The NLTK FrameNet API: Designing for Discoverability with a Rich Linguistic Resource

Nathan Schneider
Georgetown University
Washington, DC
nathan.schneider@georgetown.edu

Chuck Wooters
Semantic Machines
Berkeley, CA
wooters@semanticmachines.com

Abstract

A new Python API, integrated within the NLTK suite, offers access to the FrameNet 1.7 lexical database. The lexicon (structured in terms of frames) as well as annotated sentences can be processed programatically, or browsed with human-readable displays via the interactive Python prompt.

1 Introduction

For over a decade, the Berkeley FrameNet (henceforth, simply "FrameNet") project (Baker et al., 1998) has been documenting the vocabulary of contemporary English with respect to the theory of frame semantics (Fillmore, 1982). A freely available, linguistically-rich resource, FrameNet now covers over 1,000 semantic frames, 10,000 lexical senses, and 100,000 lexical annotations in sentences drawn from corpora. The resource has formed a basis for much research in natural language processing—most notably, a tradition of semantic role labeling that continues to this day (Gildea and Jurafsky, 2002; Baker et al., 2007; Das et al., 2014; FitzGerald et al., 2015; Roth and Lapata, 2015, *inter alia*).

Despite the importance of FrameNet, computational users are often frustrated by the complexity of its custom XML format. Whereas much of the resource is browsable on the web (http://framenet.icsi.berkeley.edu/), certain details of the linguistic descriptions and annotations languish in obscurity as they are not exposed by the HTML views of the data.[1] The few open source APIs for reading FrameNet data are now antiquated, and none has been widely adopted.[2]

We describe a new, user-friendly Python API for accessing FrameNet data. The API is included within recent releases of the popular NLTK suite (Bird et al., 2009), and provides access to nearly all the information in the FrameNet release.

2 Installation

Instructions for installing NLTK are found at nltk.org. NLTK is cross-platform and supports Python 2.7 as well as Python 3.x environments. It is bundled in the Anaconda and Enthought Canopy Python distributions for data scientists.[3]

In a working NLTK installation (version 3.2.2 or later), one merely has to invoke a method to download the FrameNet data:[4,5]

```
>>> import nltk
>>> nltk.download('framenet_v17')
```

[1] For example, one of the authors was recently asked by a FrameNet user whether frame-to-frame relations include mappings between individual frame elements. They do, but the user's confusion is understandable because these mappings are not exposed in the HTML frame definitions on the website. (They can be explored visually via the FrameGrapher tool on the website, https://framenet.icsi.berkeley.edu/fndrupal/FrameGrapher, if the user knows to look there.) In the interest of space, our API does not show them in the frame display, but they can be accessed via an individual frame relation object or with the fe_relations() method, §4.4.

[2] We are aware of:

- github.com/dasmith/FrameNet-python (Python)
- nlp.stanford.edu/software/framenet.shtml (Java)
- github.com/FabianFriedrich/Text2Process/tree/master/src/de/saar/coli/salsa/reiter/framenet (Java)
- github.com/GrammaticalFramework/gf-contrib/tree/master/framenet (Grammatical Framework)

None of these has been updated in the past few years, so they are likely not fully compatible with the latest data release.

[3] https://www.continuum.io/downloads, https://store.enthought.com/downloads

[4] >>> is the standard Python interactive prompt, generally invoked by typing python on the command line. Python code can then be entered at the prompt, where it is evaluated/executed. Henceforth, examples will assume familiarity with the basics of Python.

[5] By default, the 855MB data release is installed under the user's home directory, but an alternate location can be specified: see http://www.nltk.org/data.html.

Proceedings of the 2017 EMNLP System Demonstrations, pages 1–6
Copenhagen, Denmark, September 7–11, 2017.

Subsequently, the `framenet` module is loaded as follows (with alias `fn` for convenience):

```
>>> from nltk.corpus import framenet as fn
```

3 Overview of FrameNet

FrameNet is organized around conceptual structures known as **frames**. A semantic frame represents a **scene**—a kind of event, state, or other scenario which may be universal or culturally-specific, and domain-general or domain-specific. The frame defines participant roles or **frame elements** (FEs), whose relationships forms the conceptual background required to understand (certain senses of) vocabulary items. Oft-cited examples by Fillmore include:

- Verbs such as *buy*, *sell*, and *pay*, and nouns such as *buyer*, *seller*, *price*, and *purchase*, are all defined with respect to a commercial transaction scene (frame). FEs that are central to this frame—they may or may not be mentioned explicitly in a text with one of the aforementioned lexical items—are the **Buyer**, the **Seller**, the **Goods** being sold by the **Seller**, and the **Money** given as payment in exchange by the **Buyer**.
- The concept of REVENGE—lexicalized in vocabulary items such as *revenge*, *avenge*, *avenger*, *retaliate*, *payback*, and *get even*—fundamentally presupposes an **Injury** that an **Offender** has inflicted upon an **Injured_party**, for which an **Avenger** (who may or may not be the same as the **Injured_party**) seeks to exact some **Punishment** on the **Offender**.
- A *hypotenuse* presupposes a geometrical notion of right triangle, while a *pedestrian* presupposes a street with both vehicular and nonvehicular traffic. (Neither frame is currently present in FrameNet.)

The FEs in a frame are formally listed alongside an English description of their function within the frame. Frames are organized in a network, including an inheritance hierarchy (e.g., REVENGE is a special case of an EVENT) and other kinds of frame-to-frame relations.

Vocabulary items listed within a frame are called **lexical units** (LUs). FrameNet's inventory of LUs includes both content and function words. Formally, an LU links a lemma with a frame.[6]

[6]The lemma name incorporates a part-of-speech tag. The lemma may consist of a single word, such as *surrender.v*, or multiple words, such as *give up.v*.

In a text, a token of an LU is said to **evoke** the frame. Sentences are annotated with respect to frame-evoking tokens and their FE spans. Thus:

[Snape]$_{\textbf{Injured\_party}}$'s **revenge** [on Harry]$_{\textbf{Offender}}$

labels overt mentions of participants in the REVENGE frame.

The reader is referred to (Fillmore and Baker, 2009) for a contemporary introduction to the resource and the theory of frame semantics upon which it is based. Extensive linguistic details are provided in (Ruppenhofer et al., 2016).

4 API Overview

4.1 Design Principles

The API is designed with the following goals in mind:

Simplicity. It should be easy to access important objects in the database (primarily frames, lexical units, and annotations), whether by iterating over all entries or searching for particular ones. To avoid cluttering the API with too many methods, other information in the database should be reachable via object attributes. Calling the API's `help()` method prints a summary of the main methods for accessing information in the database.

Discoverability. Many of the details of the database are complex. The API makes it easy to browse what is in database objects via the Python interactive prompt. The main way it achieves this is with pretty-printed displays of the objects, such as the frame display in figure 1 (see §4.3). The display makes it clear how to access attributes of the object that a novice user of FrameNet might not have known about.

In our view, this approach sets this API apart from others. Some of the other NLTK APIs for complex structured data make it difficult to browse the structure without consulting documentation.

On-demand loading. The database is stored in thousands of XML files, including files indexing the lists of frames, frame relations, LUs, and full-text documents, plus individual files for all frames, LUs, and full-text documents. Unzipped, the FrameNet 1.7 release is 855 MB. Loading all of these files—particularly the corpus annotations—is slow and memory-intensive, costs which are unnecessary for many purposes. Therefore, the API is carefully designed with lazy data structures to load XML files only as needed. Once loaded, all data is cached in memory for fast subsequent access.

```
frame (347): Revenge

[URL] https://framenet2.icsi.berkeley.edu/fnReports/data/frame/Revenge.xml

[definition]
  This frame concerns the infliction of punishment in return for a
  wrong suffered. An Avenger performs a Punishment on a Offender as
  a consequence of an earlier action by the Offender, the Injury.
  The Avenger inflicting thePunishment need not be the same as the
  Injured_Party who suffered the Injury,  but the Avenger does have
  to share the judgment that the Offender's action was wrong. The
  judgment that the Offender had inflicted an Injury  is made
  without regard to the law.  '(1) They took revenge for the deaths
  of two loyalist prisoners.' '(2) Lachlan went out to avenge
  them.' '(3) The next day, the Roman forces took revenge on their
  enemies..'

[semTypes] 0 semantic types

[frameRelations] 1 frame relations
  <Parent=Rewards_and_punishments -- Inheritance -> Child=Revenge>

[lexUnit] 18 lexical units
  avenge.v (6056), avenger.n (6057), get back (at).v (10003), get
  even.v (6075), payback.n (10124), retaliate.v (6065),
  retaliation.n (6071), retribution.n (6070), retributive.a (6074),
  retributory.a (6076), revenge.n (6067), revenge.v (6066),
  revengeful.a (6073), revenger.n (6072), sanction.n (10676),
  vengeance.n (6058), vengeful.a (6068), vindictive.a (6069)


[FE] 14 frame elements
            Core: Avenger (3009), Injured_party (3022), Injury (3018),
Offender (3012), Punishment (3015)
      Peripheral: Degree (3010), Duration (12060), Instrument (3013),
Manner (3014), Place (3016), Purpose (3017), Time (3021)
  Extra-Thematic: Depictive (3011), Result (3020)

[FEcoreSets] 2 frame element core sets
  Injury, Injured_party
  Avenger, Punishment
```

Figure 1: Textual display of the REVENGE frame. Shown in square brackets are attribute names for accessing the frame's contents. In parentheses are IDs for the frame, its LUs, and its FEs.

4.2 Lexicon Access Methods

The main methods for looking up information in the lexicon are:

frames(name)	**frame**(nameOrId)
lus(name, frame)	**lu**(id)
fes(name, frame)	

The methods with plural names (left) are for searching the lexicon by regular expression pattern to be matched against the entry name. In addition (or instead), `lus()` and `fes()` allow for the results to be restricted to a particular frame. The result is a list with 0 or more elements. If no arguments are provided, all entries in the lexicon are returned.

An example of a search by frame name pattern:[7]

```
>>> fn.frames('(?i)creat')
[<frame ID=268 name=Cooking_creation>,
 <frame ID=1658 name=Create_physical_artwork>, ...]
```

Similarly, a search by LU name pattern—note that the `.v` suffix is used for all verbal LUs:

```
>>> fn.lus(r'.+en\.v')
[<lu ID=5331 name=awaken.v>,
 <lu ID=7544 name=betoken.v>, ...]
```

The `frame()` and `lu()` methods are for retrieving a single known entry by its name or ID. Attempting to retrieve a nonexistent entry triggers an exception of type `FramenetError`.

Two additional methods are available for frame lookup: `frame_ids_and_names(name)` to get a mapping from frame IDs to names, and `frames_by_lemma(name)` to get all frames with some LU matching the given name pattern.

[7] `(?i)` makes the pattern case-insensitive.

```
exemplar sentence (929548):
[sentNo] 0
[aPos] 1113164

[LU] (6067) revenge.n in Revenge

[frame] (347) Revenge

[annotationSet] 2 annotation sets

[POS] 12 tags

[POS_tagset] BNC

[GF] 4 relations

[PT] 4 phrases

[text] + [Target] + [FE] + [Noun]

A short while later Joseph had his revenge on Watney 's .
------------------ ------ ^^^ --- ******* ------------
Time               Avenge sup Ave         Offender



[Injury:DNI]
 (Avenge=Avenger, sup=supp, Ave=Avenger)
```

Figure 2: A lexicographic sentence display. The visualization of the frame annotation set at the bottom is produced by pretty-printing the combined information in the `text`, `Target`, `FE`, and `Noun` layers. Abbreviations in the visualization are expanded at the bottom in parentheses ("supp" is short for "support"). "DNI" is FrameNet jargon for "definite null instantiation"; `GF` stands for "grammatical function"; and `PT` stands for "phrase type".

4.3 Database Objects

All structured objects in the database—frames, LUs, FEs, etc.—are loaded as `AttrDict` data structures. Each `AttrDict` instance is a mapping from string keys to values, which can be strings, numbers, or structured objects. `AttrDict` is so called because it allows keys to be accessed as attributes:

```
>>> f = fn.frame('Revenge')
>>> f.keys()
dict_keys(['cBy', 'cDate', 'name', 'ID', '_type',
'definition', 'definitionMarkup', 'frameRelations',
'FE', 'FEcoreSets', 'lexUnit', 'semTypes', 'URL'])
>>> f.name
'Revenge'
>>> f.ID
347
```

For the most important kinds of structured objects, the API specifies textual **displays** that organize the object's contents in a human-readable fashion. Figure 1 shows the display for the REVENGE frame, which would be printed by entering `fn.frame('Revenge')` at the interactive prompt. The display gives attribute names in square brackets; e.g., `lexUnit`, which is a mapping from LU names to objects. Thus, after the code listing in the previous paragraph, `f.lexUnit['revenge.n']` would access to one of the LU objects in the frame, which in turn has its own attributes and textual display.

4.4 Advanced Lexicon Access

Frame relations. The inventory of frames is organized in a semantic network via several kinds of frame-to-frame relations. For instance, the REVENGE frame is involved in one frame-to-frame relation: it is related to the more general REWARDS_AND_PUNISHMENTS frame by *Inheritance*, as shown in the middle of figure 1. REWARDS_AND_PUNISHMENTS, in turn, is involved in *Inheritance* relations with other frames. Each frame-to-frame relation bundles mappings between corresponding FEs in the two frames.

Apart from the `frameRelations` attribute of frame objects, frame-to-frame relations can be browsed by the main method `frame_relations(frame, frame2, type)`, where the optional arguments allow for filtering by one or both frames and the kind of relation. Within a frame relation object, pairwise FE relations are stored in the `feRelations` attribute. Main method `fe_relations()` provides direct access to links between FEs. The inventory of relation types, including *Inheritance*, *Causative*, *Inchoative*, *Subframe*, *Perspective_on*, and others, is available

```
full-text sentence (4148528) in Tiger_Of_San_Pedro:


[POS] 25 tags

[POS_tagset] PENN

[text] + [annotationSet]

They 've been looking for him all the time for their revenge ,
              *******                                 *******
              Seeking                                 Revenge
              [3] ?                                   [2]

but it is only now that they have begun to find him out . "
                                  *****    ****
                                  Proce    Beco
                                  [1]      [4]
 (Proce=Process_start, Beco=Becoming_aware)
```

Figure 3: A sentence of full-text annotation. If this sentence object is stored under the variable `sent`, its frame annotation with respect to the target *revenge* is accessed as `sent.annotationSet[2]`. (The `?` under *looking* indicates that there is no corresponding LU defined in the SEEKING frame; in some cases the full-text annotators marked but did not define out-of-vocabulary LUs which fit an existing frame. Also, some full-text annotation sets annotate an LU without its FEs—these are shown with `!` to reflect the annotation set's status code of `UNANN`.)

via main method `frame_relation_types()`.

Semantic types. These provide additional semantic categorizations of FEs, frames, and LUs. For FEs, they mark selectional restrictions (e.g., `f.FE['Avenger'].semType` gives the *Sentient* type). Main method `propagate_semtypes()` propogates the FE semantic type labels marked explicitly to other FEs according to inference rules that follow the FE relations. This should be called prior to inspecting FE semtypes (it is not called by default because it takes several seconds to run).

The semantic types are database objects in their own right, and they are organized in their own inheritance hierarchy. Main method `semtypes()` returns all semantic types as a list; main method `semtype()` looks up a particular one by name, ID, or abbreviation; and main method `semtype_inherits()` checks whether two semantic types have a subtype–supertype relationship.

4.5 Corpus Access

Frame-semantic annotations of sentences can be accessed via the `exemplars` and `subCorpus` attributes of an LU object, or via the following main methods:

`annotations``(luname, exemplars, full_text)`
`sents()` **`exemplars`**`(luname)` **`ft_sents`**`(docname)`
`doc``(id)` **`docs`**`(name)`

`annotations()` returns a list of frame **annotation sets**. Each annotation set consists of a frame-evoking **target** (token) within a sentence, the LU in the frame it evokes, its overt FE spans in the sentence, and the status of null-instatiated FEs.[8] Optionally, the user may filter by LU name, or limit by the type of annotation (see next paragraph): `exemplars` and `full_text` both default to `True`. In the XML, the components of an annotation set are stored in several annotation layers: one (and sometimes more than one) layer of FEs, as well as additional layers for other syntactic information (including grammatical function and phrase type labels for each FE, and copular or support words relative to the frame-evoking target).

Annotation sets are organized by sentence. Corpus sentences appear in two kinds of annotation: `exemplars()` retrieves sentences with lexicographic annotation (where a single target has been selected for annotation to serve as an example of an LU); the optional argument allows for filtering the set of LUs. `ft_sents()` retrieves sentences from documents selected for full-text annotation (as many targets in the document as possible have been annotated); the optional argument allows for filtering by document name. `sents()` can be used to iterate over all sentences. Technically, each sentence object contains multiple annotation sets: the first is for sentence-level annotations, including the part-of-speech tagging and in some cases named entity labels; subsequent annotation sets are for

[8] In frame semantics, core FEs that are not overt but are conceptually required by a frame are said to be implicit via null instantiation (Fillmore and Baker, 2009).

frame annotations. As lexicographic annotations have only one frame annotation set, it is visualized in the sentence display: figure 2 shows the display for `f.lexUnit['revenge.n'].exemplars[20]`. Full-text annotations display target information only, allowing the user to drill down to see each annotation set, as in figure 3.

Sentences of full-text annotation can also be browsed by document using the `doc()` and `docs()` methods. The document display lists the sentences with numeric offsets.

5 Limitations and future work

The main part of the Berkeley FrameNet data that the API currently does *not* support are **valence patterns**. For a given LU, the valence patterns summarize the FEs' syntactic realizations across annotated tokens. They are displayed in each LU's "Lexical Entry" report on the FrameNet website.

We intend to add support for valence patterns in future releases, along with more sophisticated querying/browsing capabilities for annotations, and better displays for syntactic information associated with FE annotations. Some of this functionality can be modeled after tools like FrameSQL (Sato, 2003) and Valencer (Kabbach and Ribeyre, 2016). In addition, it is worth investigating whether the API can be adapted for FrameNets in other languages, and to support cross-lingual mappings being added to 14 of these other FrameNets in the ongoing Multilingual FrameNet project.[9]

Acknowledgments

We thank Collin Baker, Michael Ellsworth, and Miriam R. L. Petruck for helping us to understand the FrameNet annotation process and the technical aspects of the data, and for co-organizing the FrameNet tutorial in which an early version of the API was introduced (Baker et al., 2015). We also thank NLTK project leader Steven Bird, Mikhail Korborov, Pierpaolo Pantone, Rob Malouf, and anyone else who may have contributed to the release of the API by reviewing the code and reporting bugs; and anonymous reviewers for their suggestions.

References

Collin Baker, Michael Ellsworth, and Katrin Erk. 2007. SemEval-2007 Task 19: frame semantic structure extraction. In *Proc. of SemEval*, pages 99–104, Prague, Czech Republic.

Collin Baker, Nathan Schneider, Miriam R. L. Petruck, and Michael Ellsworth. 2015. Getting the roles right: using FrameNet in NLP. In *Proc. of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorial Abstracts*, pages 10–12, Denver, Colorado, USA.

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley FrameNet project. In *Proc. of COLING-ACL*, pages 86–90, Montreal, Quebec, Canada.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, Inc., Sebastopol, CA.

Dipanjan Das, Desai Chen, André F. T. Martins, Nathan Schneider, and Noah A. Smith. 2014. Frame-semantic parsing. *Computational Linguistics*, 40(1):9–56.

Charles J. Fillmore. 1982. Frame Semantics. In *Linguistics in the Morning Calm*, pages 111–137. Hanshin Publishing Co., Seoul, South Korea.

Charles J. Fillmore and Collin Baker. 2009. A frames approach to semantic analysis. In Bernd Heine and Heiko Narrog, editors, *The Oxford Handbook of Linguistic Analysis*, pages 791–816. Oxford University Press, Oxford, UK.

Nicholas FitzGerald, Oscar Täckström, Kuzman Ganchev, and Dipanjan Das. 2015. Semantic role labeling with neural network factors. In *Proc. of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 960–970, Lisbon, Portugal.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288.

Alexandre Kabbach and Corentin Ribeyre. 2016. Valencer: an API to query valence patterns in FrameNet. In *Proc. of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*, pages 156–160, Osaka, Japan.

Michael Roth and Mirella Lapata. 2015. Context-aware frame-semantic role labeling. *Transactions of the Association for Computational Linguistics*, 3:449–460.

Josef Ruppenhofer, Michael Ellsworth, Miriam R. L. Petruck, Christopher R. Johnson, Collin F. Baker, and Jan Scheffczyk. 2016. FrameNet II: extended theory and practice.

Hiroaki Sato. 2003. FrameSQL: A software tool for FrameNet. In *Proc. of ASIALEX 2003*, pages 251–258, Tokyo, Japan.

[9] `github.com/icsi-berkeley/multilingual_FN`