Traitement automatique du langage
TP 6 — Distributional semantics

Yves Scherrer & Aurélie Herbelot

1 Building your first semantic space

1. Describe what kind of words end up at the top of the obtained distributions, and what kind of nearest neighbours are returned.

   - The most characteristic contexts are words that most often occur in the same sentence as *Queen*, for example adjectives (*furious*), verbs (*roar*) and adverbs (*furiously*).
   - The nearest neighbours are words that are most similar to *Queen* (i.e. which could appear in the place of *Queen* without changing the meaning of the sentence), such as *King, player, Frog-Footman*, most of which are nouns. Note however that the results are not very reliable due to the small corpus size.

2. How do the characteristic contexts and nearest neighbours change if you modify the number of columns and rows in the semantic space?

   - Reducing the number of columns yields a more generic, compact space. This may have a positive or a negative effect: positive because nearest-neighbours methods tend to work better if the number of columns is not too large, and negative because the distance computations become less precise as information is missing.
   - Reducing the number of lines just amounts to reducing the size of the vocabulary and thus only will have an impact on the nearest neighbours, where some candidates may disappear.

3. What changes when you increase the size of the word window?

    - Increasing the window size results in considering more context words for every target word. These additional context words may be useful, but they are generally less specific because they are less intrinsically tied to the target word. The size of the context window may also depend on the language of the corpus.

4. What changes when using untagged data?

    - The provided script only keeps those context words that are tagged as verbs, nouns, adjectives or adverbs. This filter removes a lot of noise, which is particularly important when using a small corpus. If the tagging step is removed, the filter cannot be applied, leaving all function words (determiners, prepositions, pronouns, punctuation signs etc.) in the text. If the corpus is sufficiently large, these words will likely be downranked, but with a small corpus they may add a lot of noise.
    - In addition, tagged data allows the model to distinguish homonyms with different functions, but this aspect is of minor importance here.

2 Investigating a large semantic space

Combine the hyponymy and nearest neighbours code to produce a system which returns the likely hypernyms of a word.

```
from composes.utils import io_utils
from composes.similarity.cos import CosSimilarity
import sys, math

# from kneighbours.py
def getNeighbours(space, word, n):
    my_space = io_utils.load(space + ”.pkl”)
    neighbours = my_space.get_neighbours(word, n, CosSimilarity
       ())
    return neighbours

# from hyponymy.py
def loadHyponymyTable(space):
    DM=open(space + ”.dm”)
    dic={}
    for l in DM:
        pair=l.rstrip(”\n”).split(’\t’)
        dic[pair[0]]=pair[1:]
    return dic
```

```
# from hyponymy.py
def computeHyponymy(dic, word1, word2):
    h1=dic[word1]
    h2=dic[word2]
    iter=0
    sum_mins=0
    sum_h1=0
    sum_h2=0
    for weight1 in h1:
        weight2=h2[iter]
        sum_mins+=min(float(weight1),float(weight2))
        sum_h1+=float(weight1)
        sum_h2+=float(weight2)
        iter+=1
    clarkeDE=sum_mins/sum_h1
    clarkeDEinv=sum_mins/sum_h2
    invCL=math.sqrt(clarkeDE*(1-clarkeDEinv))
    return invCL

def getHypernyms(space, word):
    neighbours = getNeighbours(space, word, 50)
    i = 1
    print ”--- Neighbours ---”
    for w, v in neighbours:
        print i, w, v
        i += 1
    hypTable = loadHyponymyTable(space)
    invCLresults = {}
    for neighbourWord, neighbourValue in neighbours:
        if neighbourWord == word:
            continue
        invCLresults[neighbourWord] = computeHyponymy(hypTable,
            word, neighbourWord)
    bestHypernymsInv = sorted(invCLresults.items(), key=lambda
       x: x[1], reverse=True)
    i = 1
    print ”--- Hypernyms ---”
    for w, v in bestHypernymsInv:
        print i, w, v
        i += 1

if __name__ == ”__main__”:
    getHypernyms(sys.argv[1], sys.argv[2])
```

The script can be run as follows:

```
python getHypernyms.py ../spaces/wikipedia cat
```

Note that the extension of the space is not required – the script will use both the `.pkl` and `.dm` files.

This method does not work particularly well because the space is not large enough. For example, the best hypernyms for the word *cat* are *stray, gorilla, donky, crow* and *ape*. The words *animal* and *pet* are at positions 14 and 17, the word *feline* is not available in the space. Compared to the simple neighbours, this is an improvement for *animal* (moved from position 39 to 14), but a slight degradation for *pet* (moved from position 12 to 17).