API with a Rich Linguistic Resource

Abstract

This paper introduces a novel Python API, incorporated within the NLTK library, that facilitates access to the FrameNet 1.7 lexical database. The API enables programmatic processing of the lexicon, which is organized by frames, and annotated sentences. Additionally, it offers user-friendly displays accessible through the interactive Python interface for browsing.

1 Introduction

This paper delves into the significance of the Berkeley FrameNet project, an endeavor that has been ongoing for over a decade. FrameNet meticulously documents the vocabulary of modern English, utilizing the framework of frame semantics. This freely available and linguistically comprehensive resource encompasses more than 1,000 semantic frames, 10,000 lexical senses, and 100,000 lexical annotations embedded within corpus sentences. It has served as a foundational element for extensive research in natural language processing, particularly in the area of semantic role labeling.

Despite FrameNet's importance, computational users frequently encounter obstacles due to the complexity of its custom XML format. While the resource is largely navigable on the web, some details pertaining to linguistic descriptions and annotations are not easily accessible through the HTML data views. Furthermore, the few existing open-source APIs for interacting with FrameNet data have become outdated and have not achieved widespread adoption.

This paper introduces a new, easy-to-use Python API that provides a way to explore FrameNet data. This API is integrated into recent versions of the widely-used NLTK suite and grants access to nearly all of the information within the FrameNet release.

2 Installation

To install NLTK, please refer to the instructions at nltk.org. NLTK offers cross-platform functionality and is compatible with both Python 2.7 and Python 3.x environments. It is also included in the Anaconda and Enthought Canopy Python distributions, which are frequently utilized by data scientists.

In an active NLTK setup (version 3.2.2 or later), the FrameNet data can be downloaded through a single method call:

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

The data will be installed under the user's home directory by default. Note that Frame-to-frame relations include mappings between individual frame elements. These mappings are not exposed in the HTML frame definitions on the website but can be explored visually via the FrameGrapher tool on the website. Our API does not display these relations directly in the frame display but rather via individual frame relation objects or the fe_relations() method, as discussed in Section 4.4.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

3 Overview of FrameNet

FrameNet is built around conceptual structures called frames. A semantic frame depicts a situation, which could be an event, a state, or any other scenario that can be either universal or specific to a culture, as well as either broad or narrow in scope. The frame identifies participant roles known as frame elements (FEs). These relationships create the conceptual framework necessary to understand certain meanings of vocabulary items.

Some examples include:

- Verbs like buy, sell, and pay, along with nouns like buyer, seller, price, and purchase, are defined within a commercial transaction scenario (frame). Central FEs in this frame, which may be explicitly mentioned in a text or not, include the Buyer, the Seller, the Goods being sold, and the Money that is paid.
- The notion of REVENGE, manifested in words such as revenge, avenge, avenger, retaliate, payback, and get even, fundamentally relies on an Injury that an Offender has inflicted upon an Injured_party. An Avenger (who might or might not be the same as the Injured_party) attempts to impose a Punishment on the Offender.
- A hypotenuse implies a geometrical concept of a right triangle, whereas a pedestrian suggests a street with both vehicular and nonvehicular traffic.

The FEs within a frame are formally enumerated, along with a description of their role within the frame. Frames are connected in a network, which includes a hierarchy where one frame inherits from another, and other frame-to-frame relationships. Vocabulary items that are part of a frame are called lexical units (LUs). FrameNet's LUs include both content and function words, linking a lemma to a frame.

In a text, an LU token is said to evoke the frame. Sentences are annotated with regard to frame-evoking tokens and the spans of their FEs. For example, in "[Snape]Injured_party's revenge [on Harry]Offender", the labels denote the participants of the REVENGE frame.

4 API Overview

4.1 Design Principles

The API is built with these principles in mind:

- **Simplicity:** Access to the main database objects, such as frames, lexical units, and annotations, should be simple, whether through iteration or targeted searches. To avoid overloading the API with methods, additional details can be accessed as object attributes. The help() method provides a synopsis of key database access methods.
- **Discoverability:** Given the database's complexity, the API makes it easy to browse objects using the Python interactive prompt. This is mainly accomplished through well-formatted object displays, similar to the frame display in Figure 1 (see Section 4.3). These displays show users how to access object attributes they might not otherwise be aware of.
- **On-demand loading:** The database is split into many XML files. The FrameNet 1.7 release, once unzipped, is 855 MB. Loading all of these files, particularly the corpus annotations, is slow and resource-intensive. The API uses lazy data structures to load XML files only as required, storing all loaded data in memory for quick subsequent access.

4.2 Lexicon Access Methods

The primary methods for accessing lexicon data are:

- `frames(name)`: returns all frames matching the provided name pattern.
- `frame(nameOrId)`: returns a single frame matching the name or the ID
- `lus(name, frame)`: returns all lexical units matching the provided name pattern.
- `lu(id)`: returns a lexical unit based on its ID

- `fes(name, frame)`: returns all frame elements based on the name pattern provided

Methods with plural names use regular expressions to search entries. Also, the `lus()` and `fes()` methods allow you to specify a frame to constrain the results. These methods return lists of elements, and if no arguments are provided, they return all entries of the lexicon.

Below is an example of a search using the frame name pattern:

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

Here is an example of a search using the 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 used to get an entry by name or ID. A `FramenetError` will be raised when trying to retrieve a non-existent entry.

Two extra methods are available for frame lookups: `frame_ids_and_names(name)` gets a mapping from frame IDs to names and `frames_by_lemma(name)` returns all the frames that have LUs matching the provided name pattern.

4.3 Database Objects

All structured objects like frames, LUs, and FEs are loaded as AttrDict data structures, where keys can be accessed as attributes. For instance:

```
>>> 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
```

The API provides user-friendly displays for important object types, presenting their contents in an organized manner. For example, calling `fn.frame('Revenge')` prints the display for the `REVENGE` frame. These displays indicate attribute names in square brackets.

```
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 perfo
[semTypes] 0 semantic types
[frameRelations] 1 frame relations <Parent=Rewards_and_punishments -- Inheritance -> Child=Reveng
[lexUnit] 18 lexical units avenge.v (6056), avenger.n (6057), get back (at).v (10003), get even.v
[FE] 14 frame elements Core: Avenger (3009), Injured_party (3022), Injury (3018), Offender (3012)
[FEcoreSets] 2 frame element core sets Injury, Injured_party Avenger, Punishment
```

4.4 Advanced Lexicon Access

Frame relations. Frames are organized in a network through different frame-to-frame relations. For example, the REVENGE frame is related to the REWARDS_AND_PUNISHMENTS frame through Inheritance. Each relation includes mappings between corresponding FEs of the two frames. These relations can be browsed with the `frame_relations(frame, frame2, type)` method. Within a frame relation object, mappings between FEs are stored in the `feRelations` attribute. The method `fe_relations()` gives direct access to the links between FEs. The available relation types can be obtained by `frame_relation_types()`.

Semantic types. Semantic types provide added semantic labels for FEs, frames, and LUs. For FEs, they show selectional constraints. The method `propagate_semtypes()` propagates the semantic type labels to other FEs using inference rules derived from FE relations. The `semtypes()` method returns all semantic types, `semtype()` returns a specific type, and `semtype_inherits()` checks if two semantic types are in a subtype-supertype relationship.

4.5 Corpus Access

Frame annotations of sentences are accessible through the `exemplars` and `subCorpus` attributes of a LU object or using the following methods:

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

The `annotations()` method returns a list of frame annotation sets. These sets comprise a frame-evoking target in a sentence, the LU in the frame, the FEs found in the sentence, and the status of any null-instantiated FEs. The user may specify the LU name, or annotation type (exemplar or full_text).

Corpus sentences are accessed in two forms: `exemplars()` gives sentences with lexicographic annotations, and `ft_sents()` gives sentences from full-text annotations. `sents()` provides an iterator over all sentences. Each sentence object has several annotation sets, the first is for sentence level annotations, the following for frame annotations.

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

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

5 Limitations and Future Work

The main FrameNet component that the API does not support right now is valence patterns, which summarize the FE's syntactic realizations across annotated tokens for an LU. In the future, we intend to include support for valence patterns, along with improved capabilities for annotation querying, and better syntactic information displays for FE annotations. Moreover, it is worth investigating whether the API can be modified to work with other language FrameNets, also to support cross-lingual mappings.