

Processing Shoebox Data with the NLTK

Stuart Robinson, Max Planck Institute
stuart.robinson@mpi.nl or stuart@zapata.org

September 13, 2005

Contents

1	Shoebox	2
1.1	Pros and Cons	2
1.1.1	Pros	2
1.1.2	Cons	2
1.2	Dictionaries	2
1.2.1	A Machine Friendly Shoebox Dictionary	2
1.2.2	A Not So Machine-Friendly Shoebox Dictionary	4
2	Python and the NLTK	6
2.1	Python	6
2.2	The NLTK	7
3	Shoebox Functionality in the NLTK	7
3.1	An Object Model, Sort Of	7
3.2	The Object Model in Action	8
3.3	Some Illustrations	10
3.4	Feature Wishlist	12
4	On-line Resources	12

1 Shoebox

According to the official Shoebox web site:

“Shoebox is a computer program that helps field linguists and anthropologists integrate various kinds of text data: lexical, cultural, grammatical, etc. It has flexible options for sorting, selecting, and displaying data. It is especially useful for helping researchers build a dictionary as they use it to analyze and interlinearize text. The name Shoebox recalls the use of shoe boxes to hold note cards on which definitions of words were written in the days before researchers could use computers in the field.”

1.1 Pros and Cons

1.1.1 Pros

1. Good integration of lexicon and text interlinearizing
2. User-friendly GUI
3. Widely used and fairly feature-rich

1.1.2 Cons

Basically, the main problem is that the data is neither XML nor a relational database—i.e.,

1. No recursion
2. Limited analytical capabilities (no standardized query language)
3. Few mechanisms for constraint enforcement (exception: range sets)

Note: Toolbox is free, but Shoebox proper is not. Both are proprietary—i.e., they’re not open source so you can’t look under the hood. It would be nice if they open-sourced the project.

1.2 Dictionaries

1.2.1 A Machine Friendly Shoebox Dictionary

A machine-tractable dictionary of Rotokas (Papuan, Bougainville) is currently under development by the author. Here’s a sample entry:

(1) \lx tasiasi
 \ps V.B

```

\ge stomp
\eng stomp on
\eng step on repeatedly
\gp krungutim
\vx 2
\arg OBL
\cm -ia
\dt 25/Aug/2005
\ex Guruvara-ia tasiiasiparevoi.
\xp Em i wok long krungutim ol lip.
\xe He is stomping on the leaves.
\ex Kuvukuvu tou-ia tasiiasipai ovusia gurukopai.
\xp Ol i wok long krungutim giraun na giraun i pairap.
\xe They are stomping on the ground while it shakes.

```

Derived lexemes are related to their sources by the field `\rt` field—e.g., *kasipu* ‘to be angry’ \Rightarrow *kasipupie* (*kasipu* + *pie*) ‘to make angry’:

```

(2) \lx kasipupie
    \rt kasipu
    \ps V.B
    \ge anger
    \eng anger
    \eng enrage
    \gp mekim kros
    \vx 2
    \arg 0
    \dt 25/Aug/2005
    \ex Teapi rera kasipupieive orareoreopaoro.
    \xp ???
    \xe Don't make him angry talking among yourselves.
    \ex Ae, visii ragai kasipupietavoi.
    \xp Hei, yupela i mekim mi kros.
    \xe Hey, you guys are making me angry.

```

This creates derivational chains—e.g., *tarai* ‘understand’ \Rightarrow *tarai pie* ‘teach’ \Rightarrow *oratarai pie* ‘learn (literally: self-teach)’.

Lexemes are not unique, as can be seen from the following two entries for *keke*, distinguished by the part-of-speech field (`\ps`):

```

(3) \lx keke
    \ps V.B
    \ge look.at
    \eng look at
    \gp lukim
    \vx 2
    \arg 0
    \cmt Example needed

```

```

\sc PERCEPTION
\dcp TRUE
\dt 02/Aug/2005

\lx keke
\ps V.A
\ge look
\gp luk
\dcs True
\cm -ia
\vx ???
\dt 17/Jul/2005
\ex Vearo kekepau ragai osireiaro-ia vii oupa ovusia ragai taviri.
\xp Yu luk gut tru long ai bilong mi bai mi maritim yu sapos yu tok orait.
\xe You look good in my eyes I will marry you if you tell me.

```

No mechanism exists in Shoebox for enforcing uniqueness constraints, as far as I am aware.

1.2.2 A Not So Machine-Friendly Shoebox Dictionary

Some ways of organizing a Shoebox dictionary are less friendly to automatic parsing. For example, consider this entry from a Shoebox dictionary for Teop, an Austronesian language of Bougainville:¹

```

(4) \lx aavuu
    \ps n.e.
    \ge granny
    \de (inland dialect) grandparent
    \xv 0 bana te aavuu orau.
    \xe That is granny's mat.
    \cf bubuu
    \dt 24/Feb/2005

```

So far, so good. But now consider this one:

```

(5) \lx abana
    \hm 1

    \ps v.i.
    \ge jump
    \de jump
    \xv Abana gunaha.
    \xe Jump down.

    \se abaabana

```

¹Many thanks to Ulrike Mosel for letting me use this material.

```

\ps v.i.
\ge jump
\de jump
\xv 0 vahara beiko na abaabana rori teo biris.
\xe The children are jumping from the bridge (into the sea, in play).

\se vaaba'abana
\ps v.t.
\ge jump.repeatedly
\de jump repeatedly
\xv A teebana na vanuganuga nana, o vahara beiko na vaabaabana rori teebona.
\xe The bed is moving and shaking about, the children are jumping on it.
\dt 20/Jan/2005

```

Note that there is a second entry for *abana*, distinguished by the field `\hm`:

```

(6) \lx abana
    \hm 2

    \ps n.a.
    \ge people
    \de (group of) people
    \xv A abana paa tagusiu tea kosi atovo.
    \xe The people all went to cut sago palm leaves.

    \se abana pinopino
    \ps n.a.
    \ge
    \de people of nowhere, strangers, people with no status

    ...

```

A few things about the organization of the Teop Shoebox dictionary that are worth noting:

- head field is `\lx`
- some entries contain subsentries (`\se`)
- some subsentries lack the field `\se`—i.e., no consistent head field for subsentries

This is very hard to parse, especially without metadata (where marker hierarchies might help). A more machine-tractable alternative would be something like this:

```

(7) \lx abana1
    \ps root

```

```

\lx abana
\rt abana1
\ps v.i.
\ge jump
\de jump
\xv Abana gunaha.
\xe Jump down.

\lx abaabana
\rt abana1
\ps v.i.
\ge jump
\de jump
\xv 0 vahara beiko na abaabana rori teo biris.
\xe The children are jumping from the bridge (into the sea, in play).

\lx vaaba'abana
\rt abana1
\ps v.t.
\ge jump.repeatedly
\de jump.repeatedly
\xv A teebana na vanuganuga nana, o vahara beiko na vaabaabana rori teebona.
\xe The bed is moving and shaking about, the children are jumping on it.
\dt 20/Jan/2005

```

2 Python and the NLTK

2.1 Python

A few highlights:

- scripting language (interpreted, not compiled—”shaken, not stirred”)
- easy-to-learn, consistent syntax
- object-oriented to the bone
- large community (used for prototyping by Google, so it must be good!)
- reasonably mature and full-featured
- ample documentation

There is an introduction to Python programming for linguists in the works, being written by yours truly (with Harald Baayen), tentatively titled, *An Introduction to Python Programming for Language Research*. (Draft available by request—just email me.)

2.2 The NLTK

According to Loper and Bird (2002, 2004):

“a suite of program modules, data sets, tutorials, and exercises, covering symbolic and statistical natural language processing”

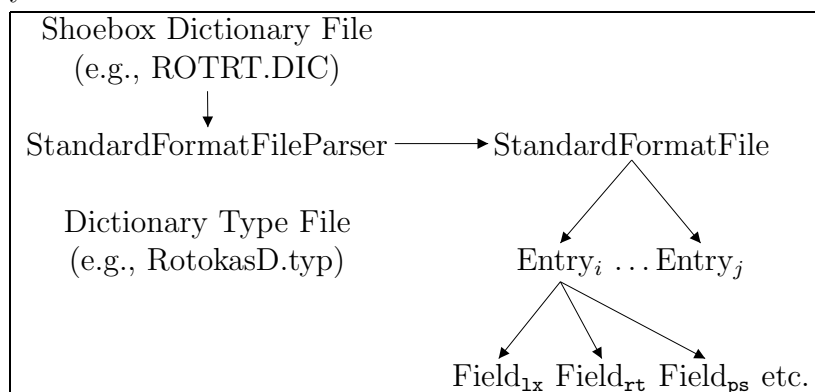
Designed with some of the following considerations in mind (this comes straight from the official web site):

- Easy-to-Use
- Consistent
- Extensible
- Simple
- Modular
- Well-documented

3 Shoebox Functionality in the NLTK

3.1 An Object Model, Sort Of

The basics of parsing a Shoebox dictionary file are illustrated diagrammatically below:



An entry consists of a dictionary of fields that preserves the order of its keys. An entry like this

```
(8) \lx aa
    \ps INTERR
    \ge which
    \gp wanem
    \gp husat
    \ex Aavaoa vao oa-pa ruipapau?
    \xp Wanem dispela yu laikim?
    \xe Which one of these do you want?
```

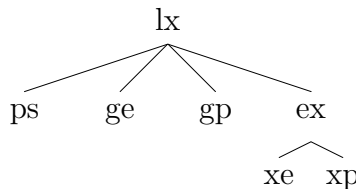
would be stored in an object as a dictionary looking something like this

```
(9) { 'lx' : ['aa'],
      'ps' : ['INTERR'],
      'ge' : ['which'],
      'gp' : ['wanem', 'husat'],
      'ex' : ['Aavaoa vao oa-pa ruipapau?'],
      'xe' : ['Wanem dispela yu laikim?'],
      'xp' : ['Which one of these do you want?'] }
```

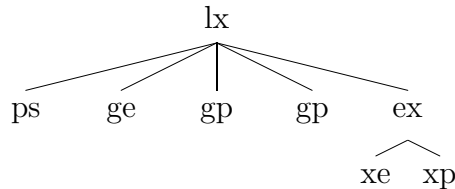
Note: A field object consists of a field marker and as many values are present in the entry from which it was obtained. This makes head fields somewhat special, since they can only have a single value. Perhaps a field object should only be a specific field and not a collection of fields. More concretely, for (8) perhaps there should be two fields for `\ge` instead of one.

Using marker hierarchies, this data could also be represented as a tree—i.e.,

(10) a.



b.



3.2 The Object Model in Action

A simple script that parses a dictionary file (without using metadata) is provided below:

```
(11) parse.py
1 from standardformat import StandardFormatFileParser
2 import sys
3
4 # Get path to Shoebox file from command-line
5 try :
6     dictFile = sys.argv[1]
7 except IndexError :
8     sys.stderr.write("Usage: %s <SHOEBOX DICTIONARY>\n" % sys.argv[0])
9     sys.exit(0)
10
```

```

11 # Create parser object from Shoebox file
12 sffp = StandardFormatFileParser(dictFile)
13
14 # Parse file into StandardFormatFile object
15 sff = sffp.parse()
16
17 # Extract entries
18 entries = sff.getEntries()
19
20 # Count entries
21 print len(entries)
22

```

It is also possible to use Shoebox metadata (Shoebox dictionary type definition file) to validate a dictionary:

```

(12) validate-range-sets.py
1 import sys
2 from shoebox      import DictionaryMetadataParser, ShoeboxValidator
3 from standardformat import StandardFormatFileParser
4
5 # Get path to dictionary (data) and dictionary type
6 # (metadata) files from command-line
7 shoeboxFile = None
8 metadataFile = None
9 try :
10     shoebox, metadataFile = sys.argv[1], sys.argv[2]
11 except :
12     sys.stderr.write("Usage: %s <DATA> <METADATA>" % sys.argv[0])
13     sys.exit(0)
14
15 # Parse metadata file into Metadata object and get marker set
16 metadataParser = DictionaryMetadataParser(metadataFile)
17 metadata = metadataParser.parse()
18 markerMetadata = metadata.getMarkerSet()
19
20 # Parser dictionary file into StandardFormatFile object
21 dictionaryParser = StandardFormatFileParser(shoebox)
22 dictionaryParser.setHeadFieldMarker(metadata.getHeadFieldMarker())
23 d = dictionaryParser.parse()
24
25 # Go through entries and check field values against range sets
26 for e in d.getEntries() :
27     badValues = []
28
29     # Go through each field
30     for fm in e.getFieldMarkers() :
31         fvs = e.getFieldValuesByFieldMarker(fm)
32         fmm = markerMetadata[fm]

```

```

33     # See a range set is defined for the field
34     if fmm.getRangeSet() :
35         for fv in fvs :
36             # See whether value is outside defined range set
37             if not fv in fmm.getRangeSet() :
38                 msg = "'%s' invalid for for '%s'" % (fv, fm)
39                 badValues.append(msg)
40     if badValues :
41         lx = e.getHeadField().getValuesAsString()
42         print "%s : %s" % (lx, ", ".join(badValues))

```

Because the order of fields in an entry is preserved, it is easy to manipulate specific fields and otherwise leave a dictionary intact. Here's some code that filters out (i.e., removes) whatever field is specified by the user:

```

(13) filter-fields.py
1 import sys, re
2 from standardformat import StandardFormatFileParser
3
4 # Get field to filter and path to Shoebox dictionary
5 # file from command-line
6 try :
7     field2Filter, filepath = sys.argv[1], sys.argv[2]
8 except :
9     sys.stderr.write("%s -m <FIELD> <FILE>\n" % sys.argv[0])
10    sys.exit(0)
11
12 # Create StandardFormatFile object
13 fp = StandardFormatFileParser(filepath)
14 sff = fp.parse()
15
16 # Get header and print it
17 print sff.getHeader()
18
19 # Remove user-specified field from each entry and print
20 # them -- entry printed out by default in standard format
21 # with fields in the same order that they were parsed in
22 for e in sff.getEntries() :
23     e.removeField(field2Filter)
24     print e

```

3.3 Some Illustrations

There are lots of possibilities, such as these discussed in the on-line tutorial (see §4):

- formatting a dictionary for display

- adding a field (or multiple fields) to a dictionary automatically
- automatically extracting minimal pairs
- extracting entries modified within a date range
- guessing entry template of dictionary without metadata
- finding entries that conform to a particular profile
- automatic analysis of word length and phonotactics
- changing the order of fields in an entry
- breaking a single field into many
- checking fields for uniqueness

For a concrete illustration, here's how I recently identified all of the ambitransitive stems (i.e., stems that can function as intransitives (Class A) or as transitives (Class B) without recourse to valency-changing derivations) in my Rotokas lexicon:

```

(14) find-ambitransitives.py
1 import sys
2 from standardformat import StandardFormatFileParser
3
4 # Get filepath from command-line
5 try :
6     dictFile = sys.argv[1]
7 except IndexError :
8     sys.stderr.write("Usage: %s <SHOEBOX DICTIONARY>\n" % sys.argv[0])
9     sys.exit(0)
10
11 # Parse dictionary and build data structure
12 stems = {}
13 sffp = StandardFormatFileParser(dictFile)
14 sff = sffp.parse()
15 for e in sff.getEntries() :
16     lx = e.getHeadField().getValuesAsString()
17     ps = e.getField("ps").getValuesAsString()
18     if not stems.has_key(lx) : stems[lx] = {}
19     stems[lx][ps] = e
20
21 # Analyze data structure and extract stems of interest
22 lexemes = stems.keys()
23 lexemes.sort()
24 for lx in lexemes :
25     partsOfSpeech = stems[lx]
26     if len(partsOfSpeech) > 1 :
27         classA, classB = None, None
28         # Go through entries for lexeme by part-of-speech
29         for ps in partsOfSpeech :
30             if ps == "V.A" :
31                 classA = stems[lx][ps]
32             elif ps == "V.B" :

```

```

33     classB = stems[lx][ps]
34     # It's an ambitransitive!
35     if classA and classB :
36         glossA = classA.getField("ge")
37         glossB = classB.getField("ge")
38         print "%s" % lx
39         print " '%s' (A)" % glossA.getValuesAsString(", ")
40         print " '%s' (B)" % glossB.getValuesAsString(", ")

```

3.4 Feature Wishlist

There are various additional features that are on the official wish list but have not yet been implemented:

- marker hierarchies
- more validation options
- RTF export support
- project file updating
- Unicode support

User feedback would be much appreciated!

4 On-line Resources

To learn more:

- www.python.org
- nltk.sourceforge.net
- www.sil.org/computing/shoebox/
- www.sil.org/computing/toolbox/

A tutorial, this handout, and the code in it, can be found somewhere here:

www.zapata.org/stuart/

References

Edward Loper and Steven Bird. NLTK: The natural language toolkit. 2002.
 URL <http://arxiv.org/abs/cs/0205028>.

Edward Loper and Steven Bird. NLTK: The natural language toolkit. 2004.
 URL <http://www ldc.upenn.edu/sb/home/papers/nltk.pdf>.