Document Extraction and Analysis

Author: Vaed Prasad

Features

This library parses PDF files for merger agreements and merger proxy statements and generates .csv files containing relevant semantic chunks extracted from the document to facilitate analysis of previous mergers. Each .csv file contains rows delineated by a particular subsection, with each row containing the following columns:

- **Document:** The PDF file name for the extracted subsection.
- **Section Header:** The top-level section name for the extracted subsection.
- **Subsection Header:** The corresponding subsection header.
- **Page Number Start:** The page number of the first page related to the subsection, as referred to within the document.
- **True Page Number Start:** The 0-indexed true page number (inclusive) of the first page related to the subsection.
- **True Page Number End:** The 0-indexed true page number (inclusive) of the last page related to the subsection.
- **Subsection Text:** The raw text for the corresponding subsection.
- **Subsection Label:** The label most related to the corresponding subsection. Possible labels include: "Termination," "Indemnification," "Confidentiality," and "Unknown".

Example CSV File

The following table is an example of the format of the output merger analysis .csv files:

Document Name	Section Header	Subsection Header	Page Number Start	True Page Number Start	True Page Number End	Subsection Text	Subsection Label
Alpha.pdf	SUMMARY	SUMMARY	1	7	7	This summary highlights...	Termination
Beta.pdf	SUMMARY	The Merger	1	7	10	Parties involved in the merger...	Termination
Charlie.pdf	SUMMARY	Merger Consideration	4	10	13	At the Effective Time...	Unknown
Delta.pdf	FORWARD-LOOKING STATEMENTS	FORWARD-LOOKING STATEMENTS	8	14	16	This proxy statement contains forward-looking statements...	Termination
Echo.pdf	THE SPECIAL MEETING	THE SPECIAL MEETING	11	17	18	The enclosed proxy is solicited on behalf...	Confidentiality
Foxtrot.pdf	THE SPECIAL MEETING	Date, Time and Place	12	18	18	We will hold the Special Meeting virtually on...	Indemnification
Gamma.pdf	THE SPECIAL MEETING	Purpose of the Special Meeting	13	19	20	At the Special Meeting, we will ask stockholders...	Unknown

Instructions

1. If you have not already, install conda here
    - After installation, be sure to run `source ~/.bash_profile`
    - Run `conda` to verify that conda has properly been installed.
2. Unzip the `code.zip` file at your desired location:

```
unzip code.zip
```

3. Enter `code/` as your working directory:

```
cd code
```

4. Create the conda environment:

```
conda env create -f environment.yml
```

5. Activate the conda environment:

```
conda activate code
```

6. Confirm you are using the correct `python`:

```
which python
> /path/to/anaconda3/envs/code/bin/python
```

7. Set your `OPENAI_API_KEY` in `code/.env`:

```
OPENAI_API_KEY=<YOUR-OPENAI-KEY>
```

7. Move your relevant dataset of PDF files to `~/code/data/`:

8. Run `main.py`!

```
python src/main.py
```

Design Decisions

Multiprocessing

The two top-level functions in `main.py` are `generate_summary_for_directory()` and `generate_summary_for_directory_parallel()`. Initially, I implemented the naive approach of `generate_summary_for_directory()`, which sequentially iterates across each `input_file_path` one-by-one when generating the corresponding .csv file. However, when profiling my code with Scalene, I noticed that this function had low CPU-utilization and experienced high wall times due to its waiting for OpenAI to complete a prompt service response. As a result, I elected to integrate multiprocessing into this logic to enable parallel processing and distribute the workload of each across multiple CPU cores. This integration resulted in a significant reduction in total processing time. Additionally, since each process runs in its own memory space, an unexpected error in one process does not crash the entire application, enabling other PDF files to continue to be extracted and summarized. However, it is worth noting that using multiprocessing to submit a significant number of OpenAI completion requests can result in rate limit errors.

Prompting Service

In order to facilitate prompting with the OpenAI SDK, I implemented an `OpenAIPromptService` to encapsulate core error-handling, automatic-retry, and response-parsing logic. During the instantiation of an `OpenAIPromptService` object, the OpenAI API Key is securely read from the user's `.env` file to isolate this sensitive information from the source code and prevent it from being accidentally committed to version control systems where it may be exposed. In an effort to simplify the user's experience, I implemented custom error-handling logic to more effectively parse OpenAI errors through friendlier error messaging. For example, if a user makes a typo when setting their OpenAI API Key, OpenAI's `openai.AuthenticationError` will route to a `FriendlyException` suggesting to the user: `Please ensure that the OpenAI API key has correctly been set in your .env file."`

When sending numerous completion requests to OpenAI, experiencing rate limit errors is a significant concern. As a result, I implemented automatic-retries with exponential backoff to mitigate this concern. The benefits of retrying with exponential backoff is succinctly explained in OpenAI's docs:

> Retrying with exponential backoff means performing a short sleep when a rate limit error is hit, then retrying the unsuccessful request. If the request is still unsuccessful, the sleep length is increased and the process is repeated. -Automatic retries means you can recover from rate limit errors without crashes or missing data -Exponential backoff means that your first retries can be tried quickly, while still benefiting from longer delays if your first few retries fail

Table of Contents Extraction

The first key step of parsing each document is extracting the table of contents (ToC). Initially, I had opted to accomplish this in a purely programmatic approach by leveraging the assumption that top-level section headers would always be uppercase. This approach relied on retrieving links for each page, checking if the textbox for the link was strictly in uppercase and not a page number (i.e. did not match with `r"^[A-Z]-\d+$"` to account for appendixes), and extracting the true page number from the link in order to create a mapping of top-level section headers to true page numbers. However, this design had several key flaws:

- It entirely neglected subsections, undermining the quality of the document parsing.
    - Incorporating subsection utilizing this approach is challenging as the assumption that subsections are always uppercase can not be relied on, making it difficult to distinguish a linked subsection header with generic linked text.
- It only rationalized the true page number extracted from the link and could not retrieve the page number of a section as referenced within the document. This distinction can be confusing to the user as they may use the ToC as the ground truth for the index of a section.
    - *Example 1:* The Summary is on the 7th page of the PDF file but the ToC claims that it is on page 1.
    - *Example 2:* The Appendix is on the 126th page of the PDF file but the ToC claims that it is on page A-1.

Given these drawbacks, I pivoted to a model-based approach that leveraged OpenAI's chat models. I made this decision because LLMs are trained on vast amounts of text data, which enable them to understand context and semantics. As a result, this allows large language models (LLMs) to more accurately identify and differentiate between ToC entries and other text elements in a document, even when the formatting or structure is inconsistent.

First, I leverage the assumption that the ToC contains links to their corresponding sections in order to filter out pages from the document that do not reach a particular threshold of links. This is important to reduce the number of input tokens in our chat completion request and consequently reduce our inference costs. I then prompt the LLM to create a mapping from the top-level section header to subsection header to the page number denoted in the ToC entry. After some exploratory analysis I realized that top-level sections can contain text that is not included in any particular subsection; therefore, I also prompt the LLM to include each top-level section as a subsection. In addition, I more robustly preserve this ToC mapping output structure by forcing LLM response format to be a JSON.

This is an example of the JSON-loaded response from the LLM:

```
{'SUMMARY': {'SUMMARY': '1',
  'Parties Involved in the Merger': '1',
  'The Merger': '1',
  'Merger Consideration': '2',
  'Material U.S. Federal Income Tax Consequences of the Merger': '3',
  'Appraisal Rights': '3',
  'Litigation Related to the Merger': '4',
  'Regulatory Approvals Required for the Merger': '5',
  'Closing Conditions': '5',
  'Financing of the Merger': '5',
  'Required Stockholder Approval': '6',
  'The Special Meeting': '6',
  'Recommendation of the NI Board of Directors': '6',
  'Opinion of BofA Securities, Inc.': '7',
  'Interests of NI’s Executive Officers and Directors in the Merger': '7',
  'Non-Solicitation Covenant': '7',
  'Termination of the Merger Agreement': '8',
  'Effect on NI If the Merger Is Not Completed': '8'},
 'QUESTIONS AND ANSWERS': {'QUESTIONS AND ANSWERS': '9'},
 'FORWARD-LOOKING STATEMENTS': {'FORWARD-LOOKING STATEMENTS': '16'},
 'THE SPECIAL MEETING': {'THE SPECIAL MEETING': '18',
  'Date, Time and Place': '18',
  'Purpose of the Special Meeting': '18',
  'Record Date; Shares Entitled to Vote; Quorum': '18',
  'Vote Required; Abstentions and Broker Non-Votes': '18',
  'Stock Ownership and Interests of Certain Persons': '19',
  'Voting at the Special Meeting': '19',
  'Revocability of Proxies': '20',
  'Board of Directors’ Recommendation': '20',
  'Solicitation of Proxies': '21',
  'Anticipated Date of Completion of the Merger': '21',
  'Appraisal Rights': '21',
  'Delisting and Deregistration of NI Common Stock': '22',
  'Other Matters': '22',
  'Householding of Special Meeting Materials': '22',
  'Questions and Additional Information': '22'},
  ...
  }
```

Page Extraction

Although the LLM is effective at capturing the page number of the subsection denoted in its ToC entry, it can not reason about the true page number of the subsection within the entire document in order to programmatically extract that subsection's text. For example, given that page number "A-8" it is not immediately clear where to extract this raw text from the document. This is a situation where a programmatic approach of deriving the true page number may be superior relative to a model-based approach.

One assumption I initially made was that each page number denoted in the ToC contains a link to that subsection, which can be used to extract the true page number. However, I soon found documents that contained ToC that did not have links for its page numbers; nevertheless, in these situations the subsections would contain a corresponding link. Therefore, I would create a mapping of the textbox of the link to the page associated with that link. Unfortunately, this approach ran into some edge-cases on several documents where across the document a particular text string would contain a link to multiple different pages. To account for this phenomena, I adjusted the values of my mapping to be a list of potential true page number candidates and selected the smallest true page number that was on or after the preceding section's true page number.

Finally, in order to get the ending true page number of the subsection I could utilize the starting page number of the subsequent subsection. For this, I leveraged the assumption that if the next section is a top-level section it would be on a new page so `true_end_page = next_section_true_start_page - 1` . Otherwise, if the next section is a subsection within the same top-level section then `true_end_page = next_section_true_start_page` and the specific span is filtered later during **Text Extraction**.

Note: The term "true" page number is used to denote the 0-indexed page number within the document. True start page numbers and true end page numbers are inclusive.

Text Extraction

Extracting the true start page number and true end page number greatly facilitates the process of retrieving the appropriate text for the subsection. We can leverage the assumption that each subsection contains the subsection header, which is corroborated by the fact that each subsection header ToC entry links to a textbox containing subsection header in the subsection. As a result, we can retrieve the entire text for all the relevant pages and crop our text for the subsection by removing text before the first occurance of the subsection header and after the first occurance of the subsequent subsection header. If the specified subsection is that last subsection in the document we can extract the remainder of the document after finding the specific subsection's header.

Subsection Labeling

In order to classify which subsections from each document relate to "Termination", "Indemnification", or "Confidentiality," we can prompt the LLM to assign a particular label when provided the subsection's text. From prior experience, it can be challenging to perform classification with an LLM because one has no guarantees on the range of possible LLM responses; however, there are several approaches that I leveraged to improve the likelihood that the LLM returns a valid label:

1. **Prompting:** By explicitly listing the labels of the document we help prevent the LLM from selecting other unintended labels.
2. **Setting Max Output Tokens:** By constraining the number of output tokens the LLM can generate, we prevent unintended responses that do not strictly contain the label.
    - *Example Unintended Response:* `"The label for this document is Confidentiality"`
3. **Fuzzy Matching:** Fuzzy matching allows the model to handle variability in human language, recognizing that the same idea can be expressed in many different ways. Fuzzy matching techniques are also tolerant of noise and errors in the input text.
    - `map_output_to_label(output="Terminate") -> "Termination"`
4. **Unknown Label:** By including an "Unknown" label in the prompt we allow the LLM to respond with a logical label if the subsection does not match any of the other labels instead of forcing the LLM to illogically guess. In addition, if the output of fuzzy matching has no labels then we can assign the subsection with the "Unknown" label.

Future Extensions

- Leverage the `tiktoken` library to compute tokenized prompt length to accurately (instead of approximately) guarantee compliance with an LLM's context window.
- Perform additional prompt engineering to improve the LLM's prompt for ToC extraction; in particular, to be more robust in retrieving subsections within the Appendix.
- Provide detailed examples and/or leverage chain-of-thought prompting to improve the LLM's prompt for subsection labeling.
- Leverage batching during OpenAI chat completion requests for subsection labeling to further reduce rate limit concerns.