preprocessing

September 29, 2021

1 Word vectors from SEC filings using gensim

In this section, we will learn word and phrase vectors from annual SEC filings using gensim to illustrate the potential value of word embeddings for algorithmic trading. In the following sections, we will combine these vectors as features with price returns to train neural networks to predict equity prices from the content of security filings.

In particular, we use a dataset containing over 22,000 10-K annual reports from the period 2013-2016 that are filed by listed companies and contain both financial information and management commentary (see chapter 3 on Alternative Data). For about half of 11K filings for companies that we have stock prices to label the data for predictive modeling

1.1 Imports & Settings

[2]:
```
from pathlib import Path
import numpy as np
import pandas as pd
from time import time
from collections import Counter
import logging
from gensim.models import Word2Vec
from gensim.models.word2vec import LineSentence
```

[3]:
```
pd.set_option('display.expand_frame_repr', False)
np.random.seed(42)
```

[ ]:
```
def format_time(t):
    m, s = divmod(t, 60)
    h, m = divmod(m, 60)
    return '{:02.0f}:{:02.0f}:{:02.0f}'.format(h, m, s)
```

1.1.1 Logging Setup

[4]:
```
logging.basicConfig(
        filename='preprocessing.log',
        level=logging.DEBUG,
        format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
        datefmt='%H:%M:%S')
```

1.1.2 Paths

Each filing is a separate text file and a master index contains filing metadata. We extract the most informative sections, namely - Item 1 and 1A: Business and Risk Factors - Item 7 and 7A: Management's Discussion and Disclosures about Market Risks

The notebook preprocessing shows how to parse and tokenize the text using spaCy, similar to the approach in chapter 14. We do not lemmatize the tokens to preserve nuances of word usage.

We use gensim to detect phrases. The Phrases module scores the tokens and the Phraser class transforms the text data accordingly. The notebook shows how to repeat the process to create longer phrases.

[80]:
```
filing_path = Path('data/filings')
```

[ ]:
```
sections_path = Path('data/sections')
if not sections_path.exists():
    sections_path.mkdir(exist_ok=True)
```

1.2 Identify Sections

[ ]:
```
for i, filing in enumerate(filing_path.glob('*.txt')):
    if i % 500 == 0:
        print(i, end=' ', flush=True)
    filing_id = int(filing.stem)
    items = {}
    for section in filing.read_text().lower().split('°'):
        if section.startswith('item '):
            if len(section.split()) > 1:
                item = section.split()[1].replace('.', '').replace(':', '').
↪replace(',', '')
                text = ' '.join([t for t in section.split()[2:]])
                    if items.get(item) is None or len(items.get(item)) <␣
↪len(text):
                        items[item] = text

    txt = pd.Series(items).reset_index()
    txt.columns = ['item', 'text']
    txt.to_csv(sections_path / (filing.stem + '.csv'), index=False)
```

1.3 Parse Sections

Select the following sections:

[81]:
```
sections = ['1', '1a', '7', '7a']
```

[ ]:
```
clean_path = Path('data/selected_sections')
if not clean_path.exists():
    clean_path.mkdir(exist_ok=True)
```

[ ]:
```
nlp = spacy.load('en', disable=['ner'])
nlp.max_length = 6000000
```

[ ]:
```
vocab = Counter()
t = total_tokens = 0
stats = []

start = time()
done = 1
for text_file in sections_path.glob('*.csv'):
    file_id = int(text_file.stem)
    clean_file = clean_path / f'{file_id}.csv'
    if clean_file.exists():
        continue
    items = pd.read_csv(text_file).dropna()
    items.item = items.item.astype(str)
    items = items[items.item.isin(sections)]
    if done % 100 == 0:
        duration = time() - start
        to_go = (to_do - done) * duration / done
        print(f'{done:>5}\t{format_time(duration)}\t{total_tokens / duration:,.
 ↪0f}\t{format_time(to_go)}')

    clean_doc = []
    for _, (item, text) in items.iterrows():
        doc = nlp(text)
        for s, sentence in enumerate(doc.sents):
            clean_sentence = []
            if sentence is not None:
                for t, token in enumerate(sentence, 1):
                    if not any([token.is_stop,
                                token.is_digit,
                                not token.is_alpha,
                                token.is_punct,
                                token.is_space,
                                token.lemma_ == '-PRON-',
                                token.pos_ in ['PUNCT', 'SYM', 'X']]):
                        clean_sentence.append(token.text.lower())
                total_tokens += t
                if len(clean_sentence) > 0:
                    clean_doc.append([item, s, ' '.join(clean_sentence)])
    (pd.DataFrame(clean_doc,
                  columns=['item', 'sentence', 'text'])
     .dropna()
     .to_csv(clean_file, index=False))
    done += 1
```

1.4 Create ngrams

[4]:
```
ngram_path = Path('data', 'ngrams')
stats_path = Path('corpus_stats')
```

[5]:
```
def create_unigrams(min_length=3):
    texts = []
    sentence_counter = Counter()
    unigrams = ngram_path / 'ngrams_1.txt'
    vocab = Counter()
    for f in path.glob('*.csv'):
        df = pd.read_csv(f)
        df.item = df.item.astype(str)
        df = df[df.item.isin(items)]
        sentence_counter.update(df.groupby('item').size().to_dict())
        for sentence in df.text.str.split().tolist():
            if len(sentence) >= min_length:
                vocab.update(sentence)
                texts.append(' '.join(sentence))
    (pd.DataFrame(sentence_counter.most_common(),
                  columns=['item', 'sentences'])
     .to_csv(stats_path / 'selected_sentences.csv', index=False))
    (pd.DataFrame(vocab.most_common(), columns=['token', 'n'])
     .to_csv(stats_path / 'sections_vocab.csv', index=False))
    unigrams.write_text('\n'.join(texts))
    return [l.split() for l in texts]
```

[ ]:
```
start = time()
if not unigrams.exists():
    texts = create_unigrams()
else:
    texts = [l.split() for l in unigrams.open()]
print('Reading: ', format_time(time() - start))
```

[ ]:
```
def create_ngrams(max_length=3):
    """Using gensim to create ngrams"""

    n_grams = pd.DataFrame()
    start = time()
    for n in range(2, max_length + 1):
        print(n, end=' ', flush=True)

        sentences = LineSentence(f'ngrams_{n - 1}.txt')
        phrases = Phrases(sentences=sentences,
                          min_count=25,  # ignore terms with a lower count
                          threshold=0.5,  # accept phrases with higher score
```

```
                          max_vocab_size=40000000,  # prune of less common
↪words to limit memory use
                          delimiter=b'_',  # how to join ngram tokens
                          progress_per=50000,  # log progress every
                          scoring='npmi')

        s = pd.DataFrame([[k.decode('utf-8'), v]
                          for k, v in phrases.export_phrases(sentences)]
                         , columns=['phrase', 'score']).assign(length=n)

        n_grams = pd.concat([n_grams, s])
        grams = Phraser(phrases)
        sentences = grams[sentences]
        Path(f'ngrams_{n}.txt').write_text('\n'.join([' '.join(s) for s in
↪sentences]))

    n_grams = n_grams.sort_values('score', ascending=False)
    n_grams.phrase = n_grams.phrase.str.replace('_', ' ')
    n_grams['ngram'] = n_grams.phrase.str.replace(' ', '_')

    with pd.HDFStore('vocab.h5') as store:
        store.put('ngrams', n_grams)

    print('\n\tDuration: ', format_time(time() - start))
    print('\tngrams: {:,d}\n'.format(len(n_grams)))
    print(n_grams.groupby('length').size())
```

[ ]:
```
create_ngrams()
```

1.5 Inspect Corpus

[40]:
```
ngrams = pd.read_parquet('corpus_stats/ngrams.parquet')
```

[41]:
```
ngrams.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 721562 entries, 10742145 to 4887103
Data columns (total 3 columns):
phrase    721562 non-null object
score     721562 non-null float64
length    721562 non-null int64
dtypes: float64(1), int64(1), object(1)
memory usage: 22.0+ MB
```

[46]:
```
percentiles=np.arange(.1, 1, .1).round(2)
ngrams.score.describe(percentiles=percentiles)
```

[46]:
```
count    721562.000000
mean          0.631225
std           0.125067
min           0.500000
10%           0.512507
20%           0.526746
30%           0.543690
40%           0.564299
50%           0.589516
60%           0.621228
70%           0.663055
80%           0.722132
90%           0.824150
max           1.000000
Name: score, dtype: float64
```

[72]:
```
ngrams[ngrams.score>.7].sort_values(['length', 'score']).head(10)
```

[72]:

	phrase	score	length
13138522	topsoe uop	0.700002	2
22155584	aastra prairiefyre	0.700009	2
21581977	sre tre	0.700009	2
9717859	twp nng	0.700017	2
1507180	ecomobile telkonet	0.700017	2
26474295	knsd kxas	0.700017	2
17960106	oxalate ssri	0.700017	2
6936430	swirl estimote	0.700017	2
25398447	gdtna gdte	0.700024	2
14638108	chun guang	0.700024	2

[51]:
```
vocab = pd.read_csv('corpus_stats/sections_vocab.csv').dropna()
```

[52]:
```
vocab.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 201443 entries, 0 to 201444
Data columns (total 2 columns):
token    201443 non-null object
n        201443 non-null int64
dtypes: int64(1), object(1)
memory usage: 4.6+ MB
```

[53]:
```
vocab.n.describe(percentiles).astype(int)
```

[53]:
```
count    201443
mean       1440
std       22366
```

```
min            1
10%            1
20%            2
30%            3
40%            4
50%            7
60%           12
70%           24
80%           61
90%          260
max      2576751
Name: n, dtype: int64
```

[57]:
```
tokens = Counter()
for l in Path('data', 'ngrams', 'ngrams_3.txt').open():
    tokens.update(l.split())
```

[58]:
```
tokens = pd.DataFrame(tokens.most_common(),
                      columns=['token', 'count'])
```

[59]:
```
tokens.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 664963 entries, 0 to 664962
Data columns (total 2 columns):
token    664963 non-null object
count    664963 non-null int64
dtypes: int64(1), object(1)
memory usage: 10.1+ MB
```

[60]:
```
tokens.loc[tokens.token.str.contains('_'), 'count'].describe(percentiles).
↪astype(int)
```

[60]:
```
count    546779
mean         56
std        1947
min           1
10%           1
20%           1
30%           2
40%           2
50%           3
60%           3
70%           4
80%           6
90%          13
max      513694
```

```
Name: count, dtype: int64
```

[74]:
```
tokens[tokens.token.str.contains('_')].head(20).to_csv('ngram_examples.csv',␣
↪index=False)
```

1.6 Get returns

[ ]:
```
with pd.HDFStore('../data/assets.h5') as store:
    stocks = store['quandl/wiki/stocks']
    prices = store['quandl/wiki/prices'].adj_close
```

[ ]:
```
sec = pd.read_csv('data/report_index.csv').rename(columns=str.lower)
sec.date_filed = pd.to_datetime(sec.date_filed)
```

[ ]:
```
idx = pd.IndexSlice
```

[ ]:
```
first = sec.date_filed.min() + relativedelta(months=-1)
last = sec.date_filed.max() + relativedelta(months=1)
prices = (prices
          .loc[idx[first:last, :]]
          .unstack().resample('D')
          .ffill()
          .dropna(how='all', axis=1)
          .filter(sec.ticker.unique()))
```

[ ]:
```
sec = sec.loc[sec.ticker.isin(prices.columns), ['ticker', 'date_filed']]

price_data = []
for ticker, date in sec.values.tolist():
    target = date + relativedelta(months=1)
    s = prices.loc[date: target, ticker]
    price_data.append(s.iloc[-1] / s.iloc[0] - 1)

df = pd.DataFrame(price_data,
                  columns=['returns'],
                  index=sec.index)

print(df.returns.describe())
sec['returns'] = price_data
print(sec.info())
sec.dropna().to_csv('data/sec_returns.csv', index=False)
```