Semantic Space Models Practical

In this practical, you will use a pre-trained word2vec model (download [1] in advance, and read [2]). Note that it has been trained on a news corpus, so might have some strange associations. You will turn in your code and a brief (~2pg) report.

Part 1. The goal is to create algorithms to play the game 'Password' in what you think is a human-like way. See how the game is played:
https://youtu.be/FSQMbsD9Llo?t=1m52s

1.1 Play the game a few times with your partner; try both roles. Think about how you choose what word to send. Is it based on meaning? Word order? What makes a password easy or hard to guess? Write how you think people (should) play this game, as concretely yet concisely as you can.

1.2 Test word2vec: use the pre-trained Google vectors and start with examples in `password.py`.

1.3 Using word2vec implement two functions: `send_word(secret)` plays as the sender (i.e., returns a word that is most likely to cause the receiver to guess `secret`); and `receive_word(hint)` plays as the receiver (i.e., returns the word the receiver believes is the password). Include comments summarizing how your functions work.

1.4 Test your sender algorithm on the following words: cut, ice, stamp, self, snail, now, bed, night, needle, scratch, bank, joke, king, salt, good, washer, east, nail, bulb, lost
Which hints seem reasonable? Why? Which seem difficult? Why?

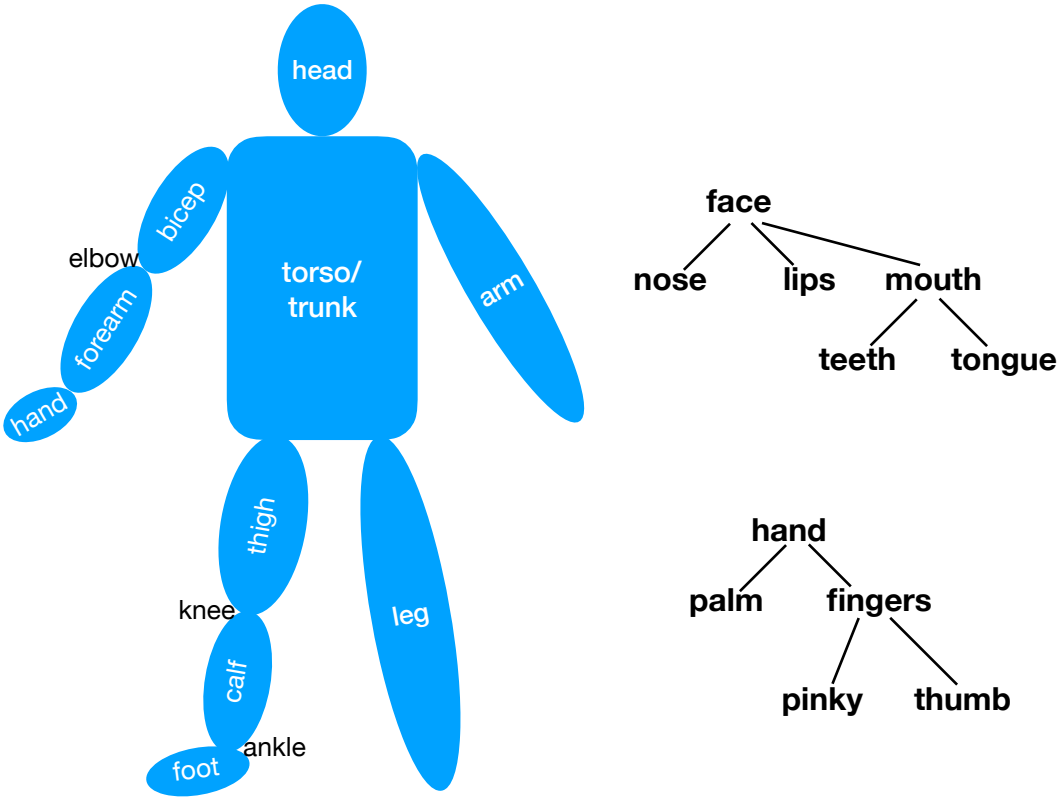
1.5 Test your receiver algorithm on each of the hints returned from your sender algorithm in 1.4. Which passwords do you manage to recover? Which were close? Which were not even close? Why?

[1] Download pre-trained word2vec vectors:
https://code.google.com/archive/p/word2vec/
(find `GoogleNews-vectors-negative300.bin`)

[2] A good explanation of word2vec:
http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
(also see part 2, which covers subsampling of frequent words and the Negative Sampling technique)
If you'd rather watch a lecture: https://youtu.be/ERibwqs9p38?t=18m35s (through 42:30)

A nice walk-through of an application of word2vec (and bag-of-words) to sentiment analysis including preprocessing steps that may be taken when cleaning text corpora:
https://www.kaggle.com/c/word2vec-nlp-tutorial/
Refer to if you haven't used python to do any NLP problems before.

Part 2. Your goal is to see how closely the representations of body part names in word2vec (and LSA or LDA) match the actual hierarchy of body parts.



2.1 Create a hierarchical clustering that represents the actual hierarchy of body parts, based on the 25 names in the figure above. Hint: *First draw the full hierarchy, then create a pairwise similarity matrix, starting by filling in the similarity of the lowest leaves with very high values. Finally, use* sklearn.cluster.AgglomerativeClustering.

2.2 Find the pairwise similarity of each body part, according to word2vec, and create a hierarchical clustering from this similarity matrix.

2.3 Compare the model's clustering to the real body part hierarchy you defined in 2.1. (Hint: *Use Adjusted Rand Index*.) How well does the model capture the hierarchy? Describe which parts it fits well, and which parts it misses. Is there an apparent explanation for what is hard and what is easy to capture?

2.4 Repeat 2.2 and 2.3 for LSA or LDA (see `LSA_and_LDA.py` for a start). Which of the models you evaluated provides the best fit to the real body part hierarchy?

2.5 Which other body parts can you capture? Consider: wrist, neck, toes, shoulders, belly, back, eyes, ears, eyebrows, cheeks, chin, and forehead. Can you say something general about which parts are captured by the model(s), and which are not? E.g., is it related to size? Leaf vs. non-terminal node? Polysemous words?