LING/C SC 581:
Advanced Computational Linguistics

Lecture 21

Prof. Sandiway Fong

Today's Topic

- Case study (contd.):
  - looking at the verb frames of *break*
  - *continue our live programming*
- Homework 10
  - *you get to finish off the programming tasks!*

Last time

```
(VP (VB break) (ADVP (IN through)) (SBAR-LOC …))
(VP (VBD broke) (NP (DT the) (NN language) (NN barrier)))
(VP (VBN broken) (PP (IN with) (NP (PRP$ his) (NNS parents)))(SBAR-PRP
(VP (VBD broke) (PRT (RP down)) (PP (IN into) (NP …)))
(VP (VBD broke) (PRT (RP up)))
(VP (VBG breaking) (NP (DT the) (JJ thick) (NN sod)))
(VP (VB break) (PRT (RP up)) (NP (NP (DT the) (NN whole)) (PP …))))
(VP (VB break) (NP (-NONE- *T*-2)))
(VP (VBP break))
…
```

- **Catalog by type:**

```
break ADV SBAR-LOC
break NP
break PP SBAR-PRP
break PRT PP
break PRT
break NP
break PRT NP
break NP
break
```

- *How many verb frames are there*?

```
>>> len(break_vfs)
224
```

Recall `pickle.dump()/.load()`

- Save results:

```
>>> import pickle
>>> with open("frames.pkl",'wb') as f:
...     pickle.dump(break_vfs,f)
...
```

- Read pickled data:

```
>>> with open("frames.pkl",'rb') as f:
...     break_vfs = pickle.load(f)
```

b = binary
Python-specific format (not JSON)

Example:

```
>>> len(break_vfs)
224
>>> break_vfs[0]
Tree('VP', [Tree('VB', ['break']), Tree('ADVP', [Tree('IN', ['through'])]), Tree('SBAR-
LOC', [Tree('WHADVP-2', [Tree('WRB', ['where'])]), Tree('S', [Tree('NP-SBJ', [Tree('NP',
[Tree('DT', ['some']), Tree('NN', ['feature'])]), Tree('PP', [Tree('IN', ['of']),
Tree('NP', [Tree('DT', ['the']), Tree('NN', ['winter']), Tree('NN',
['landscape'])])])]), Tree('VP', [Tree('MD', ['would']), Tree('VP', [Tree('VB', ['be']),
Tree('ADJP-PRD', [Tree('JJ', ['recognizable'])]), Tree('ADVP-LOC', [Tree('-NONE-',
['*T*-2'])])])])])])])])
```

Catalog *break* verb frames

- **Examples:**

```
(VP (VBD broke) (NP (DT the) (NN language) (NN barrier)))
break NP
(VP (VBN broken) (PP (IN with) (NP (PRP$ his) (NNS parents)))(SBAR-PRP …))
break PP SBAR
```

- **Idea**: for each `break_vfs` frames
  - print *break* when we have any inflectional form
  - print phrase label of each sister of *break*
- Let's write the Python code!
  - first, let's do the print version
  - then, could collect the strings into a list

Catalog *break* verb frames: step 1

```
>>> for t in break_vfs:
...     for i in range(len(t)):
...             print(t[i].label(), end=" ")
...     print()
...
VB ADVP SBAR-LOC
VBD NP
VBN PP SBAR-PRP
VBD PRT PP
VBD PRT
VBG NP
VB PRT NP
VB NP
VBP
VBG , SBAR-ADV
```

- Next:

VB
VBD
VBN
VBG
VBP

- should come out as *break*

```
VBD PRT
VBD PP
VBD PRT NP
VBN NP PP
VB NP
VBD PRT NP
VBD PRT ADVP-TMP
VB ADVP PP
VB S
VBZ PP
VB PP ADVP
VBD NP PP
VB NP
VBZ PP
…
```

Catalog *break* verb frames: step 2

- Define subtree label printing:
  - write a function `plabel(t)`
    - print *'break*' if `t.label().startswith('VB') and t[0] in verbforms`
    - otherwise print `t.label()`
- Call this string the "verb frame signature"

```
break ADVP SBAR-LOC
break NP
break PP SBAR-PRP
break PRT PP
break PRT
break NP
break PRT NP
break NP
break
```

```
break , SBAR-ADV
break PRT
break PP
break PRT NP
break NP PP
…
```

Catalog *break* verb frames: step 3

- Refine the subtree label `plabel2()` to omit subtags (–LOC etc)

```
for vf in break_vps:
   for i in range(len(vf)):
      print(plabel2(vf[i]),
  end="")
   print()
break ADVP SBAR
break NP
break PP SBAR
break PRT PP
break PRT
break NP
break PRT NP
break NP
```

```
break
break , SBAR
break PRT
break PP
break PRT NP
break NP PP
break NP
break PRT NP
break PRT ADVP
break ADVP PP
break S
break PP
break PP ADVP
```

```
break NP PP
break NP
break PP
break NP
break PRT PP , PP
VBP CC break NP PP
break NP
`` break '' CONJP VBZ SBAR
…
```

Homework 10: Part 1

- There are four tasks: **T1-T4**

Homework 10

HW10 begins here!

`t.label()`

```
VB ADVP SBAR-LOC
VBD NP
VBN PP SBAR-PRP
VBD PRT PP
VBD PRT
VBG NP
VB PRT NP
VB NP
VBP
VBG , SBAR-ADV
VBD PRT
VBD PP
VBD PRT NP
VBN NP PP
VB NP
VBD PRT NP
VBD PRT ADVP-TMP
VB ADVP PP
VB S
```

`plabel(t)`

```
break ADVP SBAR-LOC
break NP
break PP SBAR-PRP
break PRT PP
break PRT
break NP
break PRT NP
break NP
break
break , SBAR-ADV
break PRT
break PP
break PRT NP
break NP PP
break NP
break PRT NP
break PRT ADVP-TMP
break ADVP PP
break S
```

```
'break ADVP SBAR'
'break NP'
'break PP SBAR'
'break PRT PP'
'break PRT '
'break NP '
'break PRT NP '
'break NP '
'break '
'break , SBAR '
'break PRT '
'break PP '
'break PRT NP '
'break NP PP '
'break NP '
'break PRT NP '
'break PRT ADVP '
'break ADVP PP '
'break S '
```

Homework 10

- Tasks:
  - *continue the code development from today and last time*
- Instead of just printing a label at a time, create a *string* for each verb frame:

```
'break ADVP SBAR'
'break NP'
'break PP SBAR'
```

call this a signature.

Hint: `' '.join(list)`

Homework 10

- **T1**: collect *signature strings* into a list.

```
>>> break_sigs
['break ADVP SBAR', 'break NP', 'break PP SBAR', 'break PRT PP', 'break PRT', 'break NP', 'break PRT NP', 'break NP'
, 'break', 'break , SBAR', 'break PRT', 'break PP', 'break PRT NP', 'break NP PP', 'break NP', 'break PRT NP', 'brea
k PRT ADVP', 'break ADVP PP', 'break S', 'break PP', 'break PP ADVP', 'break NP PP', 'break NP', 'break PP', 'break
NP', 'break PRT PP , PP', 'VBP CC break NP PP', 'break NP', "`` break '' CONJP VBZ SBAR", 'break PP ADVP', 'break PR
T NP', 'break PRT PP ADVP', 'break ADVP', 'break ADVP', 'break NP', 'break NP PRT', 'break PP', 'break PP', 'break N
P', 'break PRT NP PP', 'break NP ADVP', 'break NP S', 'break PP', 'ADVP break PRT ADVP', 'break NP ADVP', 'break NP
PP', 'break NP', 'break NP', 'break PP , S', 'break PRT', 'break PRT PP SBAR', 'break NP', 'break NP', 'break NP', '
break PP PP', 'break NP', 'break PP', 'break NP', 'break NP', 'break ADVP', 'break NP PRT', 'break PP , S', 'break P
P PP', 'break NP PP', 'break S PP , VP', 'break PP PP', 'break PP', 'break NP PP', 'break NP', 'break PP NP', 'break
 PRT , S', 'break PRT PP', 'break PRT ADVP : `` NP', 'break ADVP', 'break NP', 'break NP PP', 'break NP S , S', 'bre
ak NP PP', 'break PRT NP', 'break PRT , S', 'break PRT PP', 'break NP PP', 'ADVP break NP', 'break NP', 'break PRT S
', 'break ADVP NP PP', 'break NP', 'break NP', 'break ADVP', 'break NP ADVP', 'break PRT', 'break NP', 'break NP', '
```

Homework 10

- **T2**: count the number of occurrences using `nltk.FreqDist()`
  - https://www.nltk.org/api/nltk.probability.FreqDist.html

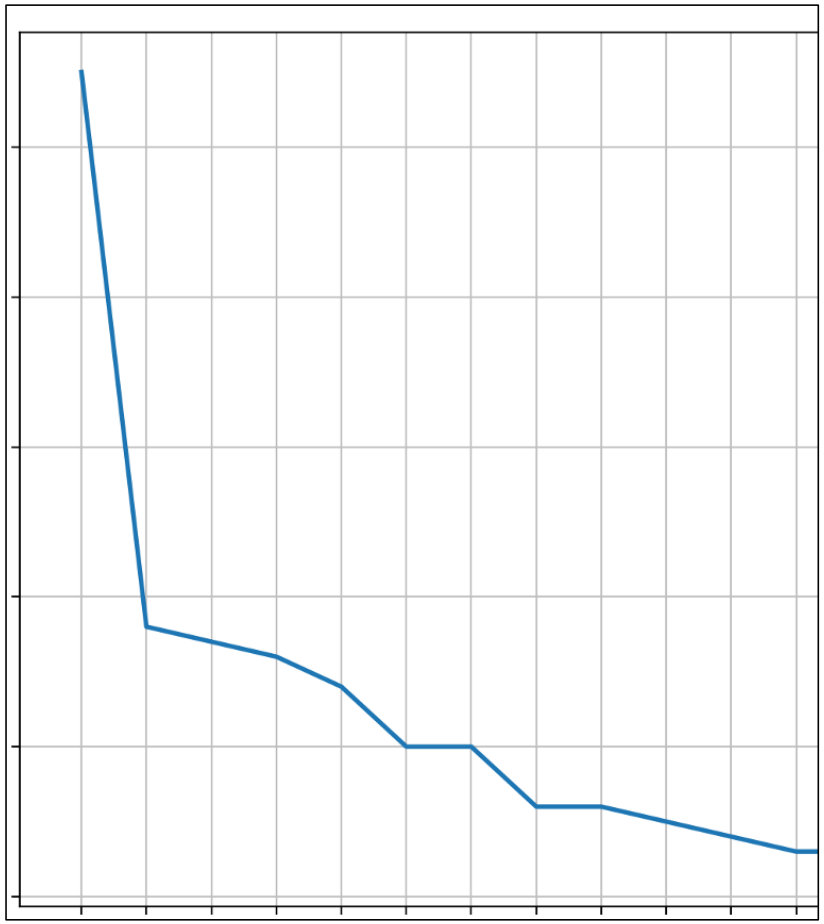
```
>>> import nltk
>>> fd = nltk.FreqDist(break_sigs)
>>> fd
FreqDist({'break NP': … 'break PP PP': ▒, ...})
```

Homework 10

- **T3**: give the top 10 most frequent *signatures* (and values)
  - e.g. '`break NP`' occurs *X* times
  - e.g. use `fd.most_common()` or `fd.tabulate()`

Homework 10

- **T4**: plot the top-20 graph



Homework 10: Part 2

- Extra Credit Task 5:
  - Use the Stanza parser in nltk to parse the PTB *break* sentences
  - Compare the distribution of verb frame signature strings to that obtained from the PTB.

Homework 10: Part 2

- Procedure:
  1. from `break.py` we have the list `break_trees`
  2. convert trees into words, excluding those with POS tag `-NONE-`
  3. `' '.join()` those words into a sentence string
  4. run the Stanza constituency parser on the sentence strings
  5. Use the code developed in class to extract the Stanza *break* verb phrases (*just like we did for the PTB*)

Homework 10: Part 2

- Example for Task 5: steps 1-5

```
>>> s = ' '.join([tuple[0] for tuple in break_trees[0].pos() if tuple[1] != '-NONE-'])
'Dr. H.V. Hilprecht , Professor of Assyrian at the University of Pennsylvania , dreamed that a
  Babylonian priest , associated with the king Kurigalzu , -LRB- 1300 B.C. -RRB- escorted him to
  the treasure chamber of the temple of Bel , gave him six novel points of information about a
  certain broken relic , and corrected an error in its identification .'
>>> import stanza
>>> stanza.download('en')
>>> nlp = stanza.Pipeline(lang='en', processors='tokenize,pos,constituency')
>>> doc = nlp(s)
>>> t = doc.sentences[0].constituency
(ROOT (S (NP (NP (NNP Dr.) (NNP H.V.) (NNP Hilprecht)) (, ,) (NP (NP (NNP Professor)) (PP (IN of)
  (NP (NNP Assyrian))) (PP (IN at) (NP (NP (DT the) (NNP University)) (PP (IN of) (NP (NNP
  Pennsylvania)))))) (, ,)) (VP (VBD dreamed) (SBAR (IN that) (S (NP (NP (DT a) (JJ Babylonian) (NN
  priest)) (, ,) (VP (VP (VBN associated) (PP (IN with) (NP (NP (DT the) (NN king)) (NP (NNP
  Kurigalzu) (, ,) (HYPH -) (NNP LRB) (HYPH -) (CD 1300) (NNP B.C.) (HYPH -) (NNP RRB))))) (, -)
  (VP (VBD escorted) (NP (PRP him)) (PP (IN to) (NP (NP (DT the) (NN treasure) (NN chamber)) (PP
  (IN of) (NP (NP (DT the) (NN temple)) (PP (IN of) (NP (NNP Bel)))))))) (, ,)) (VP (VP (VBD gave)
  (NP (PRP him)) (NP (NP (CD six) (JJ novel) (NNS points)) (PP (IN of) (NP (NP (NN information))
  (PP (IN about) (NP (DT a) (JJ certain) (VBN broken) (NN relic)))))) (, ,) (CC and) (VP (VBD
  corrected) (NP (NP (DT an) (NN error)) (PP (IN in) (NP (PRP$ its) (NN identification)))))))) (.
  .)))
```

Homework 10: Part 2

- **Note**:
  - Stanza trees are not the same as nltk trees!
  - Given tree `t`:
    - `t.label` vs. `t.label()`
    - `t.children[0]` vs. `t[0]`
    - `len(t.children)` vs. `len(t)`
  - *no convenient* `t.subtrees()`!
- I've provided code `task5.py`
  - defines generator functions `subtree(t)` and `subtrees(t)` for Stanza trees

```
# For Stanza trees in HW10 Task 5
#   Sandiway Fong, University of Arizona

def subtree(t):
    """generator for subtrees of t"""
    for i in range(len(t.children)):
        yield t.children[i]

def subtrees(t):
    """recursively generate all subtrees of t"""
    for t2 in subtree(t):
        yield t2
        yield from subtrees(t2)

def plabel(t):
    if t.label in verbtags and t.children[0].label in verbforms:
        return 'break'
    else:
        return t.label
```

Homework 10: Part 2

- **Workflow**:

```
$ python3 -i hw10ec.py
13 (s), sents: 244          initialize pipeline, extracts sentences
93 (s), stanza_trees: 244   parse sentences
s_fs: 236                   get all frames
s_vfs: 216                  limit to verb frames only
s_sigs: 216                 convert to signatures
```

```
>>> s_sigs
['break PP SBAR', 'break NP', 'break PP PP S', 'break ADVP PP', 'break PRT', 'break NP', 'break PRT NP', 'break NP',
 'break NFP NP', 'break PRT', 'break PP', 'break PRT NP', 'break NP PP', 'break NP', 'break PRT NP', 'break PRT NP',
 'break ADVP', 'break NP S', 'break PP', 'break PP NFP NP', 'break NP', 'break PP', 'break NP', 'break PRT PP , PP',
 'VBP CC break NP PP', 'break NP', 'break PP ADVP', 'break PRT NP', 'break PRT PP', 'break NFP NP', 'break ADVP', 'b
reak NP', 'break NP PRT', 'break PP', 'break PP', 'break NP', 'break PRT NP PP', 'break NP NFP NP', 'break', 'break
```

Homework 10

- Submit to sandiway@arizona.edu
- SUBJECT: 581 Homework 10 *YOUR NAME*
- One PDF file (for grading)
  - include your screenshots
  - include your Python code
- Usual Deadline:
  - midnight Saturday
  - to be grade Sunday
  - we will review the homework next Monday