore, a different scoring formula like okapi could be used to test if it improves the performance as there is no universally agreed upon scoring formula. Finally, the NER classifier can definitely act as a bottleneck for good results, so improving the classifier would likely output better candidate sentences and thus better results.