Chapter 5

Extracting Collocations

For certain definitions of collocation, a polysemous word exhibits essentially only one sense per collocation. —David Yaraowsky (1992)

5.1 What are Collocations and How to Extract Them

A collocation is a pair or a sequence of words that co-occur more often than by chance and also has certain types of syntactical relations. There are about six main types of collocations:

- adjective+noun (e.g., *strong* **wind**, *middle* **management**, *nuclear* **family**)
- noun+noun (e.g., *traffic* **light**, *motor* **cyclist**)
- verb+noun (e.g., *play* a **role**)
- adverb+adjective (e.g., *simply* **beautiful**)
- noun+adjective (e.g., *crystal* **clear**, *pigeon* **gray**)
- verbs+prepositional phrase (e,g., **comment** *on the issue*)
- verb+adverb (e.g., **prove** *true*).

For these types of collocation, we typically identify a word as head (bold-face) and other words as collocates (italic).

Second language learners tend to chose wrong words (e.g., *strong* instead of *heavy*) to match other words (e.g., *rain*). Even though *strong rain* could convey roughly the same meaning as *heavy rain* (a good collocation), this mis-collocation is considered awkward by native speakers. In other words, good command of collocations is a hallmark of native speakers. Conversely, the corresponding expression in technology, powerful computer is preferred over strong computer.

For Natural Language Processing, collocates of a head word can provide very useful insight into the meaning and usage of the head word. For language learning as well as for NLP, it is useful to develop computational technique to automatically extract collocations from a corpus.

The traditional methods for extracting collocations rely on a statistical association measures. These formulas include mutual information, t-test, z test, chi-squared test and likelihood ratio.

5.2 XTract–Smadja Algorithm for Extracting Collocations

Smadja (1993) proposes a method for extracting collocations from a large corpus. These method is based on some association measure as well as the positional distribution of headword and collocates. Smadja (1993) also describes a system XTract as an implementation of the proposed method Evaluation, based on a 10 million-word corpus of stock market news reports, showed that Xtract retrieve collocations effectively with an estimated precision rate of 80

The method proceeds in three stages:

1. Compute the ngrams in $CORP$ to obtain $\{(w_1, .., w_n, freq(w_1, .., w_n)) \| n \in [2, 6]\}$

2. Set up *SkipGram*, a hash table (*dictionary*) for storing skipped bigrams. Use two levels of key (the first level for *headword* and the second level for *collocate*). Each key is associated with a $Counter, P$ to hold the positional counts. For example, the $Counter$ for $(play, role)$ looks like { -5:5 }

3. For each ngram entry (e.g.,), update P for the bigram $\{(w_1, w_n)$ by adding $freq(w_1, .., w_n)$ to position n, as well as updating P for the bigram $\{(w_n, w_1)$ for the position $-n$

4. Filter the collocates:
   - average co-occurring frequency $\bar{f} = \sum_{0<i<=N} freq_i \ / \ N$
   - standard deviation $\sigma_f = \sum (freq_i - \bar{f})^2 \ / \ N$
   - strength of w_i, $k_i = (freq_i - \bar{freq})/\sigma > \lambda_1$ (Condition 1)
   - spread of all $p_i^j \quad (j \in [-5, 5] \ )$

5. Filter the positions for remaining w_i:
   - average positional co-occurring frequency $\bar{p}_i = \sum_{j \in [-5,5]} p_i^j \ / \ 10$
   - positional spread $V_i = \sum_{j \in [-5,5]} (p_i^j - \bar{p}_i)^2 \ / \ 10 > \lambda_2$ (Condition 2)
   - peak frequency of j such that $p_i^j > \bar{p}_i + \lambda_3 \sqrt{V_i}$ (Condition 3)

Smadja (1993) suggested using the setting of $(\lambda_1, \lambda_2, \lambda_3)$ = (1, 10, 1) for better results. Obviously, you can tune the parameters to get a good trade of between precision and recall. In other words, if you lower the values in $(\lambda_1, \lambda_2, \lambda_3)$, you will get more collocations but the percentage of false collocations will also increases.

5.3 Work Sheet

Write a program to extract collocations for words in the Academic Keyword List (ref), based roughly on Smadja (1993). For this, you will be given the follow datasets and seed program:

- A **Ngram dataset** (**citeseerx.23456.gz**) contain ngrams related to AKL in the *Citeseer^x*,

  a big-data Academic Corpus of 440 million words (ref).

```
$ zgrep 'role-n' citeseerx.ngms.gz | sort -k2nr -t $'\t'| head -20
role-n in-prep 62342
important-adj role-n 23675
important-adj role-n in-prep 17206
play-v an-det important-adj role-n 15863
role-n for-prep 8790
play-v a-det role-n 6715
key-adj role-n 5595
crucial-adj role-n 4615
central-adj role-n 4597
play-v a-det key-adj role-n 3961
critical-adj role-n 3658
major-adj role-n 3257
play-v a-det crucial-adj role-n 3254
central-adj role-n in-prep 3128
crucial-adj role-n in-prep 3110
on-prep the-det role-n 3067
play-v a-det central-adj role-n 3015
significant-adj role-n 2994
investigate-v the-det role-n 2612
critical-adj role-n in-prep 2567

$ zgrep -E '(^role-n| role-n\t)' citeseerx.ngms.gz | sort -k2nr -t $'\t'| head -40 | sort
central-adj role-n 4597
critical-adj role-n 3658
crucial-adj role-n 4615
discuss-v the-det role-n 1413
...
investigate-v the-det role-n 2612
...
play-v a-det role-n 6715
play-v a-det significant-adj role-n 2108
play-v a-det vital-adj role-n 1139
play-v an-det essential-adj role-n 1357
play-v an-det important-adj role-n 15863
play-v important-adj role-n 1677
play-v the-det role-n 1873
...
role-n as-prep 2296
role-n be-v 2293
role-n for-prep 8790
role-n in-prep 62342
role-n in-prep determine-v 1310
role-n in-prep many-adj 1188
role-n in-prep the-det development-n 1232
...
$
```

- AKL in Python (**akl.py**) contain the list of AKL words as a dictionary.

Programming Exercise

Write a Collocation-Extraction program to generate a list of collocations for each keyword in AKL. Each collocation should also associated with the most-frequent example ngram.

Hints

(1) For simplicity, focus on Conditions 1 and 3 and ignore Condition 2.

(2) Use as many existing functions and abstract data types (e.g., **defaultdict** and **Counter** in Python commonly-used **collections** module) as possible to streamline your code and avoid coding errors.

(3) Store all the example ngrams of each collocation candidate, in a similar fashion using defaultdict. For example, consider the following code snippet:

```
from collections import defaultdic, Counter
from pprint import pprint

def read_ngrams():
    # compute distance-bigram
    skipBigramDistance = defaultdict(lambda:________________ )
    skipBigramExample = defaultdict( ________________ )
    for line in sys.stdin:
        ________________
        ________________


    ________________
    return (skipBigramDistance, skipBigramExample)

if __name__ == '__main__':
    skipBigram, skipBigramExample = read_ngrams()
    ________________
```

```
$ python -i XTRACT.py
...
...
...
>>> pprint(sorted(list(skipBigram['role-n'].items()),key=lambda x: -sum(x[1].values()))[:10])
[('in-prep', Counter({1: 62342, 3: 2192, -2: 859, 4: 422, 2: 355, -3: 298, ...})),
 ('play-v', Counter({-3: 41994, -2: 13098, 1: 2523, 2: 1317, -1: 287, 3: 23, -5: 22})),
 ('important-adj', Counter({-1: 23675, -2: 142})),
 ('for-prep', Counter({1: 8790, -2: 1088, -3: 164, -1: 119, 2: 107, 3: 45, 4: 20})),
 ('be-v', Counter({1: 2293, -4: 1284, -2: 817, 2: 786, -5: 753, -3: 233, 3: 71, -1: 38, 4: 24})),
 ('have-v', Counter({-3: 2624, -4: 1430, -2: 1299, 1: 375, -5: 136, -1: 106, 2: 42})),
 ('key-adj', Counter({-1: 5595, 2: 32})),
 ('central-adj', Counter({-1: 4597, 2: 59, 3: 57})),
 ('crucial-adj', Counter({-1: 4615})),
 ('we-pron', Counter({-3: 3690, -4: 727, 2: 104, 1: 42}))]

>>> print(sorted(list(SKIPGRAM['role-n']['play-v'].items()))
[(-5, 22), (-4, 1687), (-3, 41994), (-2, 13098), (-1, 287), (1, 2523), (2, 1317), (3, 23)]

>>> print(sorted(list(SKIPGRAM['play-v']['role-n'].items()))
[(-3, 23), (-2, 1317), (-1, 2523), (1, 287), (2, 13098), (3, 41994), (4, 1687), (5, 22)]

>>>
```

(4) Additionally, in Smadja Algorithm, $Condition 1$ and 3 seem to be the same computation on different kinds of list. So, write a generic **filterCounts** that can be used to enforce both conditions and retain high-count item in the list:
(Note: k is used to denote λ_1 and λ_3 filtering parameter for collocates and positions respectively.)

```
def filterCounts(list_with_counts, getKey, k):
```

```
# getKey = function returning the count in list_with_counts[i]


____________________________________
____________________________________
return __________________
```

References

1. Daniel Jurafsky and James H. Martin. 2017. Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 3rd Edition. Chapter 17. Computing with Word Senses. (draft available at `https://web.stanford.edu/~jurafsky/slp3/`
2. Smadja, Frank. "Retrieving collocations from text: Xtract." Computational linguistics 19.1 (1993): 143-177.
3. Paquot, Magali. "Towards a productively-oriented academic word list." (2007). The AKL data is available at `https://uclouvain.be/en/research-institutes/ilc/cecl/academic-keyword-list.html`
4. Caragea, Cornelia, et al. "Citeseer x: A scholarly big dataset." European Conference on Information Retrieval. Springer, Cham, 2014. The dataset is available at `http://csxstatic.ist.psu.edu/about/data`
5. David Yarowsky, One sense per collocation, Proceedings of the workshop on Human Language Technology, March 21-24, 1993.