The NLTK FrameNet Lexicon API

Nathan Schneider, University of Edinburgh

May 31, 2015 ■ FrameNet Tutorial at NAACL-HLT

Rationale

- Do you ever lie awake at night with questions like:
  - What lemmas are the most ambiguous (have LUs in the most frames)?
  - What frame elements include “location” in their name?
  - What pairs of frames are linked by the *Causative_of* relation?

Rationale

- Do you ever want to access the FrameNet lexicon programmatically from Python?

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

- NLTK (Bird, Klein, & Loper 2009; www.nltk.org) is a Python toolkit for accessing linguistic datasets and running NLP tools
  - Open-source, community-driven, widely used
- NLTK now contains an API for the FrameNet lexicon
  - Contributed by Chuck Wooters & Nathan Schneider in Sep. 2013

Setup

1. Install NLTK.

   - See http://www.nltk.org/install.html

   - Mac/Unix, if you have pip:
     `sudo pip install -U nltk`

2. Invoke interactive Python on the command line and type:
   `import nltk`

   `nltk.download("framenet")`

   This will download FrameNet 1.5 to your NLTK home directory (default: nltk_data in your home directory).

3. In your program (or at the Python prompt), type:
   `from nltk.corpus import framenet as fn`

API Entry Points

- `frames([nameRegex])`
- `frame(exactName)`
- `frames_by_lemma(lemmaRegex)`

- `lus([nameRegex])`

- `fes([nameRegex])`

- `semtypes()`
- `propagate_semtypes()`

- `frame_relations([frame, [frame2,]] [type])`
- `frame_relation_types()`
- `fe_relations()`

Example 1: “noise” frames

```
>>> fn.frames('noise')

[<frame ID=801 name=Cause_to_make_noise>, <frame ID=60
name=Motion_noise>, …]

>>> [f.name for f in fn.frames('noise')]

['Cause_to_make_noise', 'Motion_noise', 'Make_noise',
‘Communication_noise']

>>> [f.name for f in fn.frames('(?i)noise')]

['Cause_to_make_noise', 'Noise_makers',
'Motion_noise', 'Make_noise', 'Communication_noise']
```

Example 2: Noise_makers

```
>>> fn.frame('Noise_makers')

frame (1017): Noise_makers

[definition]
  The Noise_maker is an artifact used to produce sound, especially
  for musical effect.  'The church bells rang' 'The car alarm went
  off again' 'I have never played an acoustic  guitar.' 'Each
  artist personally owns a Steinway and has chosen to perform on
  the Steinway piano professionally'

[semTypes] 0 semantic types

[frameRelations] 1 frame relations
  <Parent=Artifact -- Inheritance -> Child=Noise_makers>
```

Example 2: Noise_makers

```
>>> fn.frame('Noise_makers')

frame (1017): Noise_makers

[definition]
  The Noise_maker is an artifact used to produce sound, especially
  for musical effect.  'The church bells rang' 'The car alarm went
  off again' 'I have never played an acoustic  guitar.' 'Each
  artist personally owns a Steinway and has chosen to perform on
  the Steinway piano professionally'

[semTypes] 0 semantic types

[frameRelations] 1 frame relations
  <Parent Artifact -- Inheritance -> Child=Noise_makers>
```

attributes of the frame: for a frame `f`, access as `f.definition`, etc.

Example 2: Noise_makers

...

```
[lexUnit] 10 lexical units
  alarm.n (11243), bell.n (10211), cello.n (10217), drum.n (10216),
  guitar.n (10210), piano.n (10213), rattle.n (10214), saxophone.n
  (10218), siren.n (10212), xylophone.n (10215)


[FE] 8 frame elements
            Core: Noise_maker (6043)
     Peripheral: Creator (6045), Ground (6050), Material (6049),
Name (6047), Time_of_creation (6046), Type (6048), Use (6044)


[FEcoreSets] 0 frame element core sets
```

Example 2: Noise_makers

... database IDs for direct lookup: `fn.lu(10216)`, etc.

```
[lexUnit] 10 lexical units
  alarm.n (11243), bell.n (10211), cello.n (10217), drum.n (10216)
  guitar.n (10210), piano.n (10213), rattle.n (10214), saxophone.n
  (10218), siren.n (10212), xylophone.n (10215)


[FE] 8 frame elements
            Core: Noise_maker (6043)
      Peripheral: Creator (6045), Ground (6050), Material (6049),
Name (6047), Time_of_creation (6046), Type (6048), Use (6044)


[FEcoreSets] 0 frame element core sets
```

Example 2: Noise_makers

... `f.lexUnit` and `f.FE` are actually `dicts`

```
[lexUnit] 10 lexical units
  alarm.n (11243), bell.n (10211), cello.n (10217), drum.n (10216),
  guitar.n (10210), piano.n (10213), rattle.n (10214), saxophone.n
  (10218), siren.n (10212), xylophone.n (10215)
```

```
[FE] 8 frame elements
            Core: Noise_maker (6043)
      Peripheral: Creator (6045), Ground (6050), Material (6049),
Name (6047), Time_of_creation (6046), Type (6048), Use (6044)
```

```
[FEcoreSets] 0 frame element core sets
```

Example 2: Noise_makers

```
>>> f = fn.frame('Noise_makers')

>>> f.FE
{'Creator': <fe ID=6045 name=Creator>, 'Ground': <fe ID=6050
name=Ground>, 'Material': <fe ID=6049 name=Material>, 'Name': <fe
ID=6047 name=Name>, 'Noise_maker': <fe ID=6043 name=Noise_maker>,
'Time_of_creation': <fe ID=6046 name=Time_of_creation>, 'Type':
<fe ID=6048 name=Type>, 'Use': <fe ID=6044 name=Use>}

>>> f.FE['Noise_maker'].definition
u'This FE identifies the entity or substance that is designed to
produce sound.'

>>> f.lexUnit.keys()
['alarm.n', 'guitar.n', 'rattle.n', 'cello.n', 'drum.n',
'saxophone.n', 'xylophone.n', 'piano.n', 'siren.n', 'bell.n']
```

About the FrameNet API

- Pretty-print displays so the lexicon is **browsable** in interactive Python.

- Data structures echo the lexicon structure.
  - Each frame, FE, LU, semantic type, and frame relation is a structured object.

- FrameNet is stored in *a lot* of XML files. Under the hood, the API loads things lazily and caches them in memory.
  - When a program first iterates over all frames, it will take a few seconds.

Pretty{List,Dict}

```
>>> fn.frames('noise')
[<frame ID=801 name=Cause_to_make_noise>, <frame ID=60
name=Motion_noise>, ...]
>>> type(fn.frames('noise'))
<class 'nltk.corpus.reader.framenet.PrettyList'>
```

- `PrettyList` does 2 things: limits the number of elements shown, and suppresses printing of their full details
  - Otherwise, it is just a `list`
- Similarly, `PrettyDict` suppresses printing of its values' details

Ex 3: “location” FEs

```
>>> fn.fes('location')

[<fe ID=82 name=Location>, <fe ID=4532
name=Location_of_appearance>, ...]

>>> {fe.name for fe in fn.fes("location")}

set(['Location_of_tester', 'Location_of_perceiver',
'Orientational_Location', 'Location', 'Normal_location',
'Relative_location', 'Location_of_source', 'Body_location',
'Host_location', 'Location_of_confinement',
'Location_of_representation', 'Location_of_participant',
'Location_of_sound_source', 'Constant_location',
'Holding_Location', 'Location_of_communicator',
'Location_of_expressor', 'Useful_location',
'Location_of_inspector', 'Connected_locations',
'Fixed_location', 'Location_of_protagonist',
'Holding_location', 'Target_location', 'Location_of_Event',
'Undesirable_location', 'Location_of_appearance'])
```

Ex 3: “location” FEs

```
>>> for fe in fn.fes("location"):
...   print(fe.frame.name+'.'+fe.name)
...

Posture.Location
Coming_up_with.Location_of_appearance
Perception_active.Location_of_perceiver
Appearance.Location_of_perceiver
Make_noise.Location_of_source
Perception_experience.Location_of_perceiver
Adorning.Location
Containers.Relative_location
Dimension.Location
Residence.Location
Observable_body_parts.Orientational_Location
Locale_by_ownership.Relative_location
Hair_configuration.Location
```

Ex 4: frame relations

```
>>> fn.frame_relations(type='Causative_of')

[<Causative=Apply_heat -- Causative_of -> Inchoative/
state=Absorb_heat>, <Causative=Attaching -- Causative_of ->
Inchoative/state=Inchoative_attaching>, ...]
```

```
>>> fn.frame_relations(frame='Make_noise', type='Causative_of')

[<Causative=Cause_to_make_noise -- Causative_of -> Inchoative/
state=Make_noise>]

>>> fn.frame_relations(frame='Cause_to_make_noise',
frame2='Transitive_action')

[<Parent=Transitive_action -- Inheritance ->
Child=Cause_to_make_noise>]
```

Ex 5: *-ish* adjective LUs

```
>>> for lu in fn.lus('ish.a$'):
...   print(lu._short_repr())
...
<lu ID=14144 name=feverish.a>
<lu ID=7238 name=peckish.a>
<lu ID=12391 name=finnish.a>
<lu ID=397 name=foolish.a>
<lu ID=14898 name=turkish.a>
<lu ID=9013 name=lavish.a>
<lu ID=6590 name=boorish.a>
<lu ID=10074 name=youngish.a>
<lu ID=10075 name=oldish.a>
<lu ID=6558 name=churlish.a>
<lu ID=13887 name=irish.a>
```

Ex 6: ambiguous lemmas

```
>>> all_lus = fn.lus()
>>> all_lus[0].name

'cause.v'

>>> from collections import Counter
>>> c = Counter([lu.name for lu in all_lus])

>>> c.most_common(10)

[('in.prep', 10), ('rise.v', 9), ('make.v', 9), ('swing.v', 9), ('cut.v', 8),
('cool.a', 8), ('tie.v', 8), ('take.v', 8), ('get.v', 8), ('call.v', 8)]

>>> ' '.join(f.name for f in fn.frames_by_lemma('call'))

'Request Memory Being_named Contacting Cause_to_start Body_mark Domain
Deserving Claim_ownership Evoking Temporal_pattern Remembering_experience
Labeling Referring_by_name Simple_naming Visiting'
```

Ex 7: frames with most FEs

```
>>> c = Counter({f.name: len(f.FE) for f in fn.frames()})

>>> c.most_common(10)

[('Traversing', 32), ('Quitting_a_place', 26), ('Setting_out', 26),
('Cause_harm', 25), ('Intentional_traversing', 25),
('Cause_bodily_experience', 25), ('Travel', 25), ('Invading', 25),
('Operate_vehicle', 24), ('Departing', 24)]

>>> c = Counter(fe.frame.name for fe in fn.fes() if fe.coreType=='Core')

>>> c.most_common(10)

[('Education_teaching', 11), ('Motion_scenario', 10),
('Performers_and_roles', 10), ('Rite', 9), ('Cause_motion', 9),
('Behind_the_scenes', 9), ('Change_position_on_a_scale', 8),
('Traversing', 8), ('Cotheme', 8), ('Passing', 8)]
```

Conclusion

- NLTK provides a nice API for the FrameNet lexicon
  - Setup is easy

    www.nltk.org
  - Great for interactive browsing
  - Also great for using within programs
- Now you can sleep at night!
- No nice API for the annotations…yet!

Coming Attractions

- API for annotations is in the works! Will support FN 1.6.

- **Lexicographic exemplar sentences** stored in LU object: `lu.exemplars` (all), `lu.subCorpus` (organized by subcorpus)

- **Full-text sentences** stored in full-text document object: `doc.sentence`; individual frame annotations stored in `sentence.annotationSet`

- **New entry points:** `sents()`, `exemplars([luNameRegex])`, `ft_sents([docNameRegex])`, `annotations([luNameRegex])`, `docs([docNameRegex])`

Preview: Exemplar Sentence

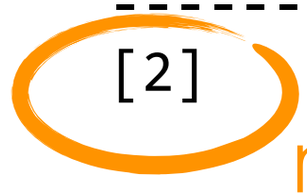
```
>>> fn.exemplars('revenge')[6]

exemplar sentence (929673):

[sentNo] 0
[aPos] 59057935
[LU] (6066) revenge.v in Revenge
[annotationSet] 2 annotation sets
[POS] 9 tags
[POS_tagset] BNC
[GF] 1 relations
[PT] 1 phrases

[text] + [Target] + [FE] + [FE2]

` This poor murdered girl must be revenged .
  ----------------------          ********
  Injured_party                              [Avenger:CNI,
            --------
            Injury

Offender:INI, Punishment:INI]
```

Preview: Exemplar Sentence

```
>>> fn.exemplars('revenge')[6].FE
# first FE layer (machine-readable)

([(2, 25, 'Injured_party')], {'Offender': 'INI', 'Avenger': 'CNI',
'Punishment': 'INI'})

>>> fn.exemplars('revenge')[6].POS

[(0, 1, 'PUQ'), (2, 6, 'DT0'), (7, 11, 'AJ0'), (12, 20, 'AJ0'), (21, 25,
'NN1'), (26, 30, 'VM0'), (31, 33, 'VBI'), (34, 42, 'VVN'), (43, 44, 'PUN')]

>>> fn.exemplars('revenge')[6].annotationSet[0]
# annotationSet[0] for a human-readable display of POS

POS annotation set (1313221) BNC in sentence 929673:

`   This poor murdered girl must be  revenged .
-   ---- ---- -------- ---- ---- --  -------- -
PUQ DT0  AJ0  AJ0      NN1  VM0  VBI VVN      PUN
```

Preview: Full-Text Sentence

```
>>> fn.ft_sents()[3]

full-text sentence (4097611) in NorthKorea_NuclearCapabilities:

[POS] 14 tags

[POS_tagset] PENN

[text] + [annotationSet]

Therefore , obtaining reliable open source information on such
           --------- -------- ---- ------ -----------
           [1]       [6]      [5]         [2]

programs is very challenging .
            -----------
            [4]
```



next slide

Preview: Full-Text Sentence

```
>>> fn.ft_sents()[3].annotationSet[2]

annotation set (6528785):

[LU] (12537) information.n in Information
[GF] 2 relations
[PT] 2 phrases

[text] + [Target] + [FE]

Therefore , obtaining reliable open source information on such
                                  ---------- ----------- --------
                                  Means_of_ga Information Topic

programs is very challenging .
--------

(Means_of_ga=Means_of_gathering)
```

target/denoted FE