docrep: A lightweight and efficient document representation framework

Tim Dawborn    James R. Curran
α-lab, School of Information Technologies
University of Sydney
{tim.dawborn,james.r.curran}@sydney.edu.au

COLING 2014
Corpus processing

- NLP is increasingly a data-driven research discipline
- Researchers are utilising a diverse collection of large-scale corpora
- Some key issues associated with large-scale corpus processing:
  - Representation of multiple annotation layers
  - Representation of overlapping annotation layers
  - Reproducibility
  - Scalability
Multiple annotation layers

- Flat-file representations of corpora are common
  - Easy to inspect and (often) easy to process
  - Cannot easily store multiple annotation layers
  - Cannot store nested annotation layers
- Traditional structured storage representations, such as databases, are less common
  - Harder to inspect
  - May require specialised search tools to extract information
- Document Representation Frameworks (\textsc{drf}s) aim to provide a better solution for the storage of corpora
Multiple annotation layers

- Corpora are increasingly multi-annotation layered
- E.g. The OntoNotes 5 corpus has:
  - Tokens
  - POS tags
  - Parse trees
  - Predicate constituents and their arguments
  - Word senses
  - In-document coreference
  - Named entities
  - Links to the Omega ontology
Reproducibility

- We should strive for reproducibility as researchers
- Hard to do due to compounding decisions that are made when processing corpora, especially pre-processing
  - Removing metadata and markup
  - Performing sentence-boundary detection and tokenisation
  - Thresholds and cutoffs at various stages in NLP pipeline
- Ideally the data format should promote reproducibility
  - These kinds of decisions should be in the metadata of corpora
  - Ideally this metadata can be transferred without licencing issues
Scalability

- The processing of large-scale corpora can often be done in parallel
- Ideally, the representation of your corpora and its annotation layers should support parallel processing
- Stream processing paradigm suits this problem well
  - Streams (corpora) of discrete units of data (documents)
  - All units need to be processed
  - Units can be processed independently from one another
  - Results can be easily joined back together
What is docrep?

- docrep aims to solve these problems, while being:
  - light weight
  - easy to use
  - compact
  - fast
- docrep is a programming language agnostic document serialisation format
- We have provided docrep APIs in C++, Python, and Java
- Available from https://github.com/schwa-lab

Tim Dawborn, James R. Curran
docrep: A lightweight and efficient document representation framework

Coling 2014
Why should you use docrep?

- No overhead after writing your Annotation classes
- Command-line friendly – no IDE required
- Streaming representation – cat files together
- As compact and efficient as what Google uses (protobufs)
  - But we also support pointers!
  - And our documents are self describing – no schema file needed!

<table>
<thead>
<tr>
<th></th>
<th>Self-describing</th>
<th>Uncompressed Time</th>
<th>Uncompressed Size</th>
<th>DEFLATE Time</th>
<th>DEFLATE Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original data</td>
<td>-</td>
<td>-</td>
<td>31.30</td>
<td>1.0</td>
<td>5.95</td>
</tr>
<tr>
<td>BSON</td>
<td>✓</td>
<td>2.5</td>
<td>188.42</td>
<td>5.3</td>
<td>30.32</td>
</tr>
<tr>
<td><strong>MessagePack</strong></td>
<td>✓</td>
<td><strong>1.6</strong></td>
<td><strong>52.15</strong></td>
<td><strong>3.2</strong></td>
<td><strong>16.61</strong></td>
</tr>
<tr>
<td>Protocol Buffers</td>
<td>×</td>
<td>1.4</td>
<td>51.51</td>
<td>3.5</td>
<td>18.52</td>
</tr>
<tr>
<td>Thrift</td>
<td>×</td>
<td>1.0</td>
<td>126.12</td>
<td>3.5</td>
<td>20.64</td>
</tr>
</tbody>
</table>
What’s wrong with **UIMA** or **GATE**?

- Large, slow, and clunky
- Very Java-oriented. **UIMA** has a **C++ API**, but it:
  - is not really documented
  - does not have all of the same functionality of the Java **API**
A docrep Document contains multiple Annotation layers

Annotation instances are stored in Stores on a Document

class Token(dr.Ann):
    span = dr.Slice()
    norm = dr.Text()

class NamedEntity(dr.Ann):
    span = dr.Slice(Token)
    label = dr.Text()

class Doc(dr.Doc):
    tokens = dr.Store(Token)
    nes = dr.Store(NamedEntity)
Annotations

- Different annotation types modelled as different `Ann` subclasses.
- Most kinds of data can be stored as attributes of an `Annotation`
  - Primitive data types
  - Byte and Unicode strings
  - Pointers to other `Ann` instances on the document
  - Lists of pointers to other `Ann` instances

```python
class ParseNode(dr.Ann):
    tag = dr.Text()
    token = dr.Pointer(Token)
    parent = dr.SelfPointer()
    children = dr.SelfPointers()
    score = dr.Field()
```
Annotations

- Annotation types can be used in multiple stores on a Document
- E.g. outputs from different systems

```python
class NamedEntity(dr.Ann):
    span = dr.Slice(Token)
    label = dr.Text()

class Doc(dr.Doc):
    system_a_nes = dr.Store(NamedEntity)
    system_b_nes = dr.Store(NamedEntity)
    system_c_nes = dr.Store(NamedEntity)
```
Documents

- Documents are where the Annotations are Stored
- They can have serialised attributes as well
  - E.g. outputs from different systems

```python
class Doc(dr.Doc):
    doc_id = dr.Text()
    tokens = dr.Store(Token)
    nes = dr.Store(NamedEntity)

with open("my-corpus.dr", "rb") as f:
    reader = dr.Reader(f, Doc)
    for doc in reader:
        logger.info("Processing document '%s'", doc.doc_id)
        for token in doc.tokens:
            process_token(token)
```
Slices

- Slices are a \(\langle \text{start index, length} \rangle\) pair over a sequence
- Represented internally as these two integer values
- They can slice over byte sequences (original document) or Stores
  - Stores are implied to have a logical ordering.

```python
1 class Token(dr.Ann):
  2     span = dr.Slice() # Slice over a byte stream
  3     norm = dr.Text()

4
5 class NamedEntity(dr.Ann):
  6     span = dr.Slice(Token) # Slice over the Store of Token's
  7     label = dr.Text()
```

Tim Dawborn, James R. Curran

docrep: A lightweight and efficient document representation framework

COLING 2014
OntoNotes 5

- TODO
Ratinov and Roth [2009]

- This paper pulled together a whole bunch of existing work to create a new state of the art
- A lot of this presentation goes over the components of such a