10-K Text Analysis

Eric He - eh1885 - Group 21

Project Objective

1. Test pipeline and data processing capabilities by systematically downloading years of 10-K data from the SEC EDGAR database.
    a. Bonus: also download 10-Q data and special filings.
2. Can compare changes in text filings Y/Y using text distance and sentiment analysis
    a. Bonus: plug largest differences into ChatGPT to highlight and summarize differences

Concepts

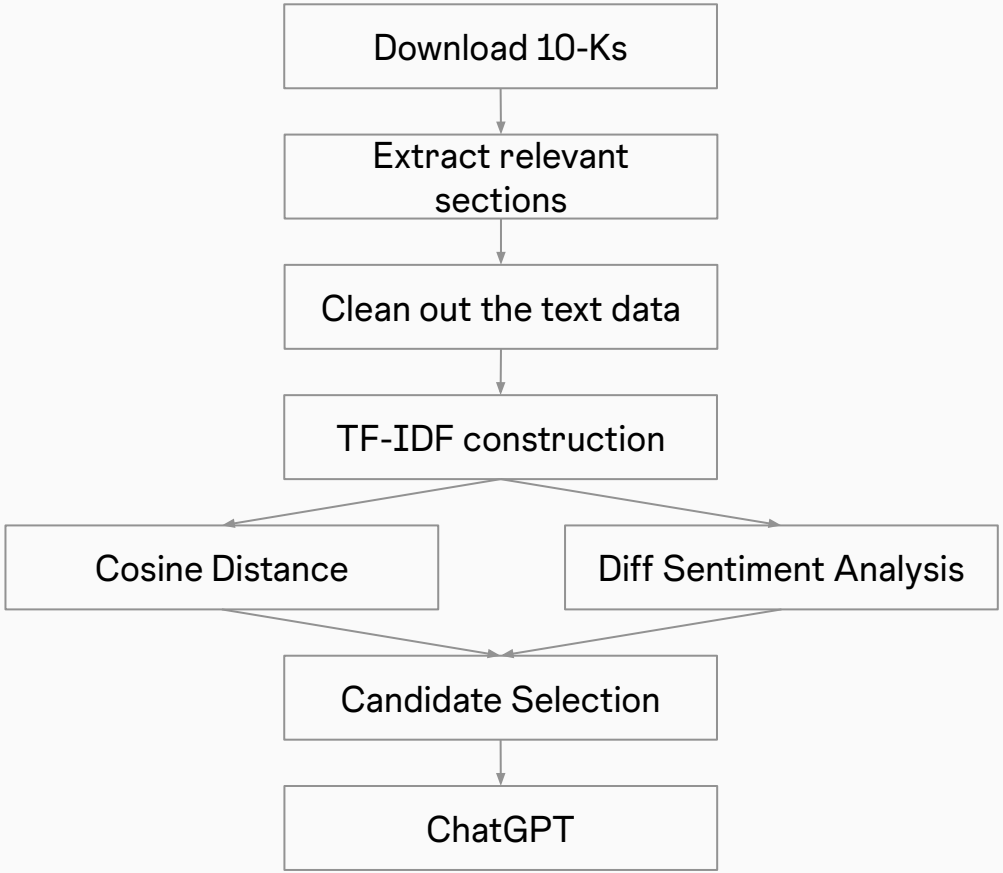
1. Systemic processing - can we parallelelize?
2. Heavy text processing - can we Cythonize or otherwise speed up?
3. Text data analysis

Background - 10Ks

```
<SEC-DOCUMENT>0001628280-16-020309.txt : 20161026
<SEC-HEADER>0001628280-16-020309.hdr.sgml : 20161026
<ACCEPTANCE-DATETIME>20161026164216
ACCESSION NUMBER:               0001628280-16-020309
CONFORMED SUBMISSION TYPE:      10-K
PUBLIC DOCUMENT COUNT:          96
CONFORMED PERIOD OF REPORT:     20160924
FILED AS OF DATE:               20161026
DATE AS OF CHANGE:              20161026

FILER:

        COMPANY DATA:
                COMPANY CONFORMED NAME:                 APPLE INC
                CENTRAL INDEX KEY:                      0000320193
                STANDARD INDUSTRIAL CLASSIFICATION:     ELECTRONIC COMPUTERS [3571]
                IRS NUMBER:                             942404110
                STATE OF INCORPORATION:                 CA
                FISCAL YEAR END:                        0924

        FILING VALUES:
                FORM TYPE:              10-K
                SEC ACT:                1934 Act
                SEC FILE NUMBER:        001-36743
                FILM NUMBER:            161953070

        BUSINESS ADDRESS:
                STREET 1:               ONE INFINITE LOOP
                CITY:                   CUPERTINO
                STATE:                  CA
                ZIP:                    95014
                BUSINESS PHONE:         (408) 996-1010

        MAIL ADDRESS:
                STREET 1:               ONE INFINITE LOOP
                CITY:                   CUPERTINO
                STATE:                  CA
                ZIP:                    95014

        FORMER COMPANY:
                FORMER CONFORMED NAME:  APPLE COMPUTER INC
                DATE OF NAME CHANGE:    19970808
</SEC-HEADER>
<DOCUMENT>
<TYPE>10-K
<SEQUENCE>1
<FILENAME>a201610-k9242016.htm
<DESCRIPTION>10-K
<TEXT>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"
<html>
        <head>
                <!-- Document created using Wdesk 1 -->
                <!-- Copyright 2016 Workiva -->
                <title>Document</title>
```

1. 10-Ks are financial statements filed by publicly traded companies with the SEC, and can be downloaded for free.
2. We can parse 10-Ks to surface disclosures and market trends, which potentially can be used for trading signal.
3. The raw data consists of 26 sections and associated filings. We are primarily interested in Sections 1 (Business), 1A (Risk Factors), 7 (Management Discussion), 7A (Market Risks)
4. Sections may have boilerplate text, raw LXML code, and requires a fairly intricate cleaning process to split 10-K or 10-Q into component sections.
5. Downloading all the data, uncompressed, is several tens of gigabytes.

The computation graph



Optimization Summary

Computation	Methods Tried	Commentary
Download 10-Ks	Multiprocessing	Got throttled by the SEC, had to rate limit the downloads.
Extract Relevant Sections	Cython, Multiprocessing	Cython was in fact slower due to higher overhead Multiprocessing had near-linear speedup.
Clean out the text data	Multiprocessing, Vectorization	Vectorized operations actually had memory blowup, had to use multiprocessing.
TF-IDF Construction	-	Could not find a faster, memory-safe way than sklearn native TfIdfVectorizer
Cosine Distance	Native sklearn functionality, Manual matrix multiplication, Numba matrix multiplication	Native sklearn cosine distance does a lot of extra computations and was not possible to scale to full dataset. Manual matrix multiplication was much faster, but Numba did not see meaningful speedup.
Sentiment Analysis	-	Haven’t tried yet.
Candidate Selection	-	Haven’t tried yet.
ChatGPT	-	Haven’t tried yet, also unlikely to be able to speed up

Extract Relevant Sections

Cython was unable to speed up the computation.

As this is dumbly parallelizable across 10-Ks, multiprocessing was able to significantly speed up the computation.

Method (10 filings at a time)	Time
Standard	3.22 s ± 154 ms per loop
Standard w/ Multiprocessing	1.81 s ± 179 ms per loop
Simple Cython	4.39 s ± 77.1 ms per loop
Simple Cython w/ Multiprocessing	2.54 s ± 31.5 ms per loop
Cythonized w/ declarations	3.27 s ± 190 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

https://github.com/EricHe98/Financial-Statements-Text-Analysis/blob/master/analyses/optimize_10k_cleaning.ipynb

Extract Relevant Sections

Identify sections 1, 1A, 7, 7A from the text document using regex search. Error-prone.

About 20% of parses do not return all the relevant sections, and maybe 3% of statements fail parse entirely.

```python
def parse_filing(text):
    regex_10k = re.search(r"(?s)(?m)<TYPE>10-K.*?(</TEXT>)", text)
    regex_items = re.compile(r"(> +Item|>Item|^Item|ITEM)((\s| | )(1A|1B|7A|7|8))")

    try:
        extracted_text = regex_10k.group(0)
    except:
        print('❌ No 10-K match for file')

    item_matches = regex_items.finditer(extracted_text)
    matches_df = pd.DataFrame([(x.group(), x.start()) for x in item_matches])
    matches_df.columns = ['item', 'start']
    matches_df['item'] = matches_df['item'].str.lower().replace('>|\.| | | |;', '', regex=True)
    matches_df = matches_df.sort_values('start').drop_duplicates(subset=['item'], keep='last')
    matches_df['end'] = matches_df['start'].shift(-1).fillna(0).astype(int)
    matches_df = matches_df[matches_df['item'].isin(['item1a', 'item1b', 'item7', 'item7a'])]
    if len(matches_df) != 4:
        print('❌ Only {sections} sections found for file'.format(sections=len(matches_df)))
    matches_df['text'] = matches_df\
        .apply(lambda row: BeautifulSoup(extracted_text[row['start']:row['end']], 'lxml').get_text('\n\n'), axis=1)

    return matches_df
```

Bulk of computation time is in these two steps. No super obvious way for C or parallelization to plug in, would need larger rewrite of the logic for faster computation.

https://github.com/EricHe98/Financial-Statements-Text-Analysis/blob/master/analyses/optimize_10k_cleaning.ipynb

Clean out text data

The standard String library was slower than native Pandas str functionality. However, pandas str implementation was memory inefficient and could not scale to the full dataset. Using standard w/ multiprocessing, we could also actually beat pandas on computation time.

Method (1k filings at a time)	Time
Standard	4.95 s ± 106 ms per loop
Pandas str functionality	3.54 s ± 107 ms per loop
Standard w/ Multiprocessing	2.34 s ± 106 ms per loop

Pandas Str has a very memory inefficient step

```
%%timeit
df.head(1000)['text'].str.replace('\W', ' ', regex=True)\
    .str.lower()\
    .str.split()\
    .str.join(' ')
```

```
def clean_string(s):
    s = re.sub('\W', ' ', s)
    s = s.lower()
    s = re.sub(' +', ' ', s)
    return s
```

https://github.com/EricHe98/Financial-Statements-Text-Analysis/blob/master/analyses/optimize_cosine_distance.ipynb

Cosine Distance

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
%%timeit
cosine_similarity(tfidf[:1000], tfidf[:1000])
```

```
742 ms ± 30.5 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
%%timeit
cosine_similarity(tfidf[:2000], tfidf[:2000])
```

```
2.67 s ± 65.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Quadratic scaling in time

```
%%timeit
(tfidf[:999].multiply(tfidf[1:1000]).sum(axis=1) / \
     np.sqrt(tfidf[:999].multiply(tfidf[:999]).sum(axis=1)) / np.sqrt(tfidf[1:1000].multiply(tfidf[1:1000]).sum(axis=1)))
```

```
76 ms ± 3.31 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
%%timeit
(tfidf[:1999].multiply(tfidf[1:2000]).sum(axis=1) / \
     np.sqrt(tfidf[:1999].multiply(tfidf[:1999]).sum(axis=1)) / np.sqrt(tfidf[1:2000].multiply(tfidf[1:2000]).sum(axis=1)))
```

```
172 ms ± 4.07 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

near-Linear scaling

```
%%timeit
(tfidf[:-1].multiply(tfidf[1:]).sum(axis=1) / \
     np.sqrt(tfidf[:-1].multiply(tfidf[:-1]).sum(axis=1)) / np.sqrt(tfidf[1:].multiply(tfidf[1:]).sum(axis=1)))
```

```
1.51 s ± 43.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Thank You

Questions?