Production Planning

January 18, 2019

1 Production Planning

1.1 Limitations of the Proof of Concept.

I'd like to quickly talk to how the system i developed meets (or fails to meet) some of the business goals outlined in the assignment.

1.1.1 Goal 1

- Regarding the first task of efficiently retrieving SEC filings, i believe the system does fairly well. I designed the system so that there are two main functional components. One, `fileUrlGenerator.py` is responsible for scanning through index files and yielding the urls corresponding to 10Q filings. The second, `fileParser.py` is responsible for requesting the actual 10Q file, and then extracting information from it. This design lends itself nicely to a parallel producer and consumer design pattern. In my implementation, one process is responsible for filling a queue with applicable urls. Multiple other processes then read from this queue and are responsible for parsing the resulting file. This design is highly scalable; with more cores the number of individual consumer processes can be increased (for this application consumption tends to be the bottle neck). Or, alternatively, entirely separate jobs can be run with multiple producers each handling a different partition of the daily index files. On my local computer, I observed average processing rates of about 300 files a minute. A pro and con of this approach is that it doesn't require any local storage of the raw filings, instead files are processed as soon as they are requested and available in memory. It is only the final parsed results that must be stored to disk.

In a production setting, I would propose a different implementation. I think it would make sense to maintain a database of all the raw filings. There are roughly 500,000 10Q filings and I estimate their total size to be roughly on the order of 100GB. So a single non-distributed database should be sufficient. Storing the raw files in this fashion would have three main advantages. First, they could be indexed for faster searching by company name or CIK. Second, I'd expect IO to a database owned and configured by citadel to be faster than IO to Edgar. Third, there are some tasks that really do benefit from batch processing. For instance, developing NLP models, be they topic model or entity-linkage related, usually requires large training corpses. Having the raw data available in a database would make these kinds of tasks simpler. After pulling the full historical dataset, a job could be created and scheduled to periodically download new 10-Q submissions and keep the database up-to-date.

- For the second task of extracting the actual short and long term debt levels, i think the application does OK. My strategy was to develop a number of different extractors - one for text files, one for HTML files, and then one for XBRL files. Each of the extractors employs different logic and parsing rules that are really designed around some of the idiosyncrasies of these filing types. Generally, I have the most confidence in the extraction results for xbrl documents, as these filings have more standardized structure and element tagging. Text files are the hardest to parse as they're the least structured. Ultimately, the application relies upon "waterfall-style" logic, where it attempts to first parse a file as xbrl, and if that fails, it falls through to HTML parsing, and then finally to text parsing.

I think one of the challenging aspects of this assignment was determining what the definition of "short" and "long" term debt really was. Balance sheets can list a variety of different forms of debt and liabilities and deciding what is and what isn't relevant was challenging. I decided to attempt to extract every gaap-related debt line item I could, and then, if there weren't obvious matches to things that could be interpreted as long and short term debt levels, I attempted to aggregate up subfields in order to form these results. As a last resort, I took fields that were likely "supersets" of long and short term debt (i.e. total current or non-current liabilities). In production, I think this sort of logic should be developed in close consultation with a subject matter expert. That said, I still think it's a reasonable approach, and one that should be refined.

- One area where I think my application fell short was the extraction of free text related to debt levels. This was primarily just a time constraint. For xbrl documents, debt disclosure paragraphs are commonly given the element tag `us-gaap:DebtDisclosureTextBlock`. My application simply grabs the text associated with that element and returns. This solution works fairly well for xbrl documents, however, it doesn't do anything for text or HTML filings. If you wanted to expand on this I think you have two options. First, one could go the the route of developing a content model and extracting paragraphs that are related to debt. This could work, although, I think my first inclination would be to start simple and see if a set of heuristics couldn't be developed. For instance, it's likely that the debt disclosures either appear in a "Note" section towards the end of the document, or as paragraphs immediately following the balance sheet. It's possible rules could be developed that would localize searching to these regions of a document, and then look for certain debt-related expressions. This approach would still allow one to exploit some of the structure inherent in HTML documents.

1.1.2 Goal 2

- My approach to quality assessment was to devise different measurements and visualizations of how frequently extraction succeeded. "Succeeded" in this sense, means whether debt levels were ultimately extracted from a given 10-Q. I found that I was able to return short and long term debt levels for close to 90% and 70% of all filings, respectively. Plotting this success metric against time proved useful as it helped to gauge how my application faired against a shifting composition of filing types. A draw back of this approach is that it says nothing about the accuracy of the extracted values. I think in production it would make sense to spend time creating a evaluation set of data. This would likely be manually curated, and span a range of time and filing types. Then the results returned by the application could be benchmarked against those produced by a human. Having this metric would be invaluable as any changes or iterations to the application could be measured in terms of this metric's

lift. However, manual data labeling is a time consuming endeavor and I decided to forgo it in this exercise.

- For the debt analysis over time, I have some concerns about how my application performs. First there's the issue of unit-scaling between different filing types which I've mentioned several times - text and HTML documents tend to report units in 1000s, but not xbrl documents. Second, there are some patterns, that when visualized seem abnormal. For instance, on first quarter's of the year there appear to be pronounced increases in short-term debt and occasional decreases in long term debt. I'd like to know if this is a known or expected feature of company accounting. The result is striking enough, that I tend to think it's an artifact of the extraction process. Second, I think the varying performance of my application on the different filing types has a strong confounding effect on the reported results.

I think the goal that I fell shortest on meeting was the creation of a model to extract insights from the debt disclosures. There are certainly some technical challenges involved with doing this. For instance, I questioned whether or not I had a large or general enough training set to perform this activity. However, my biggest limitation was time. I think that the additional insight that could be garnered from this model would likely be limited, at least in the POC. Additionally, the disclosure text returned isn't overwhelming large, and a lot of it is relevant. So, in terms of seeking to understand a given company's debt evolution I made the decision that, for the POC, it was unduly hard reading the full disclosures.