05_financial_news_word2vec_gensim

September 29, 2021

1 How to train your own word vector embeddings using Gensim

Many tasks require embeddings or domain-specific vocabulary that pre-trained models based on a generic corpus may not represent well or at all. Standard word2vec models are not able to assign vectors to out-of-vocabulary words and instead use a default vector that reduces their predictive value.

E.g., when working with industry-specific documents, the vocabulary or its usage may change over time as new technologies or products emerge. As a result, the embeddings need to evolve as well. In addition, corporate earnings releases use nuanced language not fully reflected in Glove vectors pre-trained on Wikipedia articles.

In this notebook we illustrate the more performant gensim adaptation of the code provided by the word2vec authors.

To illustrate the word2vec network architecture, we use the Financial News data that we first introduced in chapter 14 on Topic Modeling.

1.1 Imports

[1]:
```
%matplotlib inline
import warnings
from time import time
from collections import Counter
from pathlib import Path
import pandas as pd
import numpy as np
from numpy.linalg import norm
from scipy.spatial.distance import cdist, cosine

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

from gensim.models import Word2Vec, KeyedVectors
from gensim.models.word2vec import LineSentence
from sklearn.decomposition import IncrementalPCA
```

1.1.1 Settings

[2]:
```
warnings.filterwarnings('ignore')
sns.set_style('whitegrid')
pd.set_option('float_format', '{:,.2f}'.format)
np.random.seed(42)
```

1.1.2 Paths

[3]:
```
news_path = Path('data', 'fin_news')
data_path = news_path / 'data'
analogy_path = Path('data', 'analogies-en.txt')
```

[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}'
```

1.2 Model Configuration

[5]:
```
gensim_path = news_path / 'gensim'
if not gensim_path.exists():
    gensim_path.mkdir(parents=True, exist_ok=True)
```

[6]:
```
NGRAMS = 3           # Longest ngram in text
MIN_FREQ = 100
WINDOW_SIZE = 5
EMBEDDING_SIZE = 300
NEGATIVE_SAMPLES = 20
EPOCHS = 1
```

[7]:
```
FILE_NAME = f'articles_{NGRAMS}_grams.txt'
```

1.3 Sentence Generator

[8]:
```
sentence_path = data_path / FILE_NAME
sentences = LineSentence(str(sentence_path))
```

1.4 Train word2vec Model

[9]:
```
start = time()
model = Word2Vec(sentences,
                 sg=1, # set to 1 for skipgram; CBOW otherwise
                 size=EMBEDDING_SIZE,
                 window=WINDOW_SIZE,
                 min_count=MIN_FREQ,
```

```
                  negative=NEGATIVE_SAMPLES,
                  workers=8,
                  iter=EPOCHS,
                  alpha=0.05)

# persist model
model.save(str(gensim_path / 'word2vec.model'))

# persist word vectors
model.wv.save(str(gensim_path / 'word_vectors.bin'))
print('Duration:', format_time(time() - start))
```

```
Duration: 00:02:50
```

1.5 Evaluate results

[10]:
```
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'}
```

[11]:
```
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()
```

[12]:
```
# gensim computes accuracy based on source text files
detailed_accuracy = model.wv.accuracy(analogy_path.as_posix(),␣
 ↪case_insensitive=True)
```

[13]:
```
# get accuracy per category
summary = accuracy_by_category(detailed_accuracy)
print('Base Accuracy: Correct {:,.0f} | Wrong {:,.0f} | Avg {:,.2%}\n'.
 ↪format(*summary))
```

	category	correct	incorrect	average
0	capital-common-countries	322	98	0.77
10	gram6-nationality-adjective	732	324	0.69
1	capital-world	678	512	0.57
7	gram3-comparative	307	563	0.35
14	total	2941	5639	0.34
4	family	37	73	0.34
8	gram4-superlative	87	185	0.32
11	gram7-past-tense	332	790	0.30
3	currency	24	104	0.19
9	gram5-present-participle	108	492	0.18
12	gram8-plural	54	252	0.18
13	gram9-plural-verbs	77	429	0.15
2	city-in-state	153	1203	0.11
6	gram2-opposite	10	172	0.05
5	gram1-adjective-to-adverb	20	442	0.04

Base Accuracy: Correct 2,941 | Wrong 5,639 | Avg 34.28%

[14]:
```
most_sim = model.wv.most_similar(positive=['woman', 'king'], negative=['man'],␣
 ↪topn=20)
pd.DataFrame(most_sim, columns=['token', 'similarity'])
```

[14]:

	token	similarity
0	meghan	0.43
1	kings	0.42
2	princess	0.40
3	winery	0.40
4	keller	0.40
5	angela	0.39
6	curry	0.39
7	daly	0.39
8	roman	0.39
9	duke	0.38
10	barber	0.38
11	famed	0.38
12	uncle	0.38
13	patron	0.38
14	moss	0.38
15	emma	0.37
16	jake	0.37
17	kristen	0.37

```
18     malik        0.37
19       von        0.37
```

[15]:
```
counter = Counter(sentence_path.read_text().split())
```

[16]:
```
most_common = pd.DataFrame(counter.most_common(), columns=['token', 'count'])
most_common = most_common[most_common['count']> MIN_FREQ]
most_common['p'] = np.log(most_common['count'])/np.log(most_common['count']).
↪sum()
```

[17]:
```
similars = pd.DataFrame()
for token in np.random.choice(most_common.token, size=10, p=most_common.p):
    similars[token] = [s[0] for s in model.wv.most_similar(token)]
similars.T
```

[17]:
```
                                           0                        1  \
shall_sale                jurisdiction_offer  solicitation_sale_unlawful
identiv                               apptio                        ooma
liquidate                           illiquid                 liquidating
selectively          tumor_microenvironment                      inhibit
voting                                 votes                       voted
payment                             payments                        paid
nationality                           kuciak                     sarkozy
arpu                                postpaid                       rgus
beijing_monitoring_desk     editing_kim_coghill                 sam_holmes

                                                   2  \
shall_sale              prior_registration_qualification
identiv                                 juniper_networks
liquidate                                        dispose
selectively                                    receptors
voting                                              vote
payment                                     installments
nationality                               arrest_warrant
arpu                                          subscriber
beijing_monitoring_desk          editing_jacqueline_wong

                                             3  \
shall_sale              sell_solicitation_offer
identiv                          asure_software
liquidate                             defaulted
selectively                        cancer_cells
voting                               votes_cast
payment                             installment
nationality                               raped
arpu                            subscriber_base
beijing_monitoring_desk            himani_sarkar
```

```
                                                4  \
shall_sale                      offer_solicitation
identiv                          palo_alto_networks
liquidate        prior_registration_qualification
selectively                                t_cells
voting                                     ballots
payment                                  repayment
nationality                                 deport
arpu                                         churn
beijing_monitoring_desk                kim_coghill

                                                5                6  \
shall_sale                  shall_constitute_offer        buy_shall
identiv                               interdigital   cheetah_mobile
liquidate                                  offload           divest
selectively                             inhibition          antigen
voting                            polling_stations         election
payment                         plus_accrued_unpaid           alipay
nationality                        attempted_murder      deportations
arpu                                   subscribers              acv
beijing_monitoring_desk  editing_muralikumar_anantharaman  shri_navaratnam

                                               7                      8  \
shall_sale               solicitation_offer_buy  does_constitute_offer
identiv                              servicenow                orbcomm
liquidate                           bondholders              insolvent
selectively                          allogeneic              molecules
voting                          polling_station                 ballot
payment                           accrued_unpaid                    pay
nationality                              minors               forcibly
arpu                                  telephony                    ocf
beijing_monitoring_desk     christopher_cushing       shanghai_newsroom

                                                9
shall_sale               offer_sell_solicitation
identiv                                guidewire
liquidate                     jurisdiction_offer
selectively                 monoclonal_antibodies
voting                                  electing
payment                                   accrue
nationality                            immigrant
arpu                                         arr
beijing_monitoring_desk        editing_sam_holmes
```

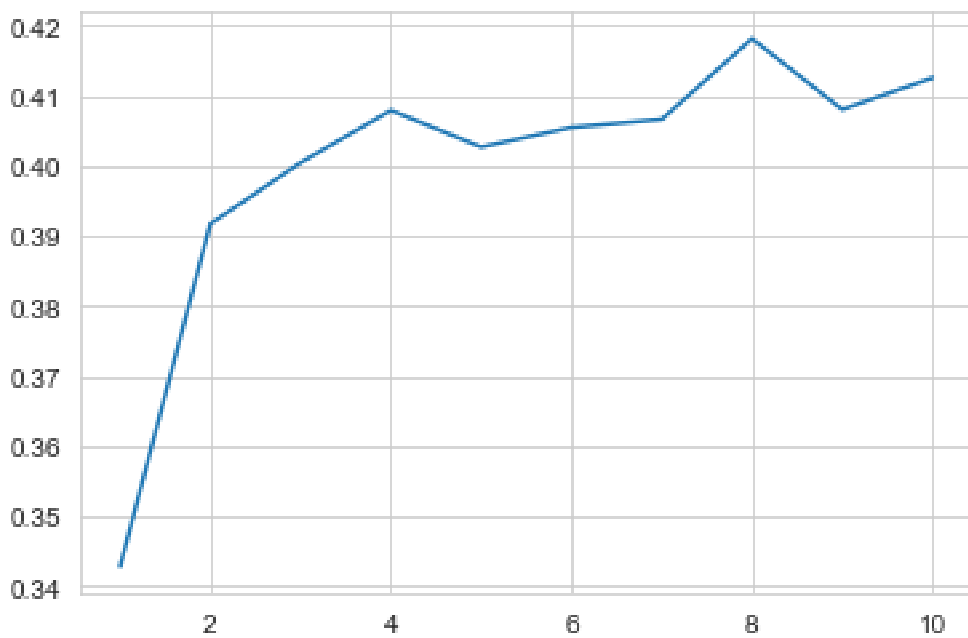
1.6 Continue Training

[18]:
```
accuracies = [summary]
best_accuracy = summary[-1]
for i in range(1, 10):
    start = time()
    model.train(sentences, epochs=1, total_examples=model.corpus_count)
    detailed_accuracy = model.wv.accuracy(analogy_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(str(gensim_path / f'word2vec_{i:02}.model'))
        model.wv.save(str(gensim_path / f'word_vectors_{i:02}.bin'))
        best_accuracy = accuracies[-1][-1]
    (pd.DataFrame(accuracies,
                  columns=['correct', 'wrong', 'average'])
     .to_csv(gensim_path / 'accuracies.csv', index=False))
model.wv.save(str(gensim_path / 'word_vectors_final.bin'))
```

```
01 | Duration: 00:02:48 | Accuracy: 39.17%
02 | Duration: 00:02:46 | Accuracy: 40.05%
03 | Duration: 00:02:54 | Accuracy: 40.79%
04 | Duration: 00:02:58 | Accuracy: 40.27%
05 | Duration: 00:02:39 | Accuracy: 40.55%
06 | Duration: 00:02:41 | Accuracy: 40.66%
07 | Duration: 00:02:40 | Accuracy: 41.82%
08 | Duration: 00:02:44 | Accuracy: 40.80%
09 | Duration: 00:02:34 | Accuracy: 41.26%
```

1.7 Evaluate Best Model

[19]:
```
pd.DataFrame(accuracies, columns=['correct', 'wrong', 'average'],␣
 ↪index=list(range(1, len(accuracies) + 1))).average.plot();
```



[20]:
```
best_model = Word2Vec.load((gensim_path / 'word2vec_06.model').as_posix())
```

[21]:
```
# gensim computes accuracy based on source text files
detailed_accuracy = best_model.wv.accuracy(analogy_path.as_posix(),␣
↪case_insensitive=True)
```

[22]:
```
# get accuracy per category
summary = accuracy_by_category(detailed_accuracy)
print('Base Accuracy: Correct {:,.0f} | Wrong {:,.0f} | Avg {:,.2%}\n'.
↪format(*summary))
```

	category	correct	incorrect	average
0	capital-common-countries	356	64	0.85
1	capital-world	911	279	0.77
10	gram6-nationality-adjective	767	289	0.73
14	total	3582	4998	0.42
7	gram3-comparative	344	526	0.40
4	family	41	69	0.37
11	gram7-past-tense	351	771	0.31
12	gram8-plural	86	220	0.28
2	city-in-state	372	984	0.27
8	gram4-superlative	71	201	0.26
3	currency	32	96	0.25
9	gram5-present-participle	125	475	0.21
13	gram9-plural-verbs	72	434	0.14
5	gram1-adjective-to-adverb	40	422	0.09

```
6                 gram2-opposite       14        168     0.08
Base Accuracy: Correct 3,582 | Wrong 4,998 | Avg 41.75%
```

[23]:
```
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')
```

[24]:
```
most_sim = best_model.wv.most_similar(positive=['woman', 'king'],␣
 ↪negative=['man'], topn=20)
pd.DataFrame(most_sim, columns=['token', 'similarity'])
```

[24]:

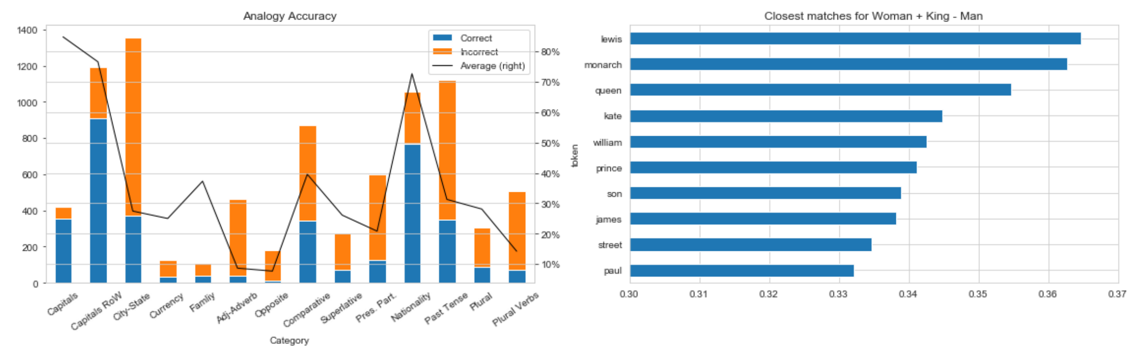
	token	similarity
0	lewis	0.36
1	monarch	0.36
2	queen	0.35
3	kate	0.34
4	william	0.34
5	prince	0.34
6	son	0.34
7	james	0.34
8	street	0.33
9	paul	0.33
10	von	0.33
11	hill	0.32
12	george	0.32
13	martin	0.32
14	murray	0.32
15	tyler	0.32
16	regional	0.32
17	ext	0.31
18	charlie	0.31
19	clark	0.31

[25]:
```
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();
```



[26]:
```
counter = Counter(sentence_path.read_text().split())
```

[27]:
```
most_common = pd.DataFrame(counter.most_common(), columns=['token', 'count'])
most_common = most_common[most_common['count']> MIN_FREQ]
most_common['p'] = np.log(most_common['count'])/np.log(most_common['count']).
↪sum()
```

[28]:
```
similars = pd.DataFrame()
for token in np.random.choice(most_common.token, size=10, p=most_common.p):
    similars[token] = [s[0] for s in best_model.wv.most_similar(token)]
similars.T
```

[28]:
```
                                         0          1              2  \
risks_uncertainties           cause_actual    factors  uncertainties
wheeler_real_estate                  trust  gladstone      infrareit
culinary                              chef     dining     restaurant
expensive                          cheaper     costly          cheap
fy                     compensation_mln_vs        mln         versus
modern                        architecture         ai   contemporary
advocacy                         nonprofit  education         groups
worsening                           severe       dire       symptoms
hawkish                             dovish        fed     rate_hikes
an                                     the       very             it
```

```
                                                            3                  4  \
risks_uncertainties  known_unknown_risks_uncertainties  differ_materially
wheeler_real_estate                               reit            redwood
culinary                                          wine             coffee
expensive                                     cheapest          difficult
fy                                               qtrly            sees_fy
modern                                   functionality         innovative
advocacy                                      practice           advocate
worsening                                     diarrhea              acute
hawkish                                         yellen    federal_reserve
an                                                   a             called

                                                      5              6  \
risks_uncertainties  differ_materially_expressed_implied          risks
wheeler_real_estate                          gi_partners        matthew
culinary                                        beverage    attractions
expensive                                           more           like
fy                                                   ceo          total
modern                                           elegant   cutting_edge
advocacy                                    philanthropy      advocates
worsening                                         suffer         plight
hawkish                                  monetary_policy         powell
an                                               appears        earlier

                               7            8             9
risks_uncertainties   statements  assumptions       involve
wheeler_real_estate        essex          agf        riocan
culinary                  garden       guests   educational
expensive              consuming       easier   complicated
fy                           cfo      quarter  compensation
modern               innovations      mission      platform
advocacy               awareness    mentoring          lgbt
worsening                 influx   escalating         fever
hawkish                     hike       policy          tone
an                           was         this      scenario
```

[29]:
```
similars.T.iloc[:5, :5].to_csv('figures/most_similar.csv')
```

1.8 Resources

- Distributed representations of words and phrases and their compositionality
- Efficient estimation of word representations in vector space
- Sebastian Ruder's Blog