Cloud Computing for the Humanities: Two Approaches for Language Technology

Graham WILCOCK [1]
University of Helsinki, Finland

Abstract. The paper describes *Aelred*, a web application that demonstrates the use of language technology in the Google App Engine cloud computing environment. *Aelred* serves up English literary texts with optional concordances for any word and a range of linguistic annotations including part-of-speech tagging, shallow parsing, and word sense definitions from WordNet.

Two alternative approaches are described. In the first approach, annotations are created offline and uploaded to the cloud datastore. In the second approach, annotations are created online within the cloud computing framework. In both cases standard HTML is generated with a template engine so that the annotations can be viewed in ordinary web browsers.

Keywords. Cloud computing, humanities computing, language technology

Introduction

The paper describes *Aelred*, a web application that demonstrates the use of language technology in a cloud computing environment. As an example of the conference theme "Language resources and technology for the Humanities", *Aelred* serves up English literary texts with optional concordances and a range of linguistic annotations including part-of-speech tagging, shallow parsing, and word sense definitions from WordNet. All the annotations are created automatically by NLP tools.

In this initial demonstration version, the texts are the six main novels of Jane Austen (Figure 1). The total number of words is about half a million. The raw texts (prior to being annotated) are the plain text versions of the novels from Project Gutenberg (`http://www.gutenberg.org`), whose pioneering work in providing freely-available texts on the web has been a huge contribution to humanities computing.

Aelred runs on the Google App Engine cloud computing framework (`http://appengine.google.com`). The application can be accessed from any web browser at the URL `http://aelred-austen.appspot.com`. In Figure 1, selecting a novel leads to a list of its chapters (Figure 2). The start of the text of each chapter is shown alongside the button. Selecting a chapter leads to the chapter text (Figure 3), initially in a plain text format which has been tokenized as described in Section 3.

[1]Corresponding Author: Graham Wilcock, University of Helsinki, P.O. Box 24, 00014 Helsinki, Finland; E-mail: graham.wilcock@helsinki.fi.

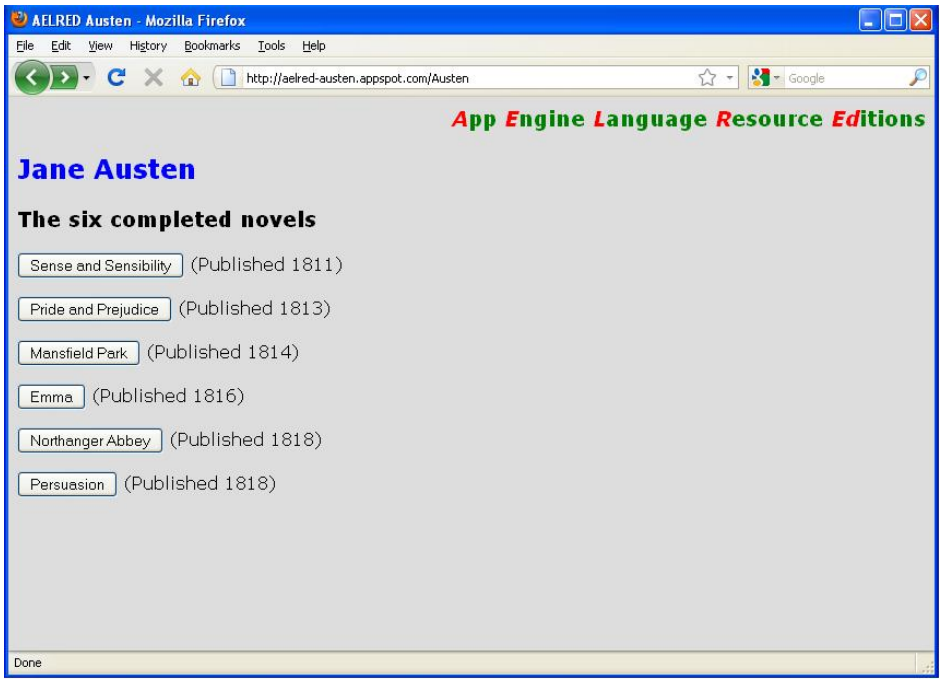

Figure 1. List of available books by Jane Austen.

Aelred is implemented in Python and uses several tools from NLTK, the Python Natural Language Toolkit. NLTK (`http://www.nltk.org`) is a set of open source Python tools and resources for natural language processing, whose companion textbook *Natural Language Processing with Python* [1] provides an excellent self-contained course in human language technology. However, not all NLTK tools can be used directly with Google App Engine, as discussed in Section 3.

1. Two Approaches

Two approaches can be taken to implementing language technology in App Engine. The reason for two different approaches is that the cloud computing framework imposes certain restrictions on applications, as specified in the App Engine documentation [2].

First, there are restrictions on data storage. Data must be stored using App Engine Datastore. This means that only a restricted set of data types can be stored as members of lists. To solve this problem, *Aelred* converts all annotations into a serialized YAML format as described in Section 2 and stores them in Datastore as large text strings.

Second, there are restrictions on the code. Python code uploaded to App Engine must be pure Python. This means that some NLTK tools cannot be used in App Engine, and alternative *Aelred* tools are used as described in Section 3.

Working within these restrictions, two alternative approaches can be taken. In one approach, annotations are created off-line. The annotations are then serialized to YAML and uploaded to Datastore. In the other approach, pure Python tools are used to perform

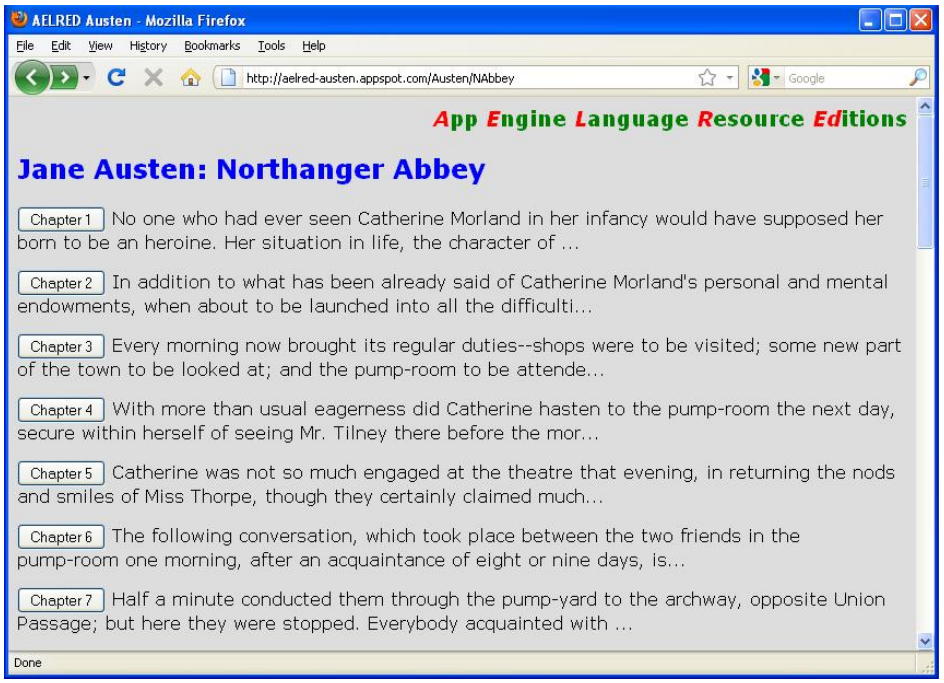

Figure 2. List of chapters of *Northanger Abbey*, with first lines.

language technology tasks on-line in App Engine. These tools also use serialized YAML formats, but store and retrieve Datastore files on-the-fly.

2. Using YAML

Aelred processes texts chapter by chapter. Each chapter is divided into a list of paragraph strings. Each paragraph is divided into a list of sentences and each sentence is divided into a list of tokens. A structured object is created for each token, modelled by a Token class. App Engine provides facilities for defining data models in a very similar way to the well-established Django facilities [3] for defining data models easily in Python. The sentence annotations are represented as lists of Token objects, and the Token objects are represented as Python dictionaries with multiple key:value pairs. The words, the part-of-speech tags, and the chunk labels are all included in the Token structures.

The lists of structured Token objects cannot be stored directly in App Engine, because Datastore only allows lists to contain a restricted set of data types [2]. Therefore the annotations are first serialized to YAML (`http://www.yaml.org`), and the YAML files are uploaded to App Engine Datastore as long text strings.

YAML ("YAML Ain't Markup Language") is a light-weight data format that many people prefer to XML. Python data structures including lists and dictionaries are easily serialized to YAML using `simplejson` [4]. When a text chapter is requested by a user, the relevant YAML file is retrieved from Datastore and deserialized. The annotations are displayed as described in Section 6.

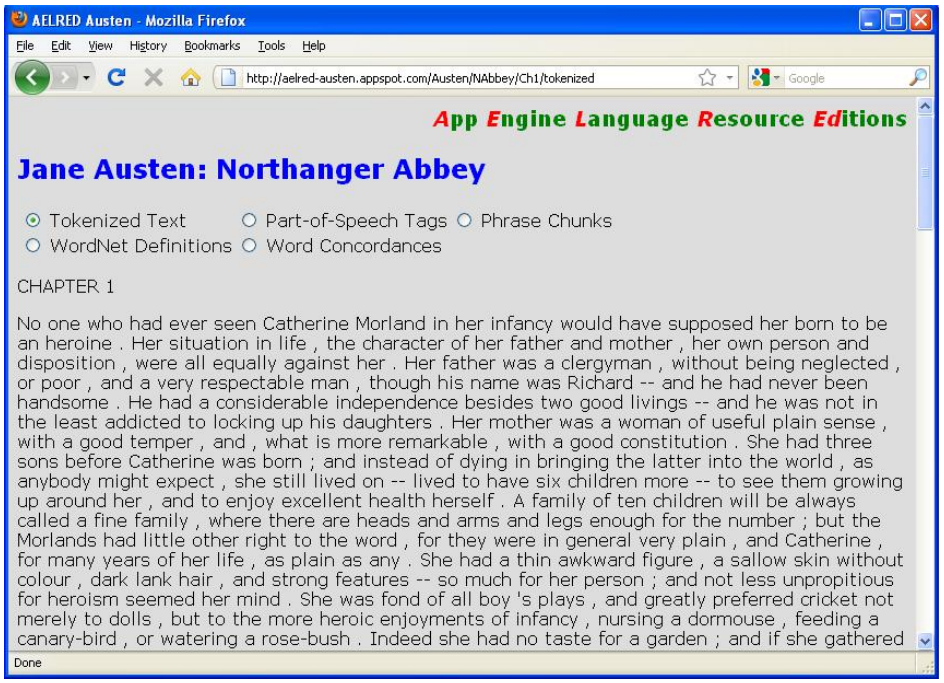

Figure 3. Tokenized Plain Text of *Northanger Abbey*, Chapter 1.

3. NLTK and App Engine

There have recently been discussions on the NLTK users forum (`http://groups.google.com/group/nltk-users`) about problems encountered when attempting to use NLTK with App Engine. One of the aims of *Aelred* is to build a demonstration prototype showing what can be done in practice, and to identify those cases where NLTK can be used with App Engine and those cases where it cannot, as detailed below.

When annotations are created offline and uploaded, there are no restrictions on the tools that create the annotations because the tools do not run inside App Engine. All the standard NLTK tools can therefore be used, including the sentence boundary detector `nltk.sent_tokenize()`, the word tokenizer `nltk.word_tokenize()`, the part-of-speech tagger `nltk.pos_tag()` and the classifier-based named entity recognizer `nltk.ne_chunker()`. However, some of these tools do not suit the Gutenberg texts, so alternative *Aelred* tools are used even in the off-line case.

In the on-the-fly approach, annotations are created by tools running inside the App Engine framework. Tools written in pure Python can be used in App Engine, but tools written in C cannot be used. Some of the NLTK tools are pure Python so they can be imported into App Engine successfully, but some cannot. *Aelred* therefore uses alternative tools that are pure Python, so they can be imported into App Engine.

The NLTK sentence detector `nltk.sent_tokenize()`, which is based on the Punkt sentence boundary detector [5], is pure Python and can be used in App Engine. The code can be loaded from a pickled file, which is uploaded to App Engine with the

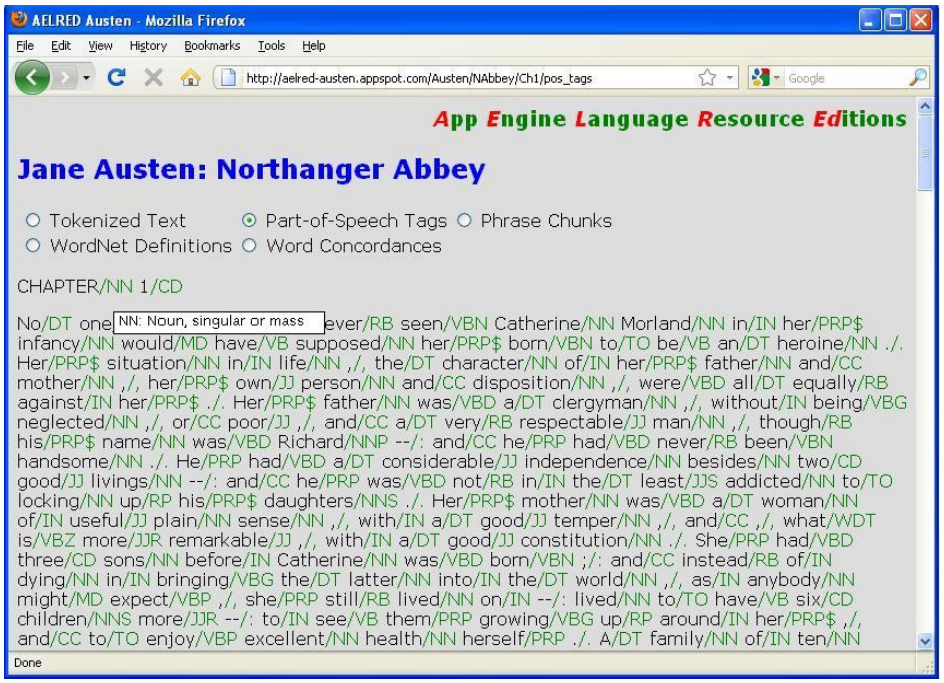

Figure 4. Part-of-Speech Tagging, with mouse over "CHAPTER/NN".

rest of the code. Pickled files can be used in App Engine so long as the pure Python `pickle` is used, not the C version (`cPickle`).

The NLTK tokenizer `nltk.word_tokenize()` is pure Python and can be used in App Engine, but *Aelred* does not use it because there are specific problems in tokenizing the Gutenberg texts. One problem is the frequent use of a double hyphen (`--`) to represent a dash. For example, the third sentence in *Northanger Abbey* starting with "*Her father was a clergyman, ...*" includes the string `Richard--and`. This is tokenized as a single token by the standard NLTK tokenizer. The *Aelred* tokenizer splits this into three tokens as shown in Figure 3.

The NLTK part-of-speech tagger `nltk.pos_tag()` cannot be used directly in App Engine because it uses the NLTK maximum entropy classifier, which uses `numpy`, and `numpy` is not pure Python as it uses C. *Aelred* therefore uses an alternative pure Python tagger trained on the NLTK Treebank corpus, a subset of the full Penn Treebank corpus. The tagger is uploaded into App Engine as a `pickle` file. Part-of-speech tagging for the start of *Northanger Abbey* is shown in Figure 4.

The part-of-speech tags are also used for phrase chunking. A shallow parser, which is currently under development, performs chunking for NPs, PPs and VPs, using NLTK tag pattern matching as described in [1]. The different kinds of phrase chunks are displayed in different colours, as described in Section 6.

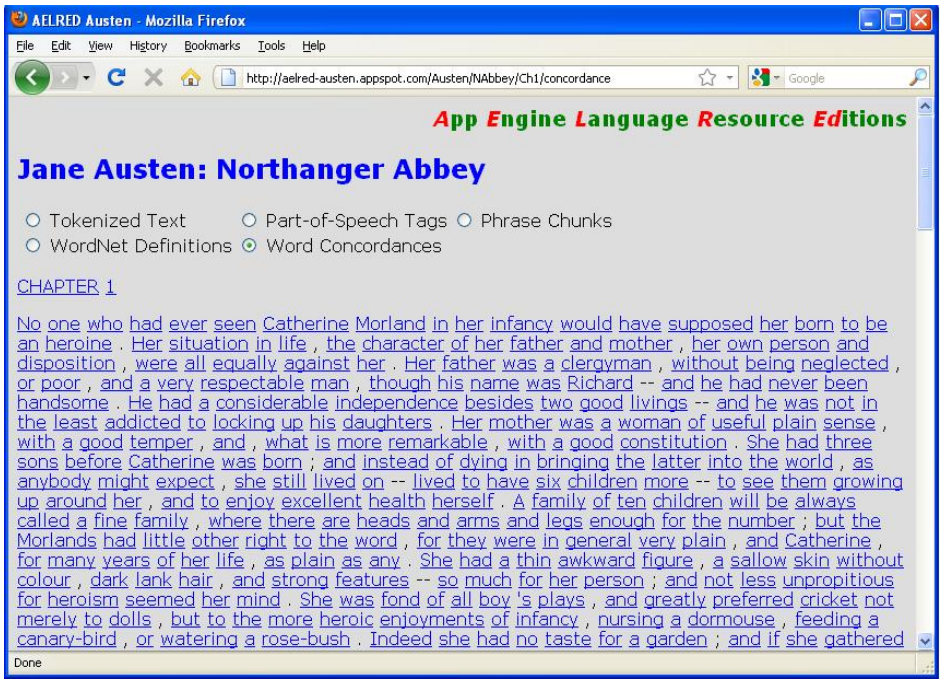

Figure 5. Words with clickable links to their concordances.

4. Making Concordances

Concordances are created using NLTK's `ConcordanceIndex()` method, and show all occurrences of a word in a novel, not chapter by chapter. The offsets for the whole novel are calculated off-line and uploaded to Datastore in a serialized YAML format.

In the concordances view of the text (Figure 5), all the words are displayed as clickable links so that any word's usage can be seen simply by clicking on the word. The concordance for the word is then generated and displayed as a standard HTML table. The concordance for *handsome* (Figure 6) shows that Austen used this adjective for both male and female characters.

5. Using WordNet

A Python interface to WordNet [6] is bundled with NLTK. *Aelred* uses the interface to get word sense definitions from WordNet for nouns, verbs, adjectives and adverbs, excluding stopwords. Words that have WordNet definitions and are not stopwords are highlighted in the browser display, as shown in Figure 7.

When the user hovers the cursor over one of these highlighted words, the WordNet definition is displayed in a pop-up tooltip as described further in Section 6. Only definitions for the part of speech given by the part-of-speech tagger are displayed. Where there are multiple word senses for the same part of speech, a simple form of word sense disambiguation is used to select the most appropriate definition.

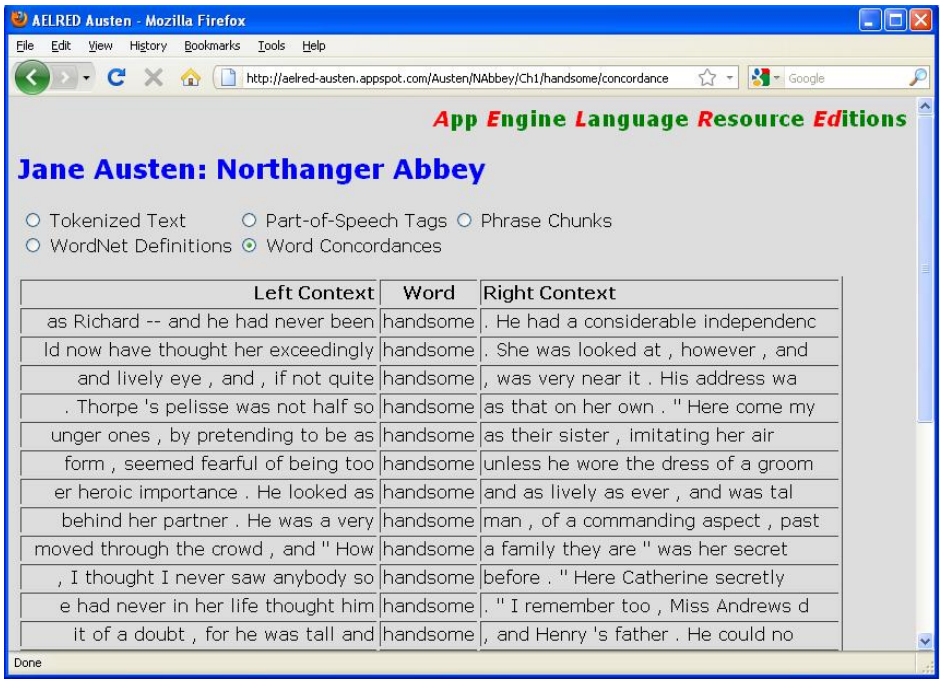

Figure 6. Concordance for "handsome", referring to males and females.

6. Displaying the Annotations

Annotation frameworks often provide special tools for viewing annotations, but App Engine is used with standard web browsers so *Aelred* displays annotations using standard HTML tags in combination with CSS stylesheets.

The HTML is generated using Django (`www.djangoproject.com`) templates. Django is a widely-used open source Python web development framework. *The Django Book* [3] is an excellent tutorial and guide on how to build web apps with Django. Django 0.96 is bundled with App Engine.

Words or phrases are highlighted in specific colours by means of the `<span>` tag. This is used in various ways: to highlight part-of-speech tags (Figure 4), to distinguish different kinds of phrase chunks (NPs, PPs, VPs), and to highlight words for which a WordNet definition is available (Figure 7). The colour scheme is specified by a CSS stylesheet, so the choice of colours can easily be changed.

Tooltip strings are displayed when the user hovers the cursor over a particular word. For example, in Figure 4 the cursor was placed over *CHAPTER/NN* and the expanded description of the NN tag from the Penn Treebank tagset is shown in the pop-up tooltip. This is done by means of the `title` attribute in the `<span>` tag. In Figure 7 the cursor was placed over the word *heroine* and the WordNet definition for this word is shown in the tooltip.

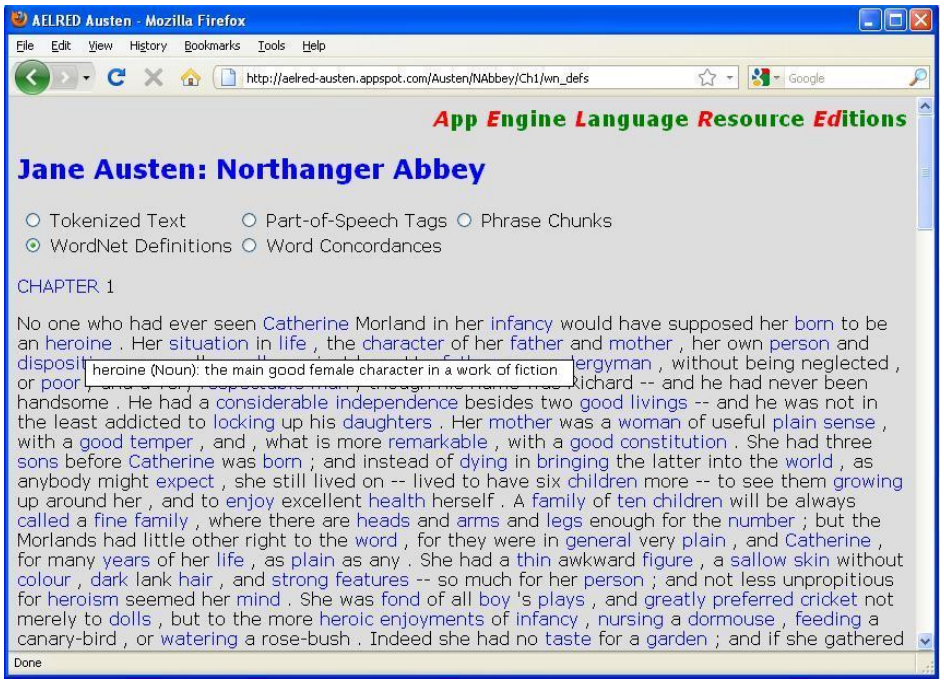

Figure 7. Pop-up WordNet Definitions, with mouse over "heroine".

7. Conclusion

The paper has described *Aelred*, a web application that demonstrates the use of language technology in the Google App Engine cloud computing environment. *Aelred* serves up English literary texts with optional concordances and a range of linguistic annotations, including part-of-speech tagging, shallow parsing, and word sense definitions from WordNet.

Two approaches were described. In the first approach, annotations are created offline and uploaded to the cloud datastore. In the second approach, annotations are created online within the cloud computing framework. In both cases standard HTML is generated with a template engine so that the annotations can be viewed in ordinary web browsers.

References

[1] S. Bird, E. Klein and E. Loper, *Natural Language Processing with Python*, O'Reilly, 2009.

[2] Google App Engine documentation, *The Python Runtime Environment*, `http://code.google.com/appengine/docs/python/`, 2010.

[3] A. Holovaty and J. Kaplan-Moss, *The Django Book (version 0.96)*, 2009.

[4] SimpleJSON documentation, *JSON encoder and decoder*, `http://code.google.com/p/simplejson/`, 2010.

[5] T. Kiss and J. Strunk, Unsupervised Multilingual Sentence Boundary Detection, *Computational Linguistics* **32** (2006), 485–525.

[6] G. A. Miller, WordNet: A Lexical Database for English, *Communcations of the ACM* **38** (1995), 39–41.