Word2vec vector analysis

March 7, 2021

1 SI630 Homework 2: Word2vec Vector Analysis

Important Note: Start this notebook only after you've gotten your word2vec model up and running!

Many NLP packages support working with word embeddings. In this notebook you can work through the various problems assigned in Task 3. We've provided the basic functionality for loading word vectors using Gensim, a good library for learning and using word vectors, and for working with the vectors.

One of the fun parts of word vectors is getting a sense of what they learned. Feel free to explore the vectors here!

2 Problem 9

[1]:
```
# username: hexinz
from gensim.models import KeyedVectors
from gensim.test.utils import datapath
```

[2]:
```
word_vectors = KeyedVectors.load_word2vec_format('output.wv', binary=False)
```

3 Problem 10

3.0.1 Common words

[3]:
```
# word_vectors['the']
word_vectors.similar_by_word("and")
```

[3]:
```
[('Wernerian', 0.31261146068573),
 ('Gynaecologists', 0.2848040461540222),
 ('REG1', 0.27965983748435974),
 ('Mordialloc', 0.27877068519592285),
 ('Juried', 0.2782945930957794),
 ('Mardhiah', 0.27755188941955566),
 ('Phitthaya', 0.2770390808582306),
 ('Frauenbund', 0.27112919092178345),
 ('Cremorne', 0.2681736946105957),
 ('Chríost', 0.26729196310043335)]
```

```
[4]: word_vectors.similar_by_word("son")
```

```
[4]: [('daughter', 0.9444119334220886),
      ('grandfather', 0.9349886775016785),
      ('eldest', 0.9347566962242126),
      ('grandson', 0.9312842488288879),
      ('child', 0.9305014610290527),
      ('heir', 0.9261252880096436),
      ('sons', 0.9237461090087891),
      ('younger', 0.9226715564727783),
      ('brothers', 0.9212406873703003),
      ('elder', 0.9160992503166199)]
```

```
[5]: word_vectors.similar_by_word("born")
```

```
[5]: [('Born', 0.9310526847839355),
      ('raised', 0.9041904807090759),
      ('2019', 0.8900091052055359),
      ('neighborhood', 0.8882691860198975),
      ('grew', 0.8880464434623718),
      ('Studios', 0.8847413659095764),
      ('wealthy', 0.8807938694953918),
      ('Education', 0.8804402351379395),
      ('Long', 0.8804317712783813),
      ('August', 0.8778650760650635)]
```

```
[6]: word_vectors.similar_by_word("top")
```

```
[6]: [('goalscorer', 0.9492651224136353),
      ('Top', 0.9429421424865723),
      ('scorer', 0.9418091177940369),
      ('scorers', 0.9362465143203735),
      ('highest', 0.9314957857131958),
      ('rotation', 0.9287024140357971),
      ('spot', 0.9262141585350037),
      ('grossing', 0.9254440069198608),
      ('ranking', 0.9253345727920532),
      ('relegation', 0.9249082803726196)]
```

3.0.2 Occasional

```
[7]: word_vectors.similar_by_word("eventually")
```

```
[7]: [('successfully', 0.9462027549743652),
      ('finally', 0.9460392594337463),
      ('however', 0.9392870664596558),
      ('McHale', 0.9370847940444946),
      ('therefore', 0.9323566555976868),
      ('profession', 0.9318313598632812),
      ('Serena', 0.9316397905349731),
      ('thus', 0.9305494427680969),
      ('tag', 0.930137038230896),
      ('formation', 0.9297938346862793)]
```

```
[8]: word_vectors.similar_by_word("amazing")
```

```
[8]: [('exciting', 0.9361180067062378),
      ('terrible', 0.9331440329551697),
      ('fantastic', 0.9317662715911865),
      ('mean', 0.9301401376724243),
      ('replied', 0.9280708432197571),
      ('happens', 0.9274816513061523),
      ('What', 0.9268141388893127),
      ('Melody', 0.9259214401245117),
      ('remarkable', 0.9257221221923828),
      ('unlikely', 0.9256329536437988)]
```

```
[9]: word_vectors.similar_by_word("China")
```

```
[9]: [('Korea', 0.9443014860153198),
      ('Taiwan', 0.932518482208252),
      ('Beijing', 0.92923903465271),
      ('Kenya', 0.9283241033554077),
      ('Republic', 0.9253071546554565),
      ('Turkey', 0.9252933859825134),
      ('Hong', 0.9238981008529663),
      ('Malaysia', 0.9235707521438599),
      ('documentary', 0.921799898147583),
      ('Italy', 0.9209052324295044)]
```

```
[10]: word_vectors.similar_by_word("SARS")
```

```
[10]: [('nervous', 0.9037822484970093),
       ('privacy', 0.8916728496551514),
       ('detailing', 0.8879326581954956),
       ('sentencing', 0.886925995349884),
       ('opium', 0.8844797611236572),
       ('disasters', 0.8843635320663452),
       ('protracted', 0.8824070692062378),
       ('blindness', 0.8817066550254822),
       ('intense', 0.881163477897644),
       ('interrogation', 0.880934476852417)]
```

3.0.3 Rare

```
[11]: word_vectors.similar_by_word("captives")
```

```
[11]: [('babies', 0.9478209018707275),
       ('journalists', 0.9245781898498535),
       ('deaths', 0.920008659362793),
       ('attendants', 0.9135136008262634),
       ('targets', 0.9130617380142212),
       ('homosexuals', 0.9128782749176025),
       ('inmates', 0.9089660048484802),
       ('unconfirmed', 0.9085569381713867),
       ('creditors', 0.9084674715995789),
       ('rumored', 0.9084410667419434)]
```

```
[12]: word_vectors.similar_by_word("stupid")
```

```
[12]: [('everyone', 0.9415018558502197),
       ('pretty', 0.939758837223053),
       ('wisdom', 0.93647301197052),
       ('knowing', 0.935626745223999),
       ('contradiction', 0.9352973103523254),
       ('laughing', 0.9343644380569458),
       ('joke', 0.9340013444680786),
       ('know', 0.932907223701477),
       ('honest', 0.9300961494445801),
       ('skeptical', 0.9278777241706848)]
```

- Firstly, we select four common words: ‘and’, ‘son’, ‘born’, and ‘top’. ‘and’ and ‘son’ appear more than 100,000 times; ‘born’ and ‘top’ appear more than 10,000 times. For ‘and’, the predicted similar words are rare words; for ‘son’, ‘born’ and ‘top’, the predicted similar words are reasonable: more verbs and words related to time appear in the similar words of ‘born’ and more words related to identity appear in the similar words of ‘son’.
- Secondly, we select four occasional words: ‘eventually’, ‘amazing’, ‘China’ and ‘SARS’. They appear more than 100 and less than 10,000 times. Similar words are successfully predicted.
- Finally, we select two rare words: ‘captives’ and , which appear less than 100 times. Similar words are partially successfully predicted.

4 Problem 11

```
[13]: def get_analogy(a, b, c):
          return word_vectors.most_similar(positive=[b, c], negative=[a])[0][0]
```

```
[14]: get_analogy('son', 'daughter', 'elder')
```

```
[14]: 'sister'
```

[15]:
```
get_analogy('student', 'teacher', 'homework')
```

[15]:
```
'exhausting'
```

[16]:
```
get_analogy('young', 'old', 'younger')
```

[16]:
```
'older'
```

[17]:
```
get_analogy('novel', 'movie', 'writer')
```

[17]:
```
'producer'
```

[18]:
```
get_analogy('short', 'long', 'young')
```

[18]:
```
'girl'
```

We can see that, among five selected word analies, four of them have good performance. - daughter + elder - son = sister, - student - homework = teacher - exhausting - young - younger = old - older - novel - writer = movie - producer

However, we could not get 'old' from 'long + young - short', indicating that the word2vec vector works better on synonymity than antonymy.

5 Problem 12

[19]:
```
word_vectors_syn = KeyedVectors.load_word2vec_format('output_update.wv',␣
 ↪binary=False)
```

[20]:
```
import pandas as pd
import numpy as np
data = pd.read_csv('word_pairs_to_estimate_similarity.test.csv')
id = data['pair_id'].values
word1 = data['word1'].values
word2 = data['word2'].values
sim = []
for i in id:
    sim.append(word_vectors.similarity(word1[i], word2[i]))
result = {'id': id, 'sim': sim}
result = pd.DataFrame(result)
header = ['pair_id', 'similarity']
result.to_csv('result.csv', header=header, index=False)
```

[21]:
```
import pandas as pd
import numpy as np
data = pd.read_csv('word_pairs_to_estimate_similarity.test.csv')
id = data['pair_id'].values
word1 = data['word1'].values
```

```
word2 = data['word2'].values
sim = []
for i in id:
    sim.append(word_vectors_syn.similarity(word1[i], word2[i]))
result = {'id': id, 'sim': sim}
result = pd.DataFrame(result)
header = ['pair_id', 'similarity']
result.to_csv('result2.csv', header=header, index=False)
```

6 Problem 15

[23]:
```
word_vectors_syn.similar_by_word("top")
```

[23]:
```
[('summit', 0.9020704627037048),
 ('meridian', 0.8762383460998535),
 ('pinnacle', 0.8365438580513),
 ('superlative', 0.8226457834243774),
 ('elevation', 0.8080981969833374),
 ('height', 0.8065374493598938),
 ('peak', 0.8059883713722229),
 ('comebacks', 0.7384368181228638),
 ('Influencers', 0.7248468995094299),
 ('Sandanme', 0.7173957228660583)]
```

[24]:
```
word_vectors_syn.similar_by_word("eventually")
```

[24]:
```
[('ultimately', 0.7511016130447388),
 ('Vyronas', 0.6867636442184448),
 ('Garachiné', 0.6825960278511047),
 ('Afterward', 0.6714228391647339),
 ('Oaș', 0.6677548289299011),
 ('initially', 0.6623428463935852),
 ('Barlog', 0.6619102954864502),
 ('luxuriously', 0.6592634916305542),
 ('physic', 0.6558774709701538),
 ('Duerrstein', 0.6493158936500549)]
```

[27]:
```
word_vectors_syn.similar_by_word("amazing")
```

[27]:
```
[('awesome', 0.8943012952804565),
 ('awful', 0.8396655321121216),
 ('unbelievable', 0.8320059776306152),
 ('wondrous', 0.820950984954834),
 ('unbelievably', 0.8129263520240784),
 ('pleasurable', 0.8038662672042847),
 ('marvelous', 0.8005156517028809),
 ('cheeky', 0.7975283861160278),
 ('incredible', 0.7962194085121155),
 ('crap', 0.7933977246284485)]
```

```
[34]: word_vectors_syn.similar_by_word("give")
```

```
[34]: [('fetch', 0.6756247282028198),
 ('convince', 0.6570402383804321),
 ('impart', 0.6450479030609131),
 ('modernize', 0.6416994333267212),
 ('Vassouras', 0.6409450173377991),
 ('avail', 0.6392044425010681),
 ('maintain', 0.6346015930175781),
 ('encourage', 0.6322163343429565),
 ('MercyMe', 0.630827009677887),
 ('cater', 0.6286659240722656)]
```

```
[35]: word_vectors.similar_by_word("give")
```

```
[35]: [('giving', 0.9432541131973267),
 ('access', 0.9427363276481628),
 ('bring', 0.9392701387405396),
 ('drop', 0.9363373517990112),
 ('offer', 0.9341126680374146),
 ('favour', 0.9329581260681152),
 ('keep', 0.9326536655426025),
 ('see', 0.9322749376296997),
 ('hear', 0.9315024018287659),
 ('gives', 0.9314975738525391)]
```

```
[38]: word_vectors_syn.similar_by_word("constitution")
```

```
[38]: [('formation', 0.7976463437080383),
 ('establishment', 0.7773997783660889),
 ('realignment', 0.7506915330886841),
 ('integral', 0.7382983565330505),
 ('reorganisation', 0.7266575694084167),
 ('constitutional', 0.7212973833084106),
 ('CPNC', 0.7196764945983887),
 ('feinting', 0.7195417881011963),
 ('stakeholder', 0.7175673246383667),
 ('rationalization', 0.7112507224082947)]
```

```
[40]: word_vectors.similar_by_word("constitution")
```

```
[40]: [('laws', 0.9439664483070374),
 ('evidence', 0.9363945722579956),
 ('corruption', 0.9318532347679138),
 ('conclusions', 0.9308825731277466),
 ('constitutional', 0.9297701120376587),
 ('yet', 0.9284285306930542),
 ('changes', 0.9282674193382263),
 ('existence', 0.928173840045929),
 ('values', 0.9261394143104553),
 ('phase', 0.9260390400886536)]
```

We select five words: son, top, eventually, SARS and stupid We can see that for adjectives and adverbs, the synonym-aware model has better performance since they contain more synonyms. For example: - the synonym-aware model for 'eventually' has 'ultimately'. On the contrary, the non-synonym-aware model for 'eventually' only has 'successfully'. - the synonym-aware model for 'top' has 'peak', 'summit', 'elecation', 'superlative' and so on while the non-synonym-aware model for 'top' don't. - the synonym-aware model for 'amazing' has 'awesome', 'awful', 'unbelievable' and so on while the non-synonym-aware model for 'amazing' don't. - the synonym-aware model for 'give' has 'fetch', 'impart', 'MercyMe' and so on while the non-synonym-aware model for 'give' has 'giving', 'offer', 'gives'. They have similar performance. - the synonym-aware model for 'constitution' has 'constitutional', 'formation', 'establishment' and so on while the non-synonym-aware model for 'constitution' don't. Therefore, from my perspective, the synonym-aware model does produce better vectors than the non-synonym-aware model.

[ ]: