IN4080_2022_exercise_7_solution

November 15, 2022

0.1 Getting started

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import gensim
```

[2]:
```
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',
                    level=logging.INFO)
```

[3]:
```
%%time

import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
```

```
2022-11-11 16:08:09,522 : INFO : loading projection weights from
C:\Users\jtl/gensim-data\word2vec-google-news-300\word2vec-google-news-300.gz
2022-11-11 16:09:03,611 : INFO : KeyedVectors lifecycle event {'msg': 'loaded
(3000000, 300) matrix of type float32 from C:\\Users\\jtl/gensim-data\\word2vec-
google-news-300\\word2vec-google-news-300.gz', 'binary': True, 'encoding':
'utf8', 'datetime': '2022-11-11T16:09:03.611537', 'gensim': '4.2.0', 'python':
'3.10.6 | packaged by conda-forge | (main, Oct 24 2022, 16:02:16) [MSC v.1916 64
bit (AMD64)]', 'platform': 'Windows-10-10.0.19042-SP0', 'event':
'load_word2vec_format'}

CPU times: total: 47.7 s
Wall time: 54.3 s
```

0.2 Exercise 1 Basics

a) How many different words are there in the model? With so many words, how come that the 'cameroon' example fails?

[4]:
```
print("The google-news-300 contains {} tokens".format(len(wv.index_to_key)))
```

```
The google-news-300 contains 3000000 tokens
```

[5]:
```
try:
    vec_cameroon = wv['cameroon']
except KeyError:
    print("The word 'cameroon' does not appear in this model")
```

```
The word 'cameroon' does not appear in this model
```

Answer The model contains the word 'Cameroon'. Be aware that some trained models have case folded all the words, while other models keep the casing from the training corpus.

b) Implement a function for calculating the norm (the length) of an (embedding) vector, and a function for calculating the cosine between two vectors.

Solution using numpy

[6]:
```
def norm_1(v):
    return (v @ v)**0.5
```

[7]:
```
def cos_1(v1, v2):
    return (v1 @ v2)/(norm_1(v1) * norm_1(v2))
```

Alternative solution using python

[8]:
```
def norm_2(v):
    s = sum([f * f for f in v])
    return s**0.5
```

[9]:
```
def cos_2(v1, v2):
    s = [f1 * f2 for f1, f2 in zip(v1, v2)]
    return sum(s)/(norm_2(v1) * norm_2(v2))
```

c) Calculate the cosine between the vectors for 'king' and 'queen' and check you get the same as by `<model>.similarity('king', 'queen')`

[10]:
```
cos_1(wv['king'], wv['queen'])
```

[10]:
```
0.6510956814003394
```

[11]:
```
cos_2(wv['king'], wv['queen'])
```

[11]:
```
0.6510956835386662
```

[12]:
```
wv.similarity('king', 'queen')
```

[12]:
```
0.6510957
```

0.3 Built-in functions

Try a few analogy tests like

- " king is to man as queen is to ..."
- " king is to queen as man is to ..."
- "cat is to kitten as dog is to ..."

[13]:
```
wv.most_similar(positive=['queen', 'man'], negative=['king'])
```

[13]:
```
[('woman', 0.7609436511993408),
 ('girl', 0.6139993667602539),
 ('teenage_girl', 0.6040961742401123),
 ('teenager', 0.5825759172439575),
 ('lady', 0.5752554535865784),
 ('boy', 0.5077577233314514),
 ('policewoman', 0.5066847801208496),
 ('schoolgirl', 0.5052096247673035),
 ('blonde', 0.4869619309902191),
 ('person', 0.4863754510879516)]
```

This covers both the first two examples.

[14]:
```
wv.most_similar(positive=['kitten', 'dog'], negative=['cat'])
```

[14]:
```
[('puppy', 0.7699725031852722),
 ('pup', 0.6861710548400879),
 ('pit_bull', 0.6776558756828308),
 ('dogs', 0.6770986318588257),
 ('Rottweiler', 0.6646621823310852),
 ('pit_bull_mix', 0.6585750579833984),
 ('Pomeranian', 0.6553813815116882),
 ('Labrador_retriever_mix', 0.6510828137397766),
 ('German_shepherd', 0.6490000486373901),
 ('puppies', 0.6450918316841125)]
```

Sweden is to Stockholm as Norway it to ... expcted: Oslo

[15]:
```
wv.most_similar(positive=['Stockholm', 'Norway'], negative=['Sweden'], topn=5)
```

[15]:
```
[('Oslo', 0.8023768067359924),
 ('Oslo_Norway', 0.6939629912376404),
 ('Norwegian', 0.660550594329834),
 ('Stavanger', 0.6027042865753174),
 ('Helsinki', 0.593083918094635)]
```

This also works as it should. Messi is to Argentina as Neymar is to ... expected: Brazil

[16]:
```
wv.most_similar(positive=['Argentina', 'Neymar'], negative=['Messi'], topn=5)
```

[16]:
```
[('Brazil', 0.6224626302719116),
 ('Uruguay', 0.5814741849899292),
 ('Chile', 0.5608118176460266),
```

```
 ('Ecuador', 0.5597018003463745),
 ('Argentine', 0.555329442024231)]
```

[17]:
```
wv.most_similar(positive=['Argentina', 'Neymar'], negative=['Brazil'], topn=5)
```

[17]:
```
[('striker_Neymar', 0.6339360475540161),
 ('Real_Madrid', 0.612285315990448),
 ('Sergio_Aguero', 0.5889219641685486),
 ('Robinho', 0.587852418422699),
 ('Juan_Roman_Riquelme', 0.5848937034606934)]
```

Her we expected Messi, but didn't get it.

[18]:
```
wv.most_similar(positive=['kitten', 'cow'], negative=['cat'])
```

[18]:
```
[('cows', 0.6302610635757446),
 ('dairy_cow', 0.5432979464530945),
 ('calves', 0.5347446799278259),
 ('piglet', 0.5273206830024719),
 ('bovines', 0.5228687524795532),
 ('heifer', 0.5118857622146606),
 ('ewe', 0.510401725769043),
 ('goat', 0.5079760551452637),
 ('piglets', 0.506416380405426),
 ('pig', 0.501568615436554)]
```

We expected calf but didn't get it

[19]:
```
wv.most_similar(positive=['calf', 'cat'], negative=['cow'])
```

[19]:
```
[('pup', 0.5659834742546082),
 ('ankle', 0.5474446415901184),
 ('calf_muscle', 0.5419501662254333),
 ('cats', 0.5395172834396362),
 ('hamstring', 0.530087411403656),
 ('kitten', 0.5284544825553894),
 ('knee', 0.5234118700027466),
 ('thigh', 0.5140002965927124),
 ('kittens', 0.5106652975082397),
 ('Achilles', 0.5101013779640198)]
```

We expected kitten, but it first appeared as number 6. Observe in particular that word embeddings do not distinguish between homonyms; i.e. 'calf' as animal and as muscle.

b) To understand the method a little better, we can try to follow the recipe more directly. Try

```
a = wv['king'] + wv['woman'] - wv['man']
```

and calculate the cosine between a and the vectors for queen, woman, man, king. You may also calculate the wv.similar_by_vector(a) What does this show regarding how the most_similar works?

[20]:
```
a = wv['king'] + wv['woman'] - wv['man']
```

[21]:
```
for w in ['queen', 'woman', 'man', 'king']:
    print("Cosine distance between {} and a is {:7.4f}".format(w,␣
 ↪cos_1(a,wv[w])))
```

```
Cosine distance between queen and a is  0.7301
Cosine distance between woman and a is  0.3916
Cosine distance between man and a is  0.1216
Cosine distance between king and a is  0.8449
```

[22]:
```
wv.distances(a, ['queen', 'woman', 'man', 'king'])
```

[22]:
```
array([0.2699483 , 0.60836613, 0.87839365, 0.15506071], dtype=float32)
```

[23]:
```
wv.similar_by_vector(a)
```

[23]:
```
[('king', 0.8449392318725586),
 ('queen', 0.7300518155097961),
 ('monarch', 0.645466148853302),
 ('princess', 0.6156251430511475),
 ('crown_prince', 0.5818676948547363),
 ('prince', 0.5777117609977722),
 ('kings', 0.5613664388656616),
 ('sultan', 0.5376776456832886),
 ('Queen_Consort', 0.5344247221946716),
 ('queens', 0.5289887189865112)]
```

`most_similar([positive=...],...)`seems to exclude the postive words in the call from the list of most similar words.

Play around with `wv.doesnt_match`, e.g.

[24]:
```
print(wv.doesnt_match(['Norway', 'Denmark', 'Finland',
                       'Sweden', 'Spain', 'Stockholm']))
```

```
Spain
```

[25]:
```
print(wv.doesnt_match(['Norway', 'Denmark', 'Finland',
                       'Sweden', 'Stockholm']))
```

```
Stockholm
```

[26]:
```
print(wv.doesnt_match(['Marx', 'Mao', 'Lenin', 'Stalin', 'Kennedy']))
```

```
Kennedy
```

[27]:
```
print(wv.doesnt_match(['cat', 'lion', 'leopard', 'tiger', 'dog']))
```

```
lion
```

[28]:
```
print(wv.doesnt_match(['cat', 'lion', 'leopard', 'tiger']))
```

```
lion
```

In the last one we expected 'cat' (domestic), in the second to last, we expected 'dog' (not the cat family.

0.4 Exercise 3 Training a toy model

[29]:
```
import nltk
from nltk.corpus import brown
```

[30]:
```
%%time

import gensim.models

model = gensim.models.Word2Vec(sentences=brown.sents())
```

```
2022-11-11 16:10:12,603 : INFO : collecting all words and their counts
2022-11-11 16:10:12,603 : INFO : PROGRESS: at sentence #0, processed 0 words,
keeping 0 word types
2022-11-11 16:10:13,205 : INFO : PROGRESS: at sentence #10000, processed 219770
words, keeping 23488 word types
2022-11-11 16:10:13,706 : INFO : PROGRESS: at sentence #20000, processed 430477
words, keeping 34367 word types
2022-11-11 16:10:14,292 : INFO : PROGRESS: at sentence #30000, processed 669056
words, keeping 42365 word types
2022-11-11 16:10:14,878 : INFO : PROGRESS: at sentence #40000, processed 888291
words, keeping 49136 word types
2022-11-11 16:10:15,426 : INFO : PROGRESS: at sentence #50000, processed 1039920
words, keeping 53024 word types
2022-11-11 16:10:15,758 : INFO : collected 56057 word types from a corpus of
1161192 raw words and 57340 sentences
2022-11-11 16:10:15,758 : INFO : Creating a fresh vocabulary
2022-11-11 16:10:15,843 : INFO : Word2Vec lifecycle event {'msg':
'effective_min_count=5 retains 15173 unique words (27.07% of original 56057,
drops 40884)', 'datetime': '2022-11-11T16:10:15.843311', 'gensim': '4.2.0',
'python': '3.10.6 | packaged by conda-forge | (main, Oct 24 2022, 16:02:16) [MSC
v.1916 64 bit (AMD64)]', 'platform': 'Windows-10-10.0.19042-SP0', 'event':
'prepare_vocab'}
2022-11-11 16:10:15,843 : INFO : Word2Vec lifecycle event {'msg':
'effective_min_count=5 leaves 1095086 word corpus (94.31% of original 1161192,
drops 66106)', 'datetime': '2022-11-11T16:10:15.843311', 'gensim': '4.2.0',
'python': '3.10.6 | packaged by conda-forge | (main, Oct 24 2022, 16:02:16) [MSC
v.1916 64 bit (AMD64)]', 'platform': 'Windows-10-10.0.19042-SP0', 'event':
'prepare_vocab'}
```

```
2022-11-11 16:10:15,927 : INFO : deleting the raw counts dictionary of 56057
items
2022-11-11 16:10:15,927 : INFO : sample=0.001 downsamples 42 most-common words
2022-11-11 16:10:15,927 : INFO : Word2Vec lifecycle event {'msg': 'downsampling
leaves estimated 781596.5348479215 word corpus (71.4%% of prior 1095086)',
'datetime': '2022-11-11T16:10:15.927940', 'gensim': '4.2.0', 'python': '3.10.6 |
packaged by conda-forge | (main, Oct 24 2022, 16:02:16) [MSC v.1916 64 bit
(AMD64)]', 'platform': 'Windows-10-10.0.19042-SP0', 'event': 'prepare_vocab'}
2022-11-11 16:10:16,059 : INFO : estimated required memory for 15173 words and
100 dimensions: 19724900 bytes
2022-11-11 16:10:16,075 : INFO : resetting layer weights
2022-11-11 16:10:16,081 : INFO : Word2Vec lifecycle event {'update': False,
'trim_rule': 'None', 'datetime': '2022-11-11T16:10:16.081575', 'gensim':
'4.2.0', 'python': '3.10.6 | packaged by conda-forge | (main, Oct 24 2022,
16:02:16) [MSC v.1916 64 bit (AMD64)]', 'platform': 'Windows-10-10.0.19042-SP0',
'event': 'build_vocab'}
2022-11-11 16:10:16,081 : INFO : Word2Vec lifecycle event {'msg': 'training
model with 3 workers on 15173 vocabulary and 100 features, using sg=0 hs=0
sample=0.001 negative=5 window=5 shrink_windows=True', 'datetime':
'2022-11-11T16:10:16.081575', 'gensim': '4.2.0', 'python': '3.10.6 | packaged by
conda-forge | (main, Oct 24 2022, 16:02:16) [MSC v.1916 64 bit (AMD64)]',
'platform': 'Windows-10-10.0.19042-SP0', 'event': 'train'}
2022-11-11 16:10:17,128 : INFO : EPOCH 0 - PROGRESS: at 14.64% examples, 115933
words/s, in_qsize 0, out_qsize 0
2022-11-11 16:10:18,141 : INFO : EPOCH 0 - PROGRESS: at 44.52% examples, 181846
words/s, in_qsize 0, out_qsize 0
2022-11-11 16:10:19,158 : INFO : EPOCH 0 - PROGRESS: at 73.02% examples, 201093
words/s, in_qsize 0, out_qsize 0
2022-11-11 16:10:19,844 : INFO : EPOCH 0: training on 1161192 raw words (781703
effective words) took 3.8s, 208197 effective words/s
2022-11-11 16:10:20,856 : INFO : EPOCH 1 - PROGRESS: at 30.18% examples, 244046
words/s, in_qsize 0, out_qsize 0
2022-11-11 16:10:21,874 : INFO : EPOCH 1 - PROGRESS: at 61.21% examples, 260847
words/s, in_qsize 0, out_qsize 0
2022-11-11 16:10:22,861 : INFO : EPOCH 1: training on 1161192 raw words (782023
effective words) took 3.0s, 259921 effective words/s
2022-11-11 16:10:23,864 : INFO : EPOCH 2 - PROGRESS: at 31.82% examples, 260086
words/s, in_qsize 0, out_qsize 0
2022-11-11 16:10:24,882 : INFO : EPOCH 2 - PROGRESS: at 63.18% examples, 272567
words/s, in_qsize 0, out_qsize 0
2022-11-11 16:10:25,784 : INFO : EPOCH 2: training on 1161192 raw words (781609
effective words) took 2.9s, 266958 effective words/s
2022-11-11 16:10:26,792 : INFO : EPOCH 3 - PROGRESS: at 31.82% examples, 259924
words/s, in_qsize 0, out_qsize 0
2022-11-11 16:10:27,789 : INFO : EPOCH 3 - PROGRESS: at 62.56% examples, 270745
words/s, in_qsize 0, out_qsize 0
2022-11-11 16:10:28,730 : INFO : EPOCH 3: training on 1161192 raw words (781531
effective words) took 2.9s, 266293 effective words/s
```

```
2022-11-11 16:10:29,732 : INFO : EPOCH 4 - PROGRESS: at 31.82% examples, 259899
words/s, in_qsize 0, out_qsize 0
2022-11-11 16:10:30,750 : INFO : EPOCH 4 - PROGRESS: at 63.18% examples, 271597
words/s, in_qsize 0, out_qsize 0
2022-11-11 16:10:31,668 : INFO : EPOCH 4: training on 1161192 raw words (782022
effective words) took 2.9s, 265693 effective words/s
2022-11-11 16:10:31,668 : INFO : Word2Vec lifecycle event {'msg': 'training on
5805960 raw words (3908888 effective words) took 15.6s, 250772 effective
words/s', 'datetime': '2022-11-11T16:10:31.668630', 'gensim': '4.2.0', 'python':
'3.10.6 | packaged by conda-forge | (main, Oct 24 2022, 16:02:16) [MSC v.1916 64
bit (AMD64)]', 'platform': 'Windows-10-10.0.19042-SP0', 'event': 'train'}
2022-11-11 16:10:31,668 : INFO : Word2Vec lifecycle event {'params':
'Word2Vec<vocab=15173, vector_size=100, alpha=0.025>', 'datetime':
'2022-11-11T16:10:31.668630', 'gensim': '4.2.0', 'python': '3.10.6 | packaged by
conda-forge | (main, Oct 24 2022, 16:02:16) [MSC v.1916 64 bit (AMD64)]',
'platform': 'Windows-10-10.0.19042-SP0', 'event': 'created'}

CPU times: total: 24.5 s
Wall time: 19.1 s
```

The Google News is trained on roughly 100 billion words. Brown corpus is in the order of 1 million words. Hence the 'word2vec-google-news-300' is trained on roughly 100,000 as many words.

b) We will compare the Brown model to the 'word2vec-google-news-300'. Try to find the 10 nearest words first to car and then to queen in the two models. What do the examples reveal about the two training corpora?

[31]:
```python
def best_list(model, positive=[], negative=[], topn=5):
    res = model.most_similar(positive=positive, negative=negative, topn=topn)
    for (w, sim) in res:
        print("{:15} score {:6.3f}".format(w, sim))
```

[32]:
```python
print('car')
print('========')
for nom, mod in [('Brown', model.wv), ('Google', wv)]:
    print(nom)
    best_list(mod, 'car', topn=10)
    print(' ')
```

```
car
========
Brown
house           score  0.947
hall            score  0.905
room            score  0.905
corner          score  0.902
town            score  0.901
road            score  0.895
desk            score  0.892
```

```
bed             score  0.892
jig             score  0.883
fire            score  0.880

Google
vehicle         score  0.782
cars            score  0.742
SUV             score  0.716
minivan         score  0.691
truck           score  0.674
Car             score  0.668
Ford_Focus      score  0.667
Honda_Civic     score  0.663
Jeep            score  0.651
pickup_truck    score  0.644
```

[33]:
```
print('queen')
print('========')
for nom, mod in [('Brown', model.wv), ('Google', wv)]:
    print(nom)
    best_list(mod, 'queen', topn=10)
    print(' ')
```

```
queen
========
Brown
calf            score  0.952
gown            score  0.950
Sloan           score  0.950
governor        score  0.948
gentle          score  0.947
shock           score  0.945
nervous         score  0.944
surgeon         score  0.941
blonde          score  0.938
cigarette       score  0.938

Google
queens          score  0.740
princess        score  0.707
king            score  0.651
monarch         score  0.638
very_pampered_McElhatton score  0.636
Queen           score  0.616
NYC_anglophiles_aflutter score  0.606
Queen_Consort   score  0.592
princesses      score  0.591
```

```
royal           score  0.564
```

Comment Google which is trained on a larger corpus is more specific. The similar words denote more similar objects than in the Brown model.

The two corpora reflect the period they were collected. Concepts like 'SUV' and 'minivan' were not invented in 1960 (Brown). On the other hand Brown seems to reflect its period directly where 'queen' gets similar words connected to the British queen's role during World War II.

We also observe that the Google news also include some multi-word terms.

0.5 Exercise 4 Evaluation

Gensim comes with several methods for evaluation, and also standard datasets for the tests. Testsets could be found by the tha datapath command, e.g.

[34]:
```
from gensim.test.utils import datapath
path=datapath('questions-words.txt')
```

```
2022-11-11 16:10:32,212 : INFO : adding document #0 to Dictionary<0 unique
tokens: []>
2022-11-11 16:10:32,212 : INFO : built Dictionary<12 unique tokens: ['computer',
'human', 'interface', 'response', 'survey']…> from 9 documents (total 29
corpus positions)
2022-11-11 16:10:32,212 : INFO : Dictionary lifecycle event {'msg': "built
Dictionary<12 unique tokens: ['computer', 'human', 'interface', 'response',
'survey']…> from 9 documents (total 29 corpus positions)", 'datetime':
'2022-11-11T16:10:32.212702', 'gensim': '4.2.0', 'python': '3.10.6 | packaged by
conda-forge | (main, Oct 24 2022, 16:02:16) [MSC v.1916 64 bit (AMD64)]',
'platform': 'Windows-10-10.0.19042-SP0', 'event': 'created'}
```

One test you may use is to see how well the model perform on the Google analogy test datset. This can be run by `<model>.evaluate_word_analogies(path)` Report the key numbers, and try to understand what they mean.

[35]:
```
%%time

google_evaluate = wv.evaluate_word_analogies(path)
```

```
2022-11-11 16:10:32,839 : INFO : Evaluating word analogies for top 300000 words
in the model on C:\Users\jtl\Anaconda3\envs\in4080_2022_transform\lib\site-
packages\gensim\test\test_data\questions-words.txt
2022-11-11 16:10:44,482 : INFO : capital-common-countries: 83.2% (421/506)
2022-11-11 16:12:24,060 : INFO : capital-world: 81.3% (3552/4368)
2022-11-11 16:12:42,605 : INFO : currency: 28.5% (230/808)
2022-11-11 16:13:39,093 : INFO : city-in-state: 72.1% (1779/2467)
2022-11-11 16:13:50,841 : INFO : family: 86.2% (436/506)
2022-11-11 16:14:13,582 : INFO : gram1-adjective-to-adverb: 29.2% (290/992)
2022-11-11 16:14:32,264 : INFO : gram2-opposite: 43.5% (353/812)
```

```
2022-11-11 16:15:02,590 : INFO : gram3-comparative: 91.3% (1216/1332)
2022-11-11 16:15:28,442 : INFO : gram4-superlative: 88.0% (987/1122)
2022-11-11 16:15:52,672 : INFO : gram5-present-participle: 78.5% (829/1056)
2022-11-11 16:16:29,228 : INFO : gram6-nationality-adjective: 90.2% (1442/1599)
2022-11-11 16:17:05,096 : INFO : gram7-past-tense: 65.4% (1020/1560)
2022-11-11 16:17:35,432 : INFO : gram8-plural: 87.0% (1159/1332)
2022-11-11 16:17:56,609 : INFO : gram9-plural-verbs: 68.2% (593/870)
2022-11-11 16:17:56,617 : INFO : Quadruplets with out-of-vocabulary words: 1.1%
2022-11-11 16:17:56,617 : INFO : NB: analogies containing OOV words were skipped
from evaluation! To change this behavior, use "dummy4unknown=True"
2022-11-11 16:17:56,617 : INFO : Total accuracy: 74.0% (14307/19330)

CPU times: total: 28min 10s
Wall time: 7min 24s
```

[36]:
```
google_evaluate[0]
```

[36]:
```
0.7401448525607863
```

[37]:
```
%%time

brown_evaluate = model.wv.evaluate_word_analogies(path)
```

```
2022-11-11 16:17:56,680 : INFO : Evaluating word analogies for top 300000 words
in the model on C:\Users\jtl\Anaconda3\envs\in4080_2022_transform\lib\site-
packages\gensim\test\test_data\questions-words.txt
2022-11-11 16:17:56,730 : INFO : capital-common-countries: 1.1% (1/90)
2022-11-11 16:17:56,768 : INFO : capital-world: 0.0% (0/44)
2022-11-11 16:17:56,780 : INFO : currency: 0.0% (0/12)
2022-11-11 16:17:56,981 : INFO : city-in-state: 0.2% (1/457)
2022-11-11 16:17:57,082 : INFO : family: 25.7% (54/210)
2022-11-11 16:17:57,454 : INFO : gram1-adjective-to-adverb: 0.5% (4/756)
2022-11-11 16:17:57,535 : INFO : gram2-opposite: 0.0% (0/132)
2022-11-11 16:17:58,148 : INFO : gram3-comparative: 6.7% (71/1056)
2022-11-11 16:17:58,277 : INFO : gram4-superlative: 0.0% (0/210)
2022-11-11 16:17:58,641 : INFO : gram5-present-participle: 2.9% (19/650)
2022-11-11 16:17:58,815 : INFO : gram6-nationality-adjective: 0.7% (2/297)
2022-11-11 16:17:59,356 : INFO : gram7-past-tense: 2.1% (27/1260)
2022-11-11 16:17:59,607 : INFO : gram8-plural: 0.7% (4/552)
2022-11-11 16:17:59,786 : INFO : gram9-plural-verbs: 1.2% (4/342)
2022-11-11 16:17:59,788 : INFO : Quadruplets with out-of-vocabulary words: 69.0%
2022-11-11 16:17:59,788 : INFO : NB: analogies containing OOV words were skipped
from evaluation! To change this behavior, use "dummy4unknown=True"
2022-11-11 16:17:59,788 : INFO : Total accuracy: 3.1% (187/6068)

CPU times: total: 9.92 s
Wall time: 3.12 s
```

[38]:
```
brown_evaluate[0]
```

[38]:
```
0.03081740276862228
```

We see that the Brwon corpus does not perform too well on the analogy set while the Google news model manages about a 75% score.

0.6 Exercise 5 Application

a) Train and test a logistic regression classifier as described. Tune the C parameter (regularization, cf. Mandatory 2.A). Report the results from the tuning in a table. How does this classifier perform compared to your results from Mandatory assignment 1?

[39]:
```
from sklearn.linear_model import LogisticRegression
```

[40]:
```
## We use the already tokenized documents

from nltk.corpus import movie_reviews

words_movie_docs = [(list(movie_reviews.words(fileid)), category) for
             category in movie_reviews.categories() for fileid in
             movie_reviews.fileids(category)]
```

[41]:
```
docs = [d for d,_ in words_movie_docs]
cats = [c for d_,c in words_movie_docs]
y = np.array(cats)
```

[42]:
```
indicies = [i for i in range(len(docs))]
```

[43]:
```
import random
random.seed(42)
random.shuffle(indicies)
indicies = np.array(indicies)
```

[44]:
```
ind_test = indicies[:200]
ind_dev = indicies[200:]
ind_dev_test = ind_dev[:200]
ind_train = ind_dev[200:]
```

[45]:
```
%%time
corpus = [w for d in docs for w in d]
len(corpus)
```

```
CPU times: total: 62.5 ms
Wall time: 62.4 ms
```

[45]:
```
1583820
```

[46]:
```
corpus_vokab = set(corpus)
len(corpus_vokab)
```

[46]: 39768

[47]:
```
vokab = corpus_vokab.intersection(wv.index_to_key)
```

[48]:
```
len(vokab)
```

[48]: 31866

[49]:
```
def doc_vector(dok, model, vokab, binary=False, normal=False):
    if binary: dok = set(dok)
    vectors = [model[w] for w in dok if w in vokab]
    le = len(vectors)
    s = sum(vectors)
    vector = s/le
    if normal:
        norm = norm_1(vector)
        vector/norm
    return vector
```

[50]:
```
y_train = y[ind_train]
y_test = y[ind_dev_test]
```

[51]:
```
%%time
X = np.array([doc_vector(d, wv, vokab) for d in docs])
```

```
CPU times: total: 2.81 s
Wall time: 3.54 s
```

[52]:
```
X.shape
```

[52]: (2000, 300)

[53]:
```
X_train = X[ind_train]
X_test = X[ind_dev_test]
```

[54]:
```
for c in [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]:
    clf=LogisticRegression(C=c, max_iter=500)
    clf.fit(X_train, y_train)
    acc = clf.score(X_test, y_test)
    print(c, acc)
```

```
0.01 0.47
0.1 0.615
1.0 0.735
10.0 0.81
```

```
100.0 0.835
1000.0 0.855
```

[55]:
```
for c in [10.0, 50.0, 80.0, 100.0, 150.0, 200.0, 300.0, 500.0]:
    clf=LogisticRegression(C=c, max_iter=500)
    clf.fit(X_train, y_train)
    acc = clf.score(X_test, y_test)
    print(c, acc)
```

```
10.0 0.81
50.0 0.835
80.0 0.84
100.0 0.835
150.0 0.86
200.0 0.86
300.0 0.85
500.0 0.85
```

n-fold We run with 9-fold to get comparable numbers to Mandatory 1B

[56]:
```
folds = 9
results=[]
l = len(ind_dev)
for i in range(folds):
    b = int(i*l/folds)
    e = int((i+1)*l/folds)
    X_te = X[ind_dev[b:e], :]
    y_te = y[ind_dev[b:e]]
    X_tr = np.concatenate([X[ind_dev[:b], :], X[ind_dev[e:], :]], axis=0)
    y_tr = np.concatenate([y[ind_dev[:b]], y[ind_dev[e:]]])
    clf=LogisticRegression(C=200, max_iter=1500)
    clf.fit(X_tr, y_tr)
    acc = clf.score(X_te, y_te)
    results.append(acc)
    results

results
```

[56]:
```
[0.86, 0.875, 0.865, 0.785, 0.82, 0.865, 0.835, 0.815, 0.79]
```

[57]:
```
round(np.mean(results),4)
```

[57]:
```
0.8344
```

The results are inferior compared to to Mandatory 1B using binary bow model.

0.7 Extension: tSNE

[58]:
```
from sklearn.manifold import TSNE
```

[59]:
```
ts = TSNE(perplexity=6, random_state=42)
```

[60]:
```
words = "king queen prince princess man woman boy girl son daughter".split()
```

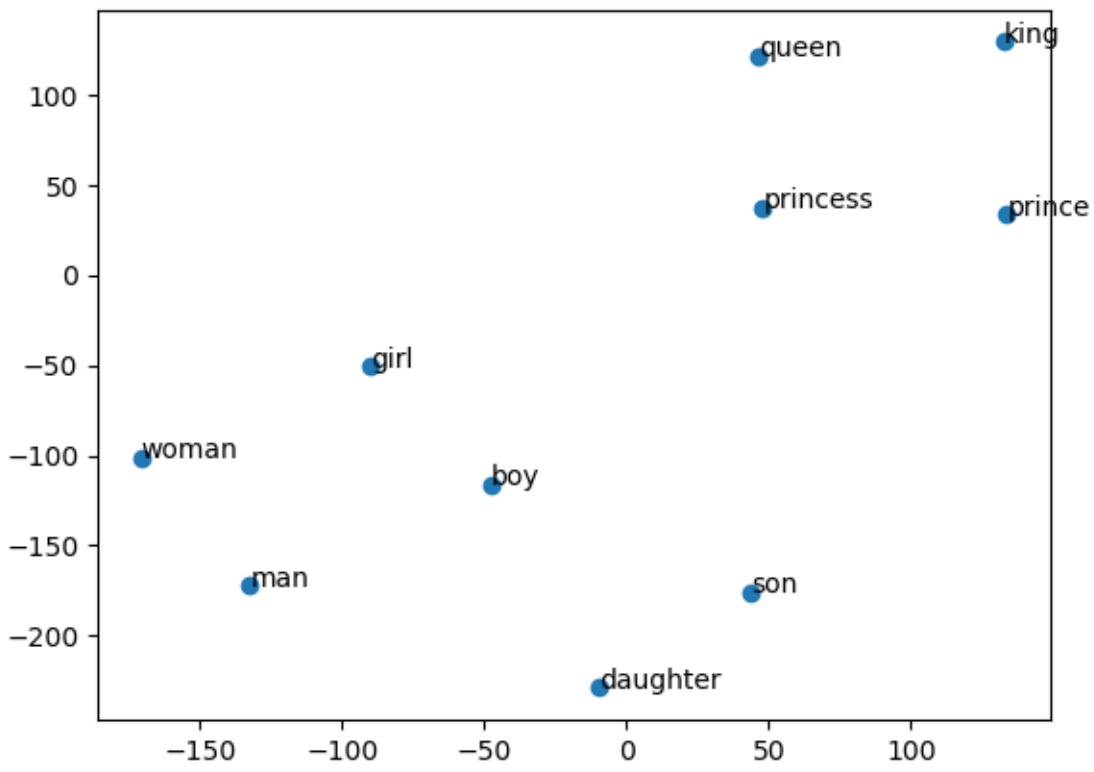
[61]:
```
embs = [wv[word] for word in words]
points = ts.fit_transform(np.array(embs))
```

```
C:\Users\jtl\Anaconda3\envs\in4080_2022_transform\lib\site-
packages\sklearn\manifold\_t_sne.py:800: FutureWarning: The default
initialization in TSNE will change from 'random' to 'pca' in 1.2.
  warnings.warn(
C:\Users\jtl\Anaconda3\envs\in4080_2022_transform\lib\site-
packages\sklearn\manifold\_t_sne.py:810: FutureWarning: The default learning
rate in TSNE will change from 200.0 to 'auto' in 1.2.
  warnings.warn(
```

[62]:
```
x=[p[0] for p in points]
y=[p[1] for p in points]

fig, ax = plt.subplots()
ax.scatter(x, y)

for i, txt in enumerate(words):
    ax.annotate(txt, (x[i], y[i]))
```


queen
king
princess
prince
girl
woman
boy
man
son
daughter
100
50
0
−50
−100
−150
−200
−150
−100
−50
0
50
100