# Generating sentences from Part-of-Speech patterns using collocation fields

Bogdan Babych [b.babych@leeds.ac.uk](mailto:b.babych@leeds.ac.uk) March 4, 2019

The task of natural language generation for advertising headlines from in current implementation (<https://github.com/bogdanbabych/colloc4nlg>) relies on

1. mapping identified keywords into PoS templates and
2. further generating a sequence of collocations for each position in the sequence filtered by the PoS

This document describes the current algorithm for generating collocations and a suggested more systematic algorithm.

## Initial implementation of the algorithm

In the initial implementation every keyword is mapped into a single PoS template, which now is either the most frequent template for a given keyword in Canadian 2018 advertising corpus, or, if the keyword is not present in that corpus – as the most frequent template in the British National Corpus (BNC), e.g.:

Word ‘*deal’* is mapped into the PoS template:

*WP V.\* J.\* IN DT J.\* !deal/NN*

The position of the keyword in the template is prefixed with the exclamation mark “!”. Lexical item in the template is separated from its PoS code by a slash “/”.

PoS templates may be further edited. Specifically, they now support syntax for excluding certain collocations which generate too many noisy examples:

* The equal sign ‘=’ after a PoS code introduces the list of such excluded items;
* Each element in that list is separated by a coma “,”;
* Each collocation that needs to be excluded from the output is prefixed with a semicolon “;”.

The example of an edited template:

WP V.\* J.\*=;valid,;redeemable,;in-store,;safe IN=:to,:from DT J.\* !deal/NN

Now templates are edited manually, but in further stages they may be useful for automated improvement of the output when some generated examples are identified as noisy in the evaluation stage and need to exclude specific words from the output.

Initial algorithm implemented in the system is presented below:

# Stage 1: generating collocations

Set **Current Position** = Keyword Position (“!” prefix in the template).

Set **Current Word** = Keyword in the template

Set **Output Collocation List of Lists** = [[Current Word]]

Set **PoS List** = right-to-left list of PoS codes in the template

(from the **Current Position** to the start of the template).

For each **Current PoS** code in **PoS List**:

**Collocation List** = a list of left collocations of the **Current Word** in 2-word window,

- filtered by the **Current PoS**

- ranked by a chosen collocation score (Log Likelihood score is the default).

Create a new position in front of the **Output Collocation List of Lists**

Save **Collocation List** in the new position of the **Output Collocation List of Lists**.

**Current Word** = top word in the **Collocation List**

Next **Current PoS**

# Stage 2: recombining collocations

**Output List of Sentences =** cartesian\_product(**Output Collocation List of Lists**)

- ranked by sum of log(collocation scores for each position)

Limitations of the initial algorithm as described above:

1. Only an immediate context is used to generate collocations for any given position, as a result, sentences are linked only via bi-grams (1st-order Markov models).
2. Only the top word in each collocation list is used to generate collocations for the next neighboring position, rather than N-best list.
3. Only one direction (from the keyword to the left) is currently implemented, so a keyword needs to be last in the generated N-gram sequence.

Visually the initial algorithm can be presented as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PoS1** | 2(top)>1 |  |  |  |  |
|  | **PoS2** | 3(top)>2 |  |  |  |
|  |  | **PoS3** | 3<kw |  |  |
|  |  |  | **kW** |  |  |
|  |  |  |  | **PoS5** |  |
|  |  |  |  |  | **PoS6** |

(The matrix shows PoS codes in the template positions on the diagonal, and the lexical item which is used to generate collocates for a given positions is shown in a given column and row, respectively).

It can be seen that the algorithm now implements just a single chain of collocation generation, going from left to right starting from the keyword.

## Modifications to the initial algorithm

To overcome these limitations, the new algorithm will generate a field of collocations, where for each position, collocations will be generated from:

* the keyword, then from
* the N-best list of collocates generated for the keyword for other positions in the sentence
* Scores for ranking lists of collocates for each position will be weighted by the distance between the PoS position and the lexical item for which collocations are generated (the keyword or an item from the N-best list).The weight function should be sufficiently high for the immediate context, but decline for more distant collocations, e.g.:

Visually the suggested algorithm can be presented as follows:

Stage 1: populating PoS positions in a template from the keyword with ranked collocation lists:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PoS1** |  |  | 1<kw |  |  |
|  | **PoS2** |  | 2<kw |  |  |
|  |  | **PoS3** | 3<kw |  |  |
|  |  |  | **kW** |  |  |
|  |  |  | kw>5 | **PoS5** |  |
|  |  |  | kw>6 |  | **PoS6** |

* For the keyword, a ranked list of collocates is generated for each position in the template, with collocation scores weighted by a distance function.
* An N-best list is selected for each position (top N collocates). Usually there will be N>4.

Stage 2: creating a collocation field for the whole matrix from the N-best lists

For each position, for each element in the ranked N-best list the same procedure is repeated as with the keyword in Stage 1: collocation lists are created for other positions:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PoS1** | 1<2 | 1<3 |  | 1<5 | 1<6 |
| 1>2 | **PoS2** | 2<3 |  | 2<5 | 2<6 |
| 1>3 | 2>3 | **PoS 3** |  | 3<5 | 3<6 |
|  |  |  | **kW** |  |  |
| 1>5 | 2>5 | 3>5 |  | **PoS5** | 5<6 |
| 1>6 | 2>6 | 3>6 |  | 5>6 | **PoS 6** |

Collocation lists are updated, if the same word is returned as a collocate of several words, the (weighted) collocation scores are added.

Stage 3. The update process is repeated with the updated N-best collocation lists k times.

Stage 4. Finally, the output sentences are generated similarly to the initial algorithm from the ranked lists of collocations.

Suggested modification of the algorithm is still based on bi-gram model, but since bigrams are mutually reinforcing each other, the output should emulate a higher order Markov model.

Further improvement may be achieved if generated sequences are checked (and re-ranked) using a pre-compiled database of skip-grams for a given corpus.