Colibri Core

Maarten van Gompel
Centre for Language Studies
Radboud University Nijmegen

August 28th, 2014

Introduction

What is colibri-core?

- Software for the **extraction**, **modelling** and **querying** of patterns aka constructions (n-grams, skip-grams, flex-grams)
- Low-level (hence the "core"), more specific software can be built on top of it
- Highly configurable, many parameters
- Designed with memory and speed efficiency in mind, but memory-based (lossless)
- Binary data formats (quick loading, smaller files)

Interfaces

Three interfaces:

- command-line tools (colibri-patternmodeller, colibri-classencode, colibri-classdecode)
- C++ API
- Python API (binding with native code, written in cython)

Patterns

Types of patterns

1. N-grams: to be or not to be
2. Skip-grams (one or more fixed-sized gaps): to be *2* to be
3. Flex-grams (one or more gaps of flexibile/undetermined length): to be * to be

Input

1. Input: plain-text (utf-8), one "entity" per line (sentence/paragraph/tweet/document), basic conversion from FoLiA supported too.
2. Input first has to be class-encoded: `colibri-classencode`, `ClassEncoder`.
3. Output has to be class-decoded: `colibri-classdecode`, `ClassDecoder`

Compression through class encoding

- All word types in a corpus are ranked (frequent to rare)
- Each word is encoded by its rank/class number
- Variable-width encoding: Frequent words (high up in the ranking) take less bytes to represent than rare word types
- Results in roughly 50% compression
- Integer comparison (by token) much quicker than string comparison (by character)
- On large data, Colibri-core will be faster and smaller than naive string-based representation
- Stored on disk in binary format for quick loading
- Caveat: Corpora are only comparable if they use the same class encoding!

Extraction & Modelling

Extraction

- What patterns are in a corpus?
- How many times do they occur?
- Where do they occur in the corpus? (= Indexed modelling)
- Extraction constrained by another model (train/test paradigm)

Result of such extraction is a **Pattern Model**

Modelling

- How many patterns are there?
  - for a specific pattern
  - of a particular pattern type and/or length?
  - how much of the corpus is covered?
  - basic statistics
- How do patterns relate to others?
  - What patterns subsume others? (parent/child relations) What fits in a gap of a skipgram?
  - What patterns follow each-other in the text?
  - What patterns co-occur, how often? (Jaccard, nPMI)
- How do two corpora compare?
  - How many/which patterns of the training corpus occur in the test corpus as well?
- Reverse indexing
  - What patterns can be found for a certain position in the corpus?

Data structures

High-level Data structures

- `Pattern`: Encoded representation of a pattern, the core building block
- `PatternModel`: *pattern* $\mapsto$ *occurrencecount*
  - `IndexedPatternModel` *pattern* $\mapsto$ *indexreferences*
- `PatternAlignmentModel`: $pattern_1 \mapsto pattern_2 \mapsto value(s)$

Low-level Data structures

C++ library is set up in such a way (templating) that you can exchange the underlying data structure whilst maintaining the same interface

- `PatternStore` *(abstract)*
  - `PatternMap`: Hash-map *(C++11 unordered_map)*: *pattern* $\mapsto$ *value*(*s*)
  - `PatternSet`: Set *(C++11 unordered_set)*
- `IndexedCorpus`: *indexreference* $\mapsto$ *unigram*
  - . . . as **Reverse Index** with a Pattern Model: *indexreference* $\mapsto$ *patterns*

Algorithms

Algorithms

N-gram extraction with threshold > 1: Iterative with sub-n-gram checks to conserve memory, multiple passes over corpus

```
def countngrams(model, corpus, threshold):
  for n in 1...maxn:
    for line in corpus:
      for ngram in getngrams(line,n):
        include = True
        for subngram in getngrams(ngram,n-1):
          if model[subngram] < threshold:
            include = False; break
        if include:
          model[ngram] += 1
```

Algorithms

Skip-gram extraction with threshold > 1

```
def extractskipgrams(model,corpus, threshold, ngram, n)
    :
  for skipgram in allpossibleskipgrams(ngram):
    include = False
    for subngram in consecutiveparts(parts)
      if model[subngram] < threshold:
        include = False; break
    if include:
      model[skipgram] += 1
```

Algorithms

Flex-gram extraction

1. Generalisation over skipgrams: Are there skipgrams which differ only in length of the gaps?
2. Directly from co-occurrence: Compute co-occurence, co-occuring patterns are flexgrams (preserving order)

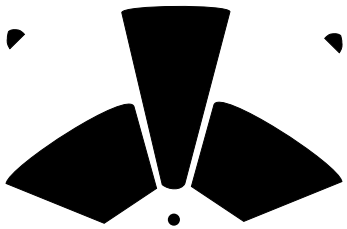
Algorithms

Relation extraction

1. Given a pattern,
2. find all the indexes where it occurs (forward index),
3. find all other patterns occuring at or near these indexes, depending relation criteria (reverse index)

Open-ended

- Colibri-core is just the foundation for more specific tools
- Lots of possible use-cases
- Things built on top of colibri core:
  - **Colibri-MT**: Modelling of source-side context for Machine Translation, through classifiers
    - **Colibrita**: Modelling of L2 context for L1 fragments, SemEval 2014 Task 5, through classifiers
  - **cococpyp**: Nonparametric Bayesian modeling with Pitman-Yor process priors
  - **DEvents** event pair extraction



Questions?