

07_sec_word2vec

September 29, 2021

1 Word vectors from SEC filings using Gensim: word2vec model

1.1 Imports & Settings

```
[1]: import warnings
warnings.filterwarnings('ignore')
```

```
[2]: from pathlib import Path
from time import time
import logging

import numpy as np
import pandas as pd

from gensim.models import Word2Vec, KeyedVectors
from gensim.models.word2vec import LineSentence

import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
import seaborn as sns
```

```
[3]: sns.set_style('white')
np.random.seed(42)
```

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

1.1.1 Paths

```
[5]: sec_path = Path('.', 'data', 'sec-filings')
ngram_path = sec_path / 'ngrams'
```

```
[6]: results_path = Path('results', 'sec-filings')

model_path = results_path / 'models'
```

```

if not model_path.exists():
    model_path.mkdir(parents=True)

log_path = results_path / 'logs'
if not log_path.exists():
    log_path.mkdir(parents=True)

```

1.1.2 Logging Setup

```

[7]: logging.basicConfig(
    filename=log_path / 'word2vec.log',
    level=logging.DEBUG,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    datefmt='%H:%M:%S')

```

1.2 word2vec

```

[22]: analogies_path = Path('data', 'analogies-en.txt')

```

1.2.1 Set up Sentence Generator

```

[9]: NGRAMS = 2

```

To facilitate memory-efficient text ingestion, the `LineSentence` class creates a generator from individual sentences contained in the provided text file:

```

[10]: sentence_path = ngram_path / f'ngrams_{NGRAMS}.txt'
      sentences = LineSentence(sentence_path)

```

1.2.2 Train word2vec Model

The `gensim.models.word2vec` class implements the skipgram and CBOW architectures.

```

[11]: start = time()
      model = Word2Vec(sentences,
                      sg=1,           # 1 for skip-gram; otherwise CBOW
                      hs=0,           # hierarchical softmax if 1, negative sampling
                      ↪ if 0
                      size=300,       # Vector dimensionality
                      window=5,       # Max distance betw. current and predicted word
                      min_count=50,    # Ignore words with lower frequency
                      negative=15,     # noise word count for negative sampling
                      workers=4,       # no threads
                      iter=1,          # no epochs = iterations over corpus
                      alpha=0.05,     # initial learning rate
                      min_alpha=0.0001 # final learning rate
                      )

```

```
print('Duration:', format_time(time() - start))
```

Duration: 00:27:05

1.2.3 Persist model & vectors

```
[12]: model.save((model_path / 'word2vec_0.model').as_posix())
      model.wv.save((model_path / 'word_vectors_0.bin').as_posix())
```

1.2.4 Load model and vectors

```
[13]: model = Word2Vec.load((model_path / 'word2vec_0.model').as_posix())
```

```
[14]: wv = KeyedVectors.load((model_path / 'word_vectors_0.bin').as_posix())
```

1.2.5 Get vocabulary

```
[15]: vocab = []
      for k, _ in model.wv.vocab.items():
          v_ = model.wv.vocab[k]
          vocab.append([k, v_.index, v_.count])
```

```
[16]: vocab = (pd.DataFrame(vocab,
                          columns=['token', 'idx', 'count'])
             .sort_values('count', ascending=False))
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 57384 entries, 715 to 25739
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  -
 0   token   57384 non-null    object
 1   idx     57384 non-null    int64
 2   count   57384 non-null    int64
dtypes: int64(2), object(1)
memory usage: 1.8+ MB
```

```
[18]: vocab.head(10)
```

```
[18]:
```

	token	idx	count
715	million	0	2340187
0	business	1	1696732
2029	december	2	1512367
6	company	3	1490617
171	products	4	1367413

2034	net	5	1246820
139	market	6	1148002
350	including	7	1109821
1623	sales	8	1095619
2244	costs	9	1018821

```
[19]: vocab['count'].describe(percentiles=np.arange(.1, 1, .1)).astype(int)
```

```
[19]: count      57384
      mean       4523
      std       35191
      min        50
      10%        60
      20%        75
      30%        96
      40%       128
      50%       176
      60%       263
      70%       442
      80%       946
      90%      3666
      max     2340187
      Name: count, dtype: int64
```

1.2.6 Evaluate Analogies

```
[23]: def accuracy_by_category(acc, detail=True):
      results = [[c['section'], len(c['correct']), len(c['incorrect'])] for c in acc]
      results = pd.DataFrame(results, columns=['category', 'correct', 'incorrect'])
      results['average'] = results.correct.div(results[['correct', 'incorrect']].sum(1))
      if detail:
          print(results.sort_values('average', ascending=False))
      return results.loc[results.category=='total', ['correct', 'incorrect', 'average']].squeeze().tolist()
```

```
[24]: detailed_accuracy = model.wv.accuracy(analogies_path.as_posix(),
      case_insensitive=True)
```

```
[25]: summary = accuracy_by_category(detailed_accuracy)
```

	category	correct	incorrect	average
10	gram6-nationality-adjective	367	445	0.451970
12	gram8-plural	228	372	0.380000
7	gram3-comparative	260	610	0.298851

14	total	2083	5543	0.273145
9	gram5-present-participle	100	280	0.263158
2	city-in-state	767	2151	0.262851
13	gram9-plural-verbs	75	231	0.245098
6	gram2-opposite	47	163	0.223810
5	gram1-adjective-to-adverb	59	247	0.192810
8	gram4-superlative	46	194	0.191667
11	gram7-past-tense	119	531	0.183077
4	family	4	26	0.133333
0	capital-common-countries	8	102	0.072727
3	currency	3	149	0.019737
1	capital-world	0	42	0.000000

```
[26]: def eval_analogies(w2v, max_vocab=15000):
    accuracy = w2v.wv.accuracy(analogies_path,
                                restrict_vocab=15000,
                                case_insensitive=True)

    return (pd.DataFrame([[c['section'],
                            len(c['correct']),
                            len(c['incorrect'])] for c in accuracy],
                          columns=['category', 'correct', 'incorrect'])
            .assign(average=lambda x:
                    x.correct.div(x.correct.add(x.incorrect))))
```

```
[27]: def total_accuracy(w2v):
    df = eval_analogies(w2v)
    return df.loc[df.category == 'total', ['correct', 'incorrect', 'average']].
    ↪squeeze().tolist()
```

```
[28]: accuracy = eval_analogies(model)
accuracy
```

	category	correct	incorrect	average
0	capital-common-countries	2	10	0.166667
1	capital-world	0	0	NaN
2	city-in-state	231	493	0.319061
3	currency	3	49	0.057692
4	family	0	0	NaN
5	gram1-adjective-to-adverb	44	166	0.209524
6	gram2-opposite	17	73	0.188889
7	gram3-comparative	234	366	0.390000
8	gram4-superlative	22	68	0.244444
9	gram5-present-participle	91	215	0.297386
10	gram6-nationality-adjective	268	194	0.580087
11	gram7-past-tense	111	351	0.240260
12	gram8-plural	96	86	0.527473
13	gram9-plural-verbs	74	136	0.352381

14	total	1193	2207	0.350882
----	-------	------	------	----------

1.2.7 Validate Vector Arithmetic

```
[29]: sims=model.wv.most_similar(positive=['iphone'], restrict_vocab=15000)
print(pd.DataFrame(sims, columns=['term', 'similarity']))
```

	term	similarity
0	ipad	0.691182
1	android	0.632260
2	app	0.609227
3	smartphone	0.605110
4	smart_phone	0.580258
5	smartphones	0.577489
6	keyboard	0.559338
7	mobile_app	0.525289
8	downloaded	0.520682
9	pc	0.511132

```
[30]: analogy = model.wv.most_similar(positive=['france', 'london'],
                                         negative=['paris'],
                                         restrict_vocab=15000)
print(pd.DataFrame(analogy, columns=['term', 'similarity']))
```

	term	similarity
0	united_kingdom	0.577512
1	germany	0.561950
2	belgium	0.542092
3	netherlands	0.537289
4	australia	0.515880
5	italy	0.512862
6	sweden	0.502201
7	spain	0.496639
8	singapore	0.495658
9	austria	0.494052

1.2.8 Check similarity for random words

```
[31]: VALID_SET = 5 # Random set of words to get nearest neighbors for
VALID_WINDOW = 100 # Most frequent words to draw validation set from
valid_examples = np.random.choice(VALID_WINDOW, size=VALID_SET, replace=False)
similars = pd.DataFrame()

for id in sorted(valid_examples):
    word = vocab.loc[id, 'token']
    similars[word] = [s[0] for s in model.wv.most_similar(word)]
similars
```

```
[31]:
```

	paradigm	fundamentally	patient \
0	paradigms	transform	patients
1	revolutionize	transforming	clinician
2	personalized_medicine	profoundly	physician
3	evolutionary	alter	outpatients
4	muscular_dystrophies	way	givers
5	adoptive	transformational	inpatient
6	radiance	revolutionized	hospital
7	thyroid_nodule	reshape	protection_affordable
8	mx_icp	paradigms	ambulation
9	invasive_candida	converging	ophthalmologist

	quality	percent
0	cleanliness	percent_percent
1	high_quality	percentage
2	originality	total
3	dependability	approximately
4	timeliness	basis_points
5	consistency	mwh_mwh
6	trustworthiness	respectively
7	freshness	percentages
8	friendliness	rces
9	professionalism	wheel_rvs

1.3 Continue Training

```
[ ]: accuracies = [summary]
best_accuracy = summary[-1]
for i in range(1, 15):
    start = time()
    model.train(sentences, epochs=1, total_examples=model.corpus_count)
    detailed_accuracy = model.wv.accuracy(analogies_path)
    accuracies.append(accuracy_by_category(detailed_accuracy, detail=False))
    print(f'{i:02} | Duration: {format_time(time() - start)} | Accuracy:␣
→{accuracies[-1][-1]:.2%} ')
    if accuracies[-1][-1] > best_accuracy:
        model.save((model_path / f'word2vec_{i:02}.model').as_posix())
        model.wv.save((model_path / f'word_vectors_{i:02}.bin').as_posix())
        best_accuracy = accuracies[-1][-1]
(pd.DataFrame(accuracies,
               columns=['correct', 'wrong', 'average'])
 .to_csv(model_path / 'accuracies.csv', index=False))
model.wv.save((model_path / 'word_vectors_final.bin').as_posix())
```

1.3.1 Sample Output

Epoch	Duration	Accuracy
01	00:14:00	31.64%
02	00:14:21	31.72%
03	00:14:34	33.65%
04	00:16:11	34.03%
05	00:13:51	33.04%
06	00:13:46	33.28%
07	00:13:51	33.10%
08	00:13:54	34.11%
09	00:13:54	33.70%
10	00:13:55	34.09%
11	00:13:57	35.06%
12	00:13:38	33.79%
13	00:13:26	32.40%

```
[ ]: (pd.DataFrame(accuracies,
                  columns=['correct', 'wrong', 'average'])
      .to_csv(results_path / 'accuracies.csv', index=False))

[ ]: best_model = Word2Vec.load((results_path / 'word2vec_11.model').as_posix())

[ ]: detailed_accuracy = best_model.wv.accuracy(analogies_path.as_posix(),
      ↪case_insensitive=True)

[ ]: summary = accuracy_by_category(detailed_accuracy)
      print('Base Accuracy: Correct {:.0f} | Wrong {:.0f} | Avg {:.2%}\n'.
      ↪format(*summary))

[ ]: cat_dict = {'capital-common-countries': 'Capitals',
                'capital-world': 'Capitals RoW',
                'city-in-state': 'City-State',
                'currency': 'Currency',
                'family': 'Famliy',
                'gram1-adjective-to-adverb': 'Adj-Adverb',
                'gram2-opposite': 'Opposite',
                'gram3-comparative': 'Comparative',
                'gram4-superlative': 'Superlative',
                'gram5-present-participle': 'Pres. Part.',
                'gram6-nationality-adjective': 'Nationality',
                'gram7-past-tense': 'Past Tense',
                'gram8-plural': 'Plural',
                'gram9-plural-verbs': 'Plural Verbs',
                'total': 'Total'}

[ ]: results = [[c['section'], len(c['correct']), len(c['incorrect'])] for c in
      ↪detailed_accuracy]
```



```

results = pd.DataFrame(results, columns=['category', 'correct', 'incorrect'])
results['category'] = results.category.map(cat_dict)
results['average'] = results.correct.div(results[['correct', 'incorrect']].
    ↳sum(1))
results = results.rename(columns=str.capitalize).set_index('Category')
total = results.loc['Total']
results = results.drop('Total')

```

```

[ ]: most_sim = best_model.wv.most_similar(positive=['woman', 'king'],
    ↳negative=['man'], topn=20)
pd.DataFrame(most_sim, columns=['token', 'similarity'])

```

```

[ ]: fig, axes = plt.subplots(figsize=(16, 5), ncols=2)

axes[0] = results.loc[:, ['Correct', 'Incorrect']].plot.bar(stacked=True,
    ↳ax=axes[0]
    , title='Analogy
    ↳Accuracy')
ax1 = results.loc[:, ['Average']].plot(ax=axes[0], secondary_y=True, lw=1,
    ↳c='k', rot=35)
ax1.yaxis.set_major_formatter(FuncFormatter(lambda y, _: '{:.0%}'.format(y)))

(pd.DataFrame(most_sim, columns=['token', 'similarity'])
 .set_index('token').similarity
 .sort_values().tail(10).plot.barh(xlim=(.3, .37), ax=axes[1], title='Closest
    ↳matches for Woman + King - Man'))
fig.tight_layout();

```