Inferring Citation Styles from Examples with a Generative Model

Simon Kornblith

May 13, 2011

A citation style format may be thought of as a simple program or a context-free grammar. Citation Style Language (CSL), the language that Zotero, Mendeley, Papers2, and several other modern bibliographic tools use to format citations, defines a citation in terms of variable elements and groups. CSL is a well-documented format (schema available at `https://github.com/citation-style-language/schema/raw/master/csl.rnc`) that is essentially quite simple, although there are a number of complexities that will not be considered here. Each variable element may have a prefix or suffix. A group may also have a prefix or suffix, but contains a set of child groups, each with their own prefixes and suffixes. This complication is necessary in order to produce proper formatting if one or more formatting elements is absent. For example, APA style for journal articles is

`author. (year). title. container-title, volume(issue), page.`

In the absence of an issue number and page number, we would like the output

`author. (year). title. container-title, volume.`

A good style definition in CSL (ignoring complexities in CSL date formatting for the moment) would thus be

```
<group suffix="." delimiter=". ">
    <text variable="author"/>
    <text prefix="(" variable="year" suffix=")"/>
    <text variable="title"/>
    <group delimiter=", ">
        <text variable="container-title"/>
        <group>
            <text prefix="" variable="volume"/>
            <text prefix="(" variable="issue" suffix=")"/>
        </group>
        <text variable="page"/>
    </group>
</group>
```

In this project, I have attempted to create a generative model that can infer citation styles from an example of their output. The probability of individual prefixes, suffixes, and delimiters is inferred from the 336 valid styles currently available in the CSL repository (https://github.com/citation-style-language/). A generative model constructs a recursive style definition based upon these prefixes, suffix, and delimiters. The Levenshtein distance between the preferred output and the output of a given sample is used to make MCMC inference possible (with noisy-lev-list-equal?).

The included code is structured as follows:

> `get_data.py` - A Python script for getting the distribution over prefixes and suffixes from the styles in the `styles/` directory. This script uses ElementTree 1.3 and thus requires Python 2.7.

`main.ss` - Church code for performing inference.

`tokenizer.ss` - Scheme (not Church) code for converting a citation style example string of the form above into the format used by `main.ss`. `main.ss` takes a citation style example as a list of strings, each of which is either a variable name or a single character, in order to make use of `noisy-lev-list-equal?`. A separate program is necessary because tokenizing requires the use of `string-length`, which is not available within a `church` block and apparently cannot be mimicked with more primitive functions.

`styles/` - The CSL styles parsed to estimate the distribution of prefixes and suffixes

The current implementation is capable of inferring a citation style that generates the desired output, but the inferred style is generally only somewhat robust to perturbation in input data. Example output for the APA example above is

```
<text variable="author" suffix="."/>
 <group delimiter=". ">
    <text variable="year" prefix=" (" suffix=")"/>
    <text variable="title"/>
    <group>
        <text variable="container-title" suffix=", "/>
        <text variable="volume"/>
    </group>
</group>
<group delimiter=", " suffix=".">
    <text variable="issue" prefix="(" suffix=")"/>
    <text variable="page"/>
</group>
```

This would format the reduced case above as:

```
author. (year). title. container-title, volume
```

(The final period is missing.) There are several limitations that I believe are responsible for the suboptimality:

- Because CSL is significantly more complicated than what I have attempted to model in this project and Python script does not attempt to account for all additional sources of complexity, the empirical probabilities are not in perfect correspondence.
- Many styles do not actually provide correct formatting when variables are missing, so the distribution over prefixes and suffixes is likely to be incorrect.
- The model is naive. It disregards the (very important) conditional dependencies between prefix, suffix, and variable. For example, the year is much more likely to be enclosed in parentheses than the author, and if the prefix contains an open parenthesis, the suffix will always contain a close parenthesis. The training data is unlikely to be sufficient to model these conditional dependencies appropriately.
- It may be necessary to enforce additional constraints or to provide reduced examples to fit to avoid generating models that inappropriately punctuate.
- MCMC inference generates correlated samples.