Word embeddings with spaCy and gensim

Introduction to NLP

Tong Liu, with Cecilia O. Alm

Use vector models

Theory of Similarity:

- Words in similar contexts tend to share similar meanings.
- "*If A and B have almost identical environments... we say that they are synonyms.*" (Harris, 1954)

Distributional methods: **vector space models**

- Compute semantic similarity (vector semantics)
    - E.g. question answering, summarization, machine translation, ...
- Represent words as features in NLP tasks
    - E.g. named entity recognition, parsing, semantic role labeling, ...

⇨ Model a word by *embedding* it into a vector space → **word embeddings**

- **Embedding**: one instance contained within another instance in some mathematical structure
- A word's meaning is a vector of numbers

Harris, Zellig S. "Distributional structure." *Word* 10.2-3 (1954): 146-162.

Words and vectors for representing meaning

- Co-occurrence matrix
  - Term-document matrix
  - Term-term matrix (word-word matrix, term-context matrix)
- A vector space model is represented as vectors with dimensions (Salton, 1971)
- There are **|V|** rows. Each row represents a word type in the vocab dictionary
- Term-document matrix:
  - Tabulates frequency of terms in documents of the collection
  - |V| x |D|
- Word-word matrix:
  - For words, tabulates other words co-occurring in their contexts
  - |V| x |V|

Salton, Gerard. "The SMART retrieval system—experiments in automatic document processing." (1971).

Weighing and measuring word associations

There are two fundamental types of word associations:

- **First-order co-occurrence** (syntagmatic): words occur nearby **each other**
  - **write** an **essay**
  - *make pancakes* v.s. *do homework*

- **Second-order co-occurrence** (paradigmatic): words have similar *neighbors*
  - **write/compose** *an essay/a letter/a report*

1. Church, Kenneth, et al. "Parsing, word associations and typical predicate-argument relations." Proceedings of the workshop on Speech and Natural Language. Association for Computational Linguistics, 1989.
2. Luhn, Hans Peter. "A statistical approach to mechanized encoding and searching of literary information." *IBM Journal of research and development* 1.4 (1957): 309-317.
3. Curran, James Richard. "From distributional to semantic similarity." (2004).
4. Jurafsky, Dan and H. Martin James. Speech & language processing. 3rd edition

- Positive Pointwise Mutual Information (PPMI) [1]

$$\text{PPMI}(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0)$$

w: word;
c: context

- Term frequency-inverse document freq (tf-idf) [2]

$$w_{ij} = \text{tf}_{ij}\text{idf}_i$$

word *i* in document *j*

- Hypothesis testing (t-test association measure) [3]

$$\text{t-test}(a,b) = \frac{P(a,b) - P(a)P(b)}{\sqrt{P(a)P(b)}}$$

a and *b* are two words

- Measuring vector similarity via cosine similarity [4]

$$\text{cosine}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

v and *w* are two vectors for words *a* and *b*

Why dense vectors?

- They have advantages over long and sparse vectors:
  - More straightforward to include as features in machine learning → less parameters to tune
  - Generalize better and help avoid model overfitting
  - Claim to capture synonymy better

- Three methods:
  - Dimensionality reduction methods (singular value decomposition)
    - A special case: Latent Semantic Analysis
  - Neural network methods - `word2vec`: skip-gram vs. CBOW [1, 2]
    - Learn word embeddings in the process of neighboring words prediction
    - Computationally fast and easy to train
    - Pre-trained embeddings available online to use
  - Brown clustering

1. Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781*. 2013.
2. Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.

spaCy

- *comes with* pre-trained, real-valued dense word vectors
  - Trained using **<u>GloVe</u>** algorithm
    - Global Vectors for Word Representation

- *facilitates* the usage of word vectors
  - The **Lexeme**, **Token**, **Span** and **Doc** classes have a ***.vector*** property (300d vectors, 32-bit floats)

- Vectors *pre-trained* from <u>Common Crawl</u> (commoncrawl.org) website data

https://spacy.io/docs/usage/word-vectors-similarities (with example)

apple vector from *GloVe* in *spacy*

```
>>> import spacy
>>> model = spacy.load('en_vectors_glove_md')
>>> doc = model('apple and orange')
>>> doc[0]
apple
>>> doc[0].vector
array([ -3.63909990e-01,   4.37709987e-01,  -2.04469994e-01,
        -2.28890002e-01,  -1.42269999e-01,   2.73959994e-01,
        -1.14350002e-02,  -1.85780004e-01,   3.73609990e-01,
         7.53390014e-01,  -3.05909991e-01,   2.37409994e-02,
        -7.78760016e-01,  -1.38019994e-01,   6.69919997e-02,
        -6.43030033e-02,  -4.00240004e-01,   1.53090000e+00,
        -1.38969999e-02,  -1.56570002e-01,   2.53659993e-01,
         2.16100007e-01,  -3.27199996e-01,   3.49739999e-01,
        -6.48450032e-02,  -2.95010000e-01,  -6.39230013e-01,
        -6.20170012e-02,   2.45590001e-01,  -6.93340003e-02,
        -3.99670005e-01,   3.09250001e-02,   4.90330011e-01,
         6.75239980e-01,   1.94810003e-01,   5.14880002e-01,
        -3.11489999e-01,  -7.99390003e-02,  -6.20959997e-01,
        -5.32770017e-03,  -1.12640001e-01,   8.35279971e-02,
        -7.69469980e-03,  -1.07879996e-01,   1.66280001e-01,
         4.22729999e-01,  -1.90090001e-01,  -2.90349990e-01,
         4.56300005e-02,   1.01199999e-01,  -4.08549994e-01,
        -3.49999994e-01,  -3.61750007e-01,  -4.13960010e-01,
         5.94850004e-01,  -1.15240002e+00,   3.24239992e-02,
         3.43640000e-01,  -1.92090005e-01,   4.32550013e-02,
         4.92269993e-02,  -5.42580009e-01,   9.12750006e-01,
         2.95760006e-01,   2.36579999e-02,  -6.87370002e-01,
        -1.95030004e-01,  -1.10590003e-01,  -2.25669995e-01,
         2.41799995e-01,  -3.12299997e-01,   4.26999986e-01,
         8.39520022e-02,   2.27029994e-01,   3.05810004e-01,
        -1.72759995e-01,   3.25360000e-01,   5.46960020e-03,
        -3.27450007e-01,   1.94389999e-01,   2.26160005e-01,
         7.47419968e-02,   2.20330000e-01,  -4.03010011e-01,
        -3.15939993e-01,  -2.89099999e-02,   9.78579998e-01,
         7.18599975e-01,   1.49949998e-01,   6.34210035e-02,
         2.83320010e-01,  -1.52309999e-01,   3.93299997e-04,
         1.80759996e-01,  -4.01989996e-01,   6.01870008e-02,
        -2.75430009e-02,   1.65900007e-01,  -2.57739991e-01,
         1.61500007e-01,   3.72469991e-01,  -3.82730007e-01,
         2.40119994e-01,  -4.26170006e-02,  -6.67850018e-01,
        -9.44369972e-01,   2.79159993e-01,   1.04759999e-01,
         1.39520001e+00,  -1.42959997e-01,  -5.50490022e-01,
         5.39820008e-02,  -7.75240004e-01,  -2.82550007e-01,
        -2.33229995e-02,   2.48009995e-01,   2.28550002e-01,
```

gensim

- *reads* pre-trained, real-valued dense word vectors

- *provided* here with the GoogleNews-vectors-negative300.bin.gz
    - Downloaded from: https://github.com/mmihaltz/word2vec-GoogleNews-vectors
    - Mirrors the model on the original word2vec website: https://code.google.com/archive/p/word2vec/

- Vectors *pre-trained* from Google News corpus data (~300B total words and phrases) using the **word2vec** algorithm

apple vector from *word2vec* in *gensim*

```
>>> import gensim
Using TensorFlow backend.
>>> model = gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative3
00.bin.gz', binary=True)
>>> model.wv['apple']
array([-0.06445312, -0.16015625, -0.01208496,  0.13476562, -0.22949219,
        0.16210938,  0.3046875 , -0.1796875 , -0.12109375,  0.25390625,
       -0.01428223, -0.06396484, -0.08056641, -0.05688477, -0.19628906,
        0.2890625 , -0.05151367,  0.14257812, -0.10498047, -0.04736328,
       -0.34765625,  0.35742188,  0.265625  ,  0.00188446, -0.01586914,
        0.00195312, -0.35546875,  0.22167969,  0.05761719,  0.15917969,
        0.08691406, -0.0267334 , -0.04785156,  0.23925781, -0.05981445,
        0.0378418 ,  0.17382812, -0.41796875,  0.2890625 ,  0.32617188,
        0.02429199, -0.01647949, -0.06494141, -0.08886719,  0.07666016,
       -0.15136719,  0.05249023, -0.04199219, -0.05419922,  0.00108337,
       -0.20117188,  0.12304688,  0.09228516,  0.10449219, -0.00408936,
       -0.04199219,  0.01409912, -0.02111816, -0.13476562, -0.24316406,
        0.16015625, -0.06689453, -0.08984375, -0.07177734, -0.00595093,
       -0.00482178, -0.00089264, -0.30664062, -0.0625    ,  0.07958984,
       -0.00909424, -0.04492188,  0.09960938, -0.33398438, -0.3984375 ,
        0.05541992, -0.06689453, -0.04467773,  0.11767578, -0.13964844,
       -0.26367188,  0.17480469, -0.17382812, -0.40625   , -0.06738281,
       -0.07617188,  0.09423828,  0.20996094, -0.16308594, -0.08691406,
       -0.0534668 , -0.10351562, -0.07617188, -0.11083984, -0.03515625,
       -0.14941406,  0.0378418 ,  0.38671875,  0.14160156, -0.2890625 ,
       -0.16894531, -0.140625  , -0.04174805,  0.22753906,  0.24023438,
       -0.01599121, -0.06787109,  0.21875   , -0.42382812, -0.5625    ,
       -0.49414062, -0.3359375 ,  0.13378906,  0.01141357,  0.13671875,
        0.0324707 ,  0.06835938, -0.27539062, -0.15917969,  0.00121307,
        0.01208496, -0.0039978 ,  0.00442505, -0.04541016,  0.08642578,
        0.09960938, -0.04296875, -0.11328125,  0.13867188,  0.41796875,
       -0.28320312, -0.07373047, -0.11425781,  0.08691406, -0.02148438,
        0.328125  , -0.07373047, -0.01348877,  0.17773438, -0.02624512,
        0.13378906, -0.11132812, -0.12792969, -0.12792969,  0.18945312,
       -0.13867188,  0.29882812, -0.07714844, -0.37695312, -0.10351562,
        0.16992188, -0.10742188, -0.29882812,  0.00866699, -0.27734375,
       -0.20996094, -0.1796875 , -0.19628906, -0.22167969,  0.08886719,
       -0.27734375, -0.13964844,  0.15917969,  0.03637695,  0.03320312,
       -0.08105469,  0.25390625, -0.08691406, -0.21289062, -0.18945312,
       -0.22363281,  0.06542969, -0.16601562,  0.08837891, -0.359375  ,
       -0.09863281,  0.35546875, -0.00741577,  0.19042969,  0.16992188,
       -0.06005859, -0.20605469,  0.08105469,  0.12988281, -0.01135254,
        0.33203125, -0.08691406,  0.27539062, -0.03271484,  0.12011719,
       -0.0625    ,  0.1953125 , -0.10986328, -0.11767578,  0.20996094,
        0.19921875,  0.02954102, -0.16015625,  0.00276184, -0.01367188,
        0.03442383, -0.19335938,  0.00352478, -0.06542969, -0.05566406,
        0.09423828,  0.29296875,  0.04052734, -0.09326172, -0.10107422,
       -0.27539062,  0.04394531, -0.07275391,  0.13867188,  0.02380371,
        0.13085938,  0.00236511, -0.2265625 ,  0.34765625,  0.13574219,
```

GloVe VS. Word2vec

- Similarities:
    - Both methods learn geometrical embeddings of words based on their co-occurrence information.
- Differences in approaches:
    - GloVe -- “count-based” model
        - Factorizes co-occurrence count matrix to get a lower-dimensional word matrix
        - Computational advantage - parallelization
    - word2vec -- “predictive” model
        - Predicts the n^{th} word given prior words [1,...,n-1] to get word vectors
        - Or the other way round to get context vectors

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation." EMNLP. Vol. 14. 2014.
Word2vec: https://code.google.com/archive/p/word2vec/
Baroni, Marco, Georgiana Dinu, and Germán Kruszewski. "Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors." ACL (1). 2014.

Word analogies

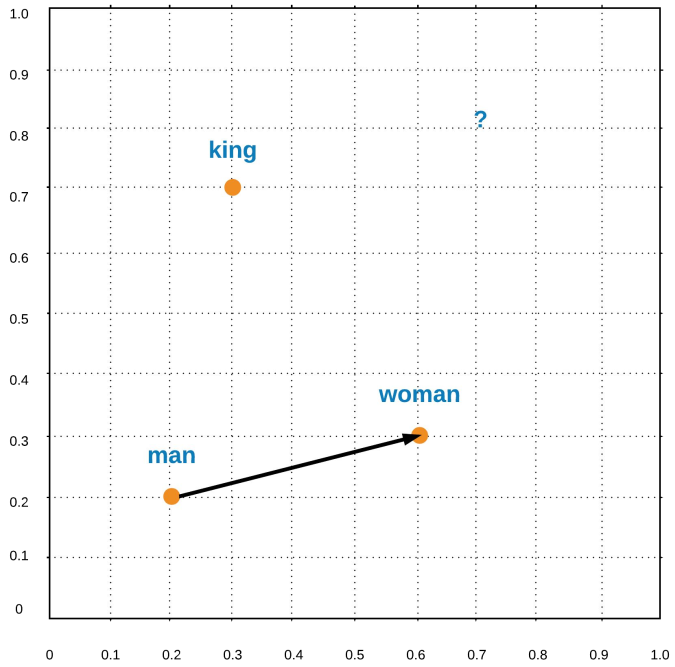
Man : Woman = King : **?**

Linear relationships (examined by Mikolov et al.)

King - ***Man*** + ***Woman*** = ?

What if:

King	[0.30, 0.70]
Man	[0.20, 0.20]
Woman	[0.60, 0.30]
?	[0.70, 0.80]



Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013. Figure on this and next slides modified and adapted from http://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf

Your turn - download embeddings.zip

Tasks 1-2: Load vectors and explore similarity between words

1. spaCy
2. gensim

Task 3: Explore known analogy between words - royalty

Task 4: Exploring another analogy - adjectives in positive and comparative forms

Tasks 5-6: Plot words visually with two dimensionality reduction methods

Task 1.1- Load word vectors to explore similarity -- spaCy

```
>>> import spacy
>>> nlp = spacy.load('en_vectors_glove_md') *
```

The entry point into spaCy, to construct a language processing pipeline ***nlp***: by default an instance of class `spacy.language.English` with a series of arguments

```
>>> doc = nlp("Universities and colleges are similar. Cats and cars aren't.")
```

doc is a class containing linguistic annotations, with attributes and functions https://spacy.io/docs/api/doc

```
>>> type(doc)
<class 'spacy.tokens.doc.Doc'>
```

```
>>> len(doc)
```

are, *n't*, *punctuation* own tokens.

```
12
```

* In spaCy's current version, this particular model needs to be downloaded and included as an argument. It is not the default. See details at https://spacy.io/docs/usage/models. Code snippet source: Derived from https://spacy.io/docs/usage/lightning-tour.

Task 1.2 - Compare semantic similarity -- spaCy

```
>>> universities = doc[0]
# ‘universities’

>>> colleges = doc[2]
# ‘colleges’

>>> universities.similarity(colleges)
0.87026231326132808
```

```
>>> cats = doc[6]
# ‘cats’

>>> cars = doc[8]
# ‘cars’

>>> cats.similarity(cars)
0.27139370651158923
```

Reflect on these questions:

- Which pair is more similar in meaning?
- How would you characterize the semantic relationship between the more similar pair?
- Next, explore pairs of words reflecting two or three other kinds of lexical similarities or relations that you have learned about.

Task 2.1 - Load word vectors to explore similarity -- gensim

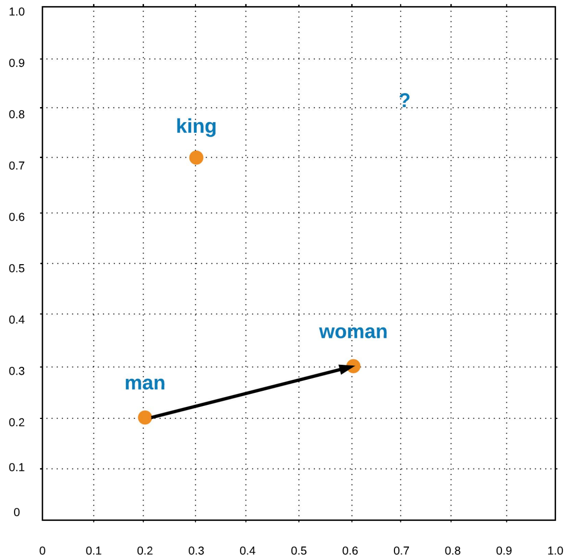
```
>>> from gensim.models.keyedvectors import KeyedVectors
Using TensorFlow backend.

>>> model =
KeyedVectors.load_word2vec_format('/usr/local/vectors/GoogleNews-vectors-negative300.bin',
binary=True)

>>> model.wv.similarity('universities', 'colleges')
0.7495764848017803

>>> model.wv.similarity('cats', 'cars')
0.23900522892985537
```

- Try the above and also go back to your previous examples.
- Do different word vectors produce similar or different similarity scores?
- Why is this so?
- How may this impact your choice of pre-trained embedding in a project?

Task 3.1 - Exploring known analogy -- gensim

Man : Woman = King : **?**

- Linear relationships (examined by Mikolov et al.)
- **King** - **Man** + **Woman** = ?




```
>>> model.wv.most_similar(positive=['woman', 'king'], negative=['man'])

[('queen', 0.7118192315101624),
('monarch', 0.6189674139022827),
('princess', 0.5902431011199951),
('crown_prince', 0.5499460697174072),
('prince', 0.5377321839332581),
('kings', 0.5236844420433044),
('Queen_Consort', 0.5235946178436279),
('queens', 0.5181134343147278),
('sultan', 0.5098593235015869),
('monarchy', 0.5087412595748901)]
```

Words provided add to or subtract from similarity to target concept.

Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.
Figure modified and adapted from http://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf

Task 3.2 - Return to spaCy for exploring known analogy

Man : Woman = King : **?**

Code in the script:
task3.2_word_analogies2171.py

Use code snippets since the model is in memory.

(As needed, run the script.)

```python
import spacy, numpy

def similarity(wv1, wv2):

    if (numpy.linalg.norm(wv1) == 0) or (numpy.linalg.norm(wv2) == 0):
        return 0.0
    return numpy.dot(wv1, wv2) / numpy.linalg.norm(wv1) * numpy.linalg.norm(wv2)

def main():

    nlp = spacy.load("en_vectors_glove_md")

    king = nlp.vocab['king']
    man = nlp.vocab['man']
    woman = nlp.vocab['woman']

    result = king.vector - man.vector + woman.vector

    words_by_similarity = []

    for w in nlp.vocab:
        if (w.has_vector) and (w.orth_.islower()) and (w.lower_ not in ['man', 'woman', 'king']):
                words_by_similarity.append(w)
    words_by_similarity.sort(key = lambda w: similarity(w.vector, result), reverse=True)

    for word in words_by_similarity[:10]:
        print(word.orth_, word.similarity(nlp('king')))

if __name__ == '__main__':
    main()
```

queen 0.725261050047
prince 0.733773667281
kings 0.7876613463
princess 0.514030326244
royal 0.616881105619
throne 0.672600414528
queens 0.527981882422
monarch 0.587183089023
kingdom 0.660404570669
empress 0.420372561303

Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.
Figure modified and adapted from http://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf

Task 4 - Using word vectors to fill these blanks

Let's find the comparatives of the following adjectives using the pre-trained word vectors.
(Examples from Mikolov et al.; simplified without superlative)

- Slow : slower (: slowest)
- Strong : ? (: ?)
- Dark : ? (: ?)
- Clear : ? (: ?)
- Soft : ? (: ?)
- Short : ? (: ?)

- Use the code snippets on the next slide to get started.
- They show how to do the left hand side problem with adjectives.
- For your convenience, a commented copy of the script task4_word_analogies2171.py is available in the tutorial folder

Slow : slower = strong : ?

```
>>> import spacy
# load in the GloVe Common Crawl vectors
>>> nlp = spacy.load("en_vectors_glove_md")
>>> adjective1 = "slow"
>>> comparative1 = "slower"
>>> adjective2 = "strong"
# initialize the adjectives and comparative

>>> adj1 = nlp(adjective1)
>>> comp1 = nlp(comparative1)
>>> print(("similarity between " + adjective1 + " and " +
comparative1 + ": %s") % comp1.similarity(adj1))
>>> similarity between slow and slower: 0.784029954196
# obtain the similarity between the adjective and its comparative

>>> adj2 = nlp(adjective2)
>>> allWords = []
# gather all known words in pre-trained vectors, except the given
words
# take only the lowercased version
```

Script task4_word_analogies2171.py is among tutorial materials

```
>>> for w in nlp.vocab:
...     if w.has_vector and w.orth_.islower():
...         if w.lower_ not in [adjective1, comparative1, adjective2]:
...             allWords.append(w)
...
>>> allWords.sort(key = lambda w: w.similarity(adj2),
reverse=True)
# sort the word list by the similarity of each word against the
original adjective2
>>> topN = 2
>>> for word in allWords[:topN]:
...     print(word.orth_)
...     print(word.similarity(adj2))
>>> stronger
0.743369607861
strength
0.678621980911

>>> print(adjective1 + " : " + comparative1 + " = " + adjective2 + " :
(" + allWords[:topN][0].orth_ + ")")
>>> slow : slower = strong : (stronger)
```

Visualizing words with dimensionality reduction

t-SNE

- t-Distributed Stochastic Neighbor Embedding
- A nonlinear and supervised method
- scikit-learn implementation is used http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
- Further reading https://lvdmaaten.github.io/tsne/ http://jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf

PCA

- Principal Component Analysis
- An unsupervised method to produce linearly uncorrelated variables (principal components)
- scikit-learn implementation is used http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
- Further reading https://www.utdallas.edu/~herve/abdi-awPCA2010.pdf

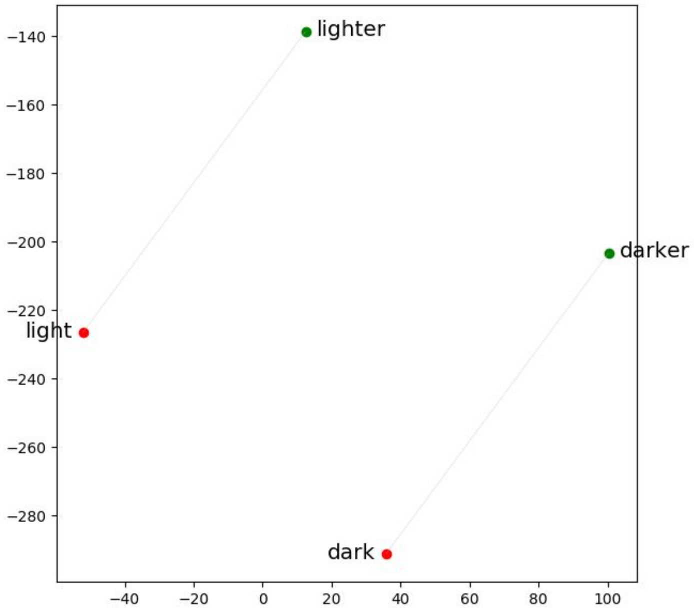
Task 5 - Produce visualizations of word vectors in gensim (Google News) with 2 methods: t-SNE and PCA

```
>>> import gensim
>>> import matplotlib.pyplot as plt
>>> from sklearn.decomposition import PCA
>>> from sklearn.manifold import TSNE

>>> trained_model = gensim.models.KeyedVectors.load_word2vec_format('/usr/local/vectors/GoogleNews-vectors-negative300.bin', binary=True)
# Using TensorFlow backend.

>>> words = ['light', 'lighter', 'dark', 'darker']
# Pairs of adjectives and their comparative forms

>>> draw_words(trained_model, words, False, True, True, -3, 3, -2, 2.2, 'word_embeddings_gensim_tSNE.png')
# Using t-SNE

>>> draw_words(trained_model, words, True, True, True, -3, 3, -2, 2.2, 'word_embeddings_gensim_PCA.png')
# Using PCA
```

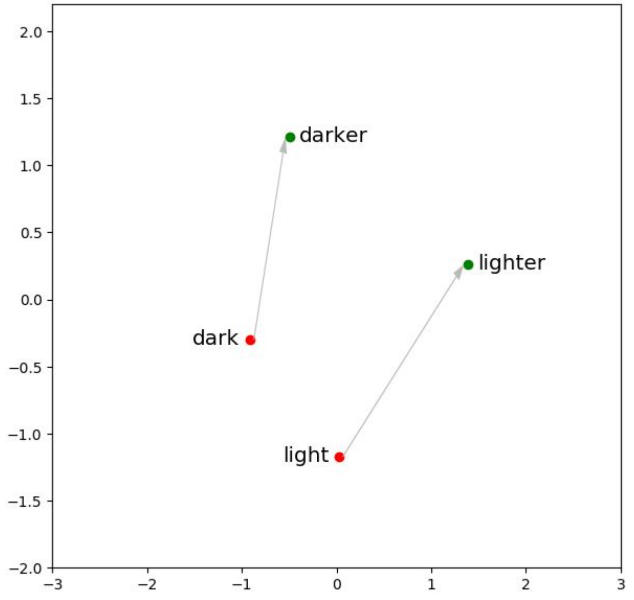
Plotting script task5_embedding_visualize_gensim2171.py is among the tutorial material (based on Mueller, Egger, with revision)

Task 5.1 - Visualized word vectors (light-lighter, dark-darker)

t-SNE



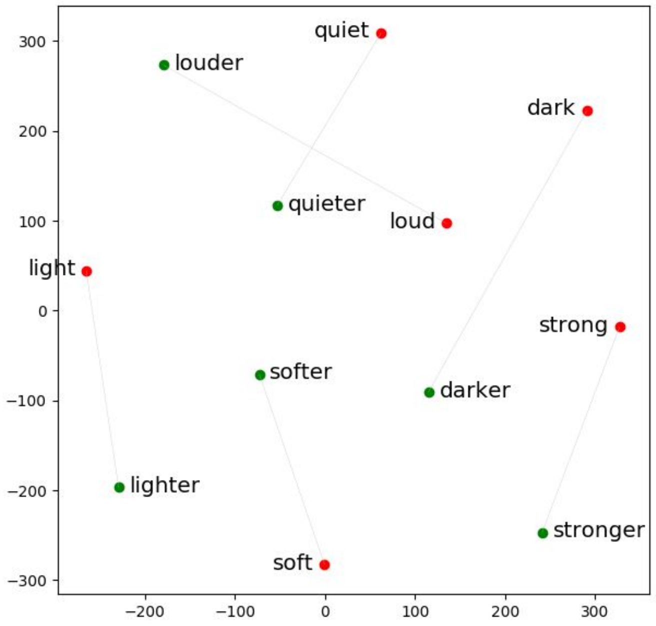
PCA



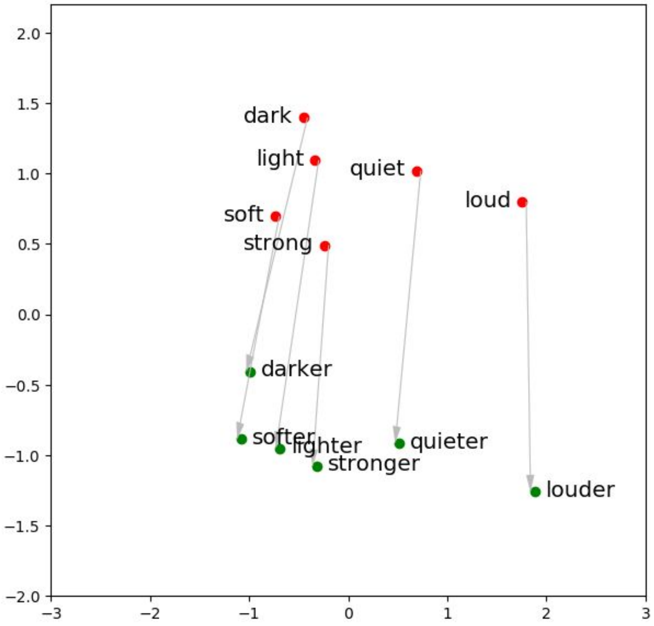
Are the relationships systematic within a plot?

Task 5.2 - Visualize additional word vectors (more adjectives)

t-SNE



PCA




```
>>> words = ['quiet', 'quieter', 'loud', 'louder', 'light', 'lighter', 'dark', 'darker', 'soft', 'softer', 'strong', 'stronger']
# Pairs of adjectives and their comparative forms - go back to 6.1 for the rest of the code.
```

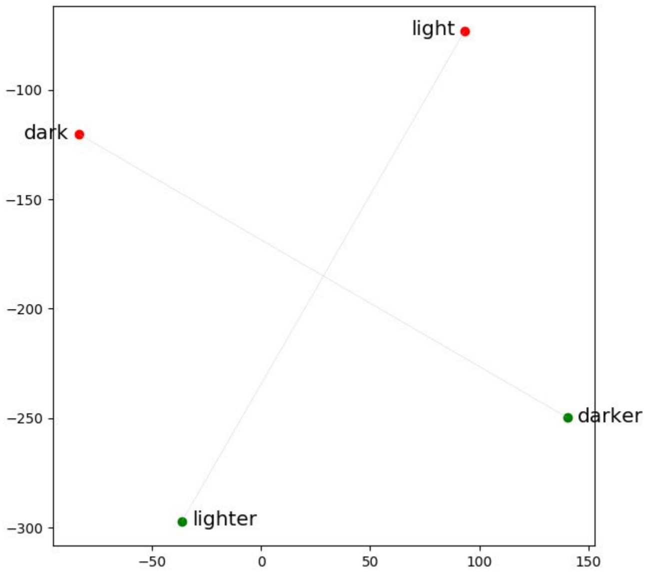
Task 6 - Produce visualizations of word vectors in spaCy (Common Crawl) with 2 methods: t-SNE & PCA

```
>>> import spacy
>>> import matplotlib.pyplot as plt
>>> from sklearn.decomposition import PCA
>>> from sklearn.manifold import TSNE

>>> trained_model = spacy.load('en_vectors_glove_md')

>>> words = ['light', 'lighter', 'dark', 'darker']
# Pairs of adjectives and their comparative forms


>>> draw_words(trained_model, words, False, True, True, -3, 3, -2, 2.2, 'word_embeddings_spaCy_tSNE.png')
# Using t-SNE

>>> draw_words(trained_model, words, True, True, True, -3, 3, -2, 2.2, 'word_embeddings_spaCy_PCA.png')
# Using PCA
```

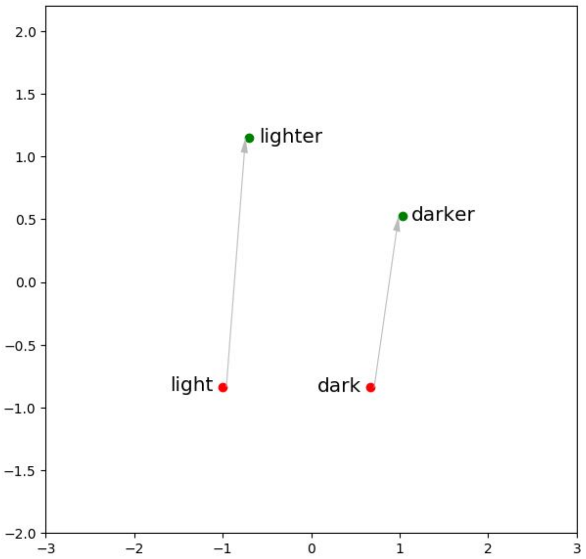
Plotting script task6_embedding_visualize_spacy2171.py is among the tutorial material (based on Mueller, Egger, with revision)

Task 6.1 - Visualized word vectors (light-lighter, dark-darker)

t-SNE



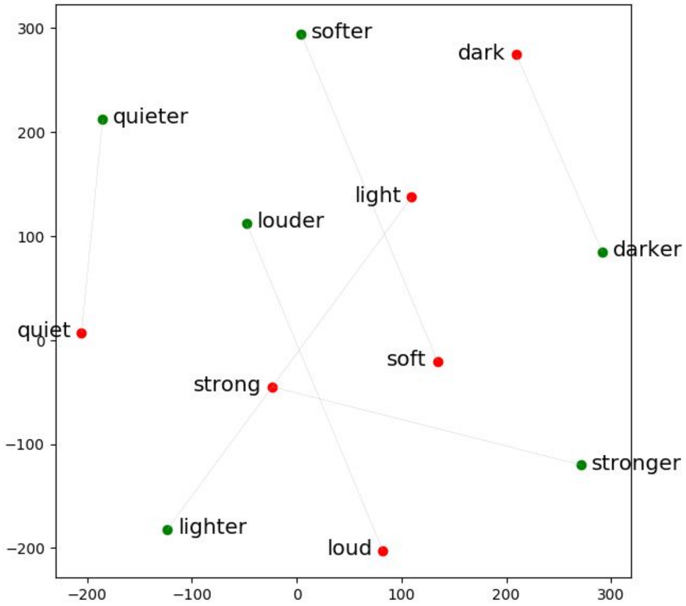
PCA



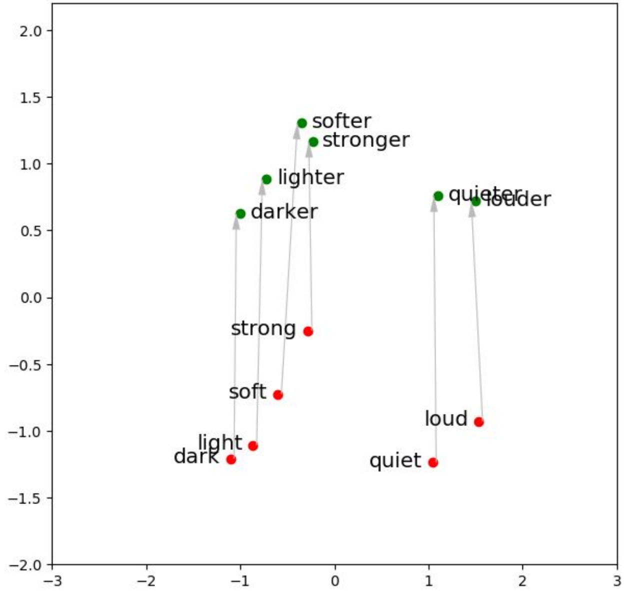
What do you note about how relations are captured based on training with different data and procedures? Go back to the 4.1 plots - how do they correspond?

Task 6.2 - Visualize additional word vectors (more adjectives)

t-SNE



PCA




```
>>> words = ['quiet', 'quieter', 'loud', 'louder', 'light', 'lighter', 'dark', 'darker', 'soft', 'softer', 'strong', 'stronger']
# Pairs of adjectives and their comparative forms
```

Consider: What is your impression regarding PCA or t-SNE based reductions for visualization?