word2vec

September 29, 2021

0.1 Imports & Settings

[1]:
```
from pathlib import Path
from time import time
import warnings
from collections import Counter
import logging
from ast import literal_eval as make_tuple
import numpy as np
import pandas as pd

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

[2]:
```
pd.set_option('display.expand_frame_repr', False)
warnings.filterwarnings('ignore')
np.random.seed(42)
```

[3]:
```
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)
```

0.1.1 Logging Setup

[4]:
```
logging.basicConfig(
        filename='logs/word2vec.log',
        level=logging.DEBUG,
        format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
        datefmt='%H:%M:%S')
```

0.2 word2vec

[6]:
```
analogies_path = Path().cwd().parent / 'data' / 'analogies' / 'analogies-en.txt'
```

0.2.1 Set up Sentence Generator

[8]:
```
NGRAMS = 2
```

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

[9]:
```
sentence_path = Path('data', 'ngrams', f'ngrams_{NGRAMS}.txt')
sentences = LineSentence(sentence_path)
```

0.2.2 Train word2vec Model

The gensim.models.word2vec class implements the skipgram and CBOW architectures introduced above. The notebook word2vec contains additional implementation detail.

[10]:
```
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=3,      # Max distance betw. current and predicted word
                 min_count=50,  # Ignore words with lower frequency
                 negative=10,    # noise word count for negative sampling
                 workers=8,     # no threads
                 iter=1,        # no epochs = iterations over corpus
                 alpha=0.025,   # initial learning rate
                 min_alpha=0.0001 # final learning rate
                )
print('Duration:', format_time(time() - start))
```

```
Duration: 00:10:47
```

0.2.3 Persist model & vectors

[11]:
```
model.save('models/baseline/word2vec.model')
model.wv.save('models/baseline/word_vectors.bin')
```

0.2.4 Load model and vectors

[40]:
```
model = Word2Vec.load('models/archive/word2vec.model')
```

[8]:
```
wv = KeyedVectors.load('models/baseline/word_vectors.bin')
```

0.2.5 Get vocabulary

[12]:
```
vocab = []
for k, _ in model.wv.vocab.items():
    v_ = model.wv.vocab[k]
    vocab.append([k, v_.index, v_.count])
```

[13]:
```
vocab = (pd.DataFrame(vocab,
                      columns=['token', 'idx', 'count'])
         .sort_values('count', ascending=False))
```

[14]:
```
vocab.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50491 entries, 104 to 46372
Data columns (total 3 columns):
token    50491 non-null object
idx      50491 non-null int64
count    50491 non-null int64
dtypes: int64(2), object(1)
memory usage: 1.5+ MB
```

[15]:
```
vocab.head(10)
```

[15]:

	token	idx	count
104	million	0	2340243
0	business	1	1700662
66	december	2	1513533
627	company	3	1490752
477	products	4	1368711
1071	net	5	1253343
145	market	6	1149048
380	including	7	1110482
381	sales	8	1098312
60	costs	9	1020383

[16]:
```
vocab['count'].describe(percentiles=np.arange(.1, 1, .1)).astype(int)
```

[16]:
```
count     50491
mean       5110
std       37525
min          50
10%          61
20%          78
30.0%       102
40%         137
50%         195
60%         300
```

```
70%          522
80%         1164
90%         4578
max      2340243
Name: count, dtype: int64
```

0.2.6 Evaluate Analogies

[110]:
```python
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))))
```

[52]:
```python
def total_accuracy(w2v):
    df = eval_analogies(w2v)
    return df.loc[df.category == 'total', ['correct', 'incorrect', 'average']].
↪squeeze().tolist()
```

[42]:
```python
accuracy = eval_analogies(model)
accuracy
```

[42]:

	category	correct	incorrect	average
0	capital-common-countries	2	4	0.333333
1	capital-world	0	0	0.000000
2	city-in-state	140	390	0.264151
3	currency	2	26	0.071429
4	family	0	0	0.000000
5	gram1-adjective-to-adverb	48	134	0.263736
6	gram2-opposite	23	67	0.255556
7	gram3-comparative	240	222	0.519481
8	gram4-superlative	19	53	0.263889
9	gram5-present-participle	90	182	0.330882
10	gram6-nationality-adjective	250	130	0.657895
11	gram7-past-tense	94	286	0.247368
12	gram8-plural	87	69	0.557692
13	gram9-plural-verbs	72	138	0.342857
14	total	1067	1701	0.385477

0.2.7 Validate Vector Arithmetic

[105]:
```
pd.read_csv(ANALOGIES_PATH, header=None, sep=' ').head()
```

[105]:

	0	1	2	3
0	:	capital-common-countries	NaN	NaN
1	athens	greece	baghdad	iraq
2	athens	greece	bangkok	thailand
3	athens	greece	beijing	china
4	athens	greece	berlin	germany

[112]:
```
sims=model.wv.most_similar(positive=['iphone'],
                           restrict_vocab=15000)
print(pd.DataFrame(sims, columns=['term', 'similarity']))
```

```
                  term  similarity
0              android    0.600454
1           smartphone    0.581685
2                  app    0.559129
3          smartphones    0.533848
4  smartphones_tablets    0.526129
5             handsets    0.514813
6         smart_phones    0.512868
7                apple    0.507795
8                 apps    0.505517
9              handset    0.491526
```

[113]:
```
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    0.606630
1         germany    0.585644
2     netherlands    0.578868
3           italy    0.547168
4           india    0.545213
5           spain    0.539029
6       singapore    0.535106
7       australia    0.525464
8         belgium    0.523677
9          sweden    0.510462
```

0.2.8 Check similarity for random words

[41]:
```
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
```

```
/home/stefan/.pyenv/versions/at-3.6/lib/python3.6/site-
packages/gensim/matutils.py:737: FutureWarning: Conversion of the second
argument of issubdtype from `int` to `np.signedinteger` is deprecated. In
future, it will be treated as `np.int64 == np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

[41]:
```
           staff    enables                                times
fees                         sources
0      personnel     allows                                twice
fee                          source
1           team   enabling               standpoint_advantageous
professional_fees                  primary_source
2          teams      helps                    vimovo_orange_book
checkcard                        sourced
3  professionals     enable                           millisecond
commissions                 readily_available
4        staffed   allowing                             saturdays
atm_debit_card              internally_generated
5         hiring    enabled  assets_liabilities_react_differently
gds_reservation_booking                       generated
6    consultants      allow                          twice_weekly
interchange_fees_swipe  biological_contaminants_pollen
7          hired  leverages                                   day
noticing      repair_reconstruct_damaged
8      engineers       lets                              weekdays
nonsufficient                    alternative
9    salespeople       easy                                   uvb
bno_usci_cper_usag                            znse
```

0.3 Continue Training

[ ]:
```
accuracies = (eval_analogies(model)
              .set_index('category')
              .average
              .to_frame('baseline'))
```

[76]:
```
for i in range(1, 11):
    start = time()
    model.train(sentences, epochs=1, total_examples=model.corpus_count)
    accuracy = eval_analogies(model).set_index('category').average
    accuracies = accuracies.join(accuracy.to_frame(f'{n}'))
    print(f'{i} | Duration: {format_time(time() - start)} | Accuracy: {accuracy.
 ↪total:.2%}')
    model.save(f'word2vec/models/word2vec_{i}.model')
```

```
/home/stefan/.pyenv/versions/at-3.6/lib/python3.6/site-
packages/ipykernel_launcher.py:5: DeprecationWarning: Call to deprecated
`accuracy` (Method will be removed in 4.0.0, use self.evaluate_word_analogies()
instead).
  """
/home/stefan/.pyenv/versions/at-3.6/lib/python3.6/site-
packages/gensim/matutils.py:737: FutureWarning: Conversion of the second
argument of issubdtype from `int` to `np.signedinteger` is deprecated. In
future, it will be treated as `np.int64 == np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
1 | Duration: 464.0 | Accuracy: 28.93%
2 | Duration: 457.8 | Accuracy: 28.83%
3 | Duration: 459.2 | Accuracy: 28.97%
4 | Duration: 456.9 | Accuracy: 28.60%
5 | Duration: 457.4 | Accuracy: 29.69%
6 | Duration: 456.8 | Accuracy: 29.40%
7 | Duration: 457.7 | Accuracy: 29.91%
8 | Duration: 456.4 | Accuracy: 29.61%
9 | Duration: 456.1 | Accuracy: 29.37%
10 | Duration: 454.6 | Accuracy: 29.17%
```

[ ]:
```
model.wv.save('word_vectors_final.bin')
```