Natural Language Processing (NLP)

Word Embedding

Instructor: Santosh Chapaneri

Sep 2021

Playing with vectors

In [1]:
```
import numpy as np
from nltk.corpus import wordnet
from collections import OrderedDict
from itertools import combinations
```

Normalizing the vector:

$\hat{\mathbf{u}} = \frac{\mathbf{u}}{||\mathbf{u}||}$, where $||\mathbf{u}|| = \sqrt{\mathbf{u} \cdot \mathbf{u}}$ is the norm of vector $\mathbf{u}$

In [2]:
```
u = np.array([1, 2, 4])

# Norm
np.sqrt(u.dot(u))
```

Out[2]:
```
4.58257569495584
```

In [3]:
```
# In-built norm
np.linalg.norm(u)
```

Out[3]:
```
4.58257569495584
```

In [4]:
```
# Normalize the vector so that all its values are between 0 and 1
def normalize_vector(vector):
    norm = np.linalg.norm(vector)
    if norm:
        return vector / norm
    else:
        # if norm == 0, then original vector was all 0s
        return vector
```

In [5]:
```
print("original vector", u)

print("normalized vector", normalize_vector(u))

# 0.218 is 1/4th of 0.873 just like 1 is 1/4th of 4
```

```
original vector [1 2 4]
normalized vector [0.21821789 0.43643578 0.87287156]
```

Calculating Cosine Similarity

In [6]:
```
def cos_sim(u, v):
    # ensure that both vectors are already normalized
    u_norm = normalize_vector(u)
    v_norm = normalize_vector(v)

    # calculate the dot product between the two normalized vectors
    return u_norm.dot(v_norm)
```

In [7]:
```
x = np.array([1, -1, 2])
y = np.array([2, 3, -1])

print("cosine similarity of u1 and u2", cos_sim(x, y))
```

```
cosine similarity of u1 and u2 -0.32732683535398865
```

In [8]:
```
u1 = np.array([1, 1, 1, 1, 1])
u2 = np.array([1, 1, 1, 1, 2])
u3 = np.array([1, 2, 3, 4, 5])
u4 = np.array([10, 20, 30, 40, 50])

print("cosine similarity of u1 and u2", cos_sim(u1, u2))
print("cosine similarity of u1 and u3", cos_sim(u1, u3))
print("cosine similarity of u1 and u4", cos_sim(u1, u4))
```

```
cosine similarity of u1 and u2 0.9486832980505137
cosine similarity of u1 and u3 0.9045340337332909
cosine similarity of u1 and u4 0.9045340337332909
```

Word to Feature Vector Representation

- **Can we use one-hot feature vector for each word?**

Encode spelling representation

Let's build word vectors represented by the frequency of the letters present

In [23]:
```
import string
alphabet = list(string.ascii_lowercase)
alphabet
```

Out[23]:
```
['a',
 'b',
 'c',
 'd',
 'e',
 'f',
 'g',
 'h',
 'i',
 'j',
 'k',
 'l',
 'm',
 'n',
 'o',
 'p',
 'q',
 'r',
 's',
 't',
 'u',
 'v',
 'w',
 'x',
 'y',
 'z']
```

In [26]:
```
# We don't need to worry about "out-of-vocabulary" now
def lookup_letter(letter):
    return alphabet.index(letter.lower())

print("a", lookup_letter('a'))
print("C", lookup_letter('C'))
```

```
a 0
C 2
```

In [27]:
```
# Converts a word into a vector of dimension 26
# where each cell contains the count for that letter
def make_spelling_vector(word):
    # initialize vector with zeros
    spelling_vector = np.zeros((26))

    # iterate through each letter and update count
    for letter in word:
        if letter in string.ascii_letters:
            letter_index = lookup_letter(letter)
            spelling_vector[letter_index] += 1
    return spelling_vector
```

In [28]:
```
make_spelling_vector("apple")
```

Out[28]:
```
array([1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 2., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [29]:
```
vocabulary = ['apple', 'banana', 'orange', 'cantaloupe', 'peach']

# Add an OOV word to vocabulary
vocabulary_plus_oov = vocabulary + ["Pune"]

all_combinations = combinations(vocabulary_plus_oov, 2)

# iterate through all words
for (word1, word2) in all_combinations:
    spelling_vector_1 = make_spelling_vector(word1)
    spelling_vector_2 = make_spelling_vector(word2)
    print("cosine similarity between {} and {}".format(word1, word2),
          cos_sim(spelling_vector_1, spelling_vector_2))
```

```
cosine similarity between apple and banana 0.3030457633656632
cosine similarity between apple and orange 0.3086066999241838
cosine similarity between apple and cantaloupe 0.6546536707079772
cosine similarity between apple and peach 0.6761234037828132
cosine similarity between apple and Pune 0.5669467095138407
cosine similarity between banana and orange 0.545544725589981
cosine similarity between banana and cantaloupe 0.6172133998483678
cosine similarity between banana and peach 0.3585685828003181
cosine similarity between banana and Pune 0.2672612419124244
cosine similarity between orange and cantaloupe 0.5892556509887897
cosine similarity between orange and peach 0.36514837167011077
cosine similarity between orange and Pune 0.4082482904638631
cosine similarity between cantaloupe and peach 0.6454972243679029
cosine similarity between cantaloupe and Pune 0.5773502691896258
cosine similarity between peach and Pune 0.4472135954999579
```

- We've successfully generated similarity scores! But...
- Do they really reflect anything **semantic**?
- In other words, does it make sense that "apple" and "Pune" (cosine similarity = 0.567) are more similar than "apple" and "orange" (cosine similarity = 0.308)?

Word Embeddings

- Create a "dense" representation of each word where proximity in vector space represents "similarity".

In [30]:
```
import gensim.downloader as api

w2v = api.load("glove-wiki-gigaword-100")
# load pre-trained word-vectors from gensim-data
# each word has a 100-dimensional vector representation
```

```
[==================================================] 100.0% 128.1/128.1MB downloaded
```

In [31]:
```
w2v["the"][0:50]
```

Out[31]:
```
array([-0.038194, -0.24487 ,  0.72812 , -0.39961 ,  0.083172,  0.043953,
       -0.39141 ,  0.3344  , -0.57545 ,  0.087459,  0.28787 , -0.06731 ,
        0.30906 , -0.26384 , -0.13231 , -0.20757 ,  0.33395 , -0.33848 ,
       -0.31743 , -0.48336 ,  0.1464  , -0.37304 ,  0.34577 ,  0.052041,
        0.44946 , -0.46971 ,  0.02628 , -0.54155 , -0.15518 , -0.14107 ,
       -0.039722,  0.28277 ,  0.14393 ,  0.23464 , -0.31021 ,  0.086173,
        0.20397 ,  0.52624 ,  0.17164 , -0.082378, -0.71787 , -0.41531 ,
        0.20335 , -0.12763 ,  0.41367 ,  0.55187 ,  0.57908 , -0.33477 ,
       -0.36559 , -0.54857 ], dtype=float32)
```

In [32]:
```
len(w2v["the"])
```

Out[32]:
```
100
```

Perform some vector algebra on words:

In [34]:
```
# Example 1
d1 = w2v['king'] - w2v['man']
d2 = w2v['queen'] - w2v['woman']

d = d1 - d2
np.sqrt(np.mean(d**2))
```

Out[34]:
```
0.4081079
```

In [35]:
```
# Example 2
d1 = w2v['king'] - w2v['man']
d2 = w2v['cat'] - w2v['desk']

d = d1 - d2
np.sqrt(np.mean(d**2))
```

Out[35]:
```
0.89675885
```

Evaluation of word embeddings

In [36]:
```
# king - man + woman = queen
sol = w2v.most_similar(
    positive = ['king', 'woman'],
    negative = ['man'],
    topn = 5)
print(sol)
```

```
[('queen', 0.7698541283607483), ('monarch', 0.6843380928039551), ('throne', 0.6755735874176025),
('daughter', 0.6594556570053101), ('princess', 0.6520534753799438)]
```

In [37]:
```
# mouse - dollar + dollars = mice
sol = w2v.most_similar(
    positive = ['mouse', 'dollars'],
    negative = ['dollar'],
    topn = 5)
print(sol)
```

```
[('mice', 0.5946231484413147), ('rabbit', 0.5506148338317871), ('cat', 0.5420147180557251), ('spid
er', 0.5348110198974609), ('clone', 0.5339481830596924)]
```

In [38]:
```
# brother - uncle + aunt = sister
sol = w2v.most_similar(
    positive = ['brother','aunt'],
    negative = ['uncle'],
    topn = 5)
print(sol)
```

```
[('wife', 0.8692924976348877), ('mother', 0.8691741228103638), ('daughter', 0.8637559413909912),
('sister', 0.8537876605987549), ('grandmother', 0.8403504490852356)]
```

In [39]:
```
# find most similar n words to a given word
similar = w2v.similar_by_word("queen", topn = 10)
similar
```

Out[39]:
```
[('princess', 0.7947245240211487),
 ('king', 0.7507690787315369),
 ('elizabeth', 0.7355712652206421),
 ('royal', 0.7065026760101318),
 ('lady', 0.7044796943664551),
 ('victoria', 0.6853758096694946),
 ('monarch', 0.6683257818222046),
 ('crown', 0.6680562496185303),
 ('prince', 0.6640505790710449),
 ('consort', 0.6570538282394409)]
```

In [42]:
```
# find  most similar n words to a given vector
cat_vector = w2v['cat']

cat_sim = w2v.similar_by_vector(cat_vector, topn = 10)
cat_sim
```

Out[42]:
```
[('cat', 1.0),
 ('dog', 0.8798074722290039),
 ('rabbit', 0.7424426674842834),
 ('cats', 0.7323004007339478),
 ('monkey', 0.7288709878921509),
 ('pet', 0.7190139889717102),
 ('dogs', 0.7163872718811035),
 ('mouse', 0.6915251016616821),
 ('puppy', 0.6800068020820618),
 ('rat', 0.6641027331352234)]
```

In [43]:
```
# find which word doesn't match
list_of_words = "breakfast cereal dinner lunch"

doesnt_match = w2v.doesnt_match(list_of_words.split())
doesnt_match
```

```
/usr/local/lib/python3.7/dist-packages/gensim/models/keyedvectors.py:895: FutureWarning: arrays to
stack must be passed as a "sequence" type such as list or tuple. Support for non-sequence iterable
s such as generators is deprecated as of NumPy 1.16 and will raise an error in the future.
  vectors = vstack(self.word_vec(word, use_norm=True) for word in used_words).astype(REAL)
```

Out[43]:
```
'cereal'
```