Handout 13: Information extraction

1. Tabular data (relational database)

a. Input

> **Atlas Widgets, Inc.** announced on **Friday** that **Constance White** would take the helm on **March 1**. **Ms. White** will replace **Milifred Peirce** as CEO.

b. Relation **office**:

org	office	holder	begin	end
O-49	CEO	P-0155	2008 Mar 1	NA
O-49	CEO	P-0023	NA	2008 Mar 1

c. Relation **person**:

id	first	last
P-0023	Milifred	Peirce
P-0155	Constance	White

d. Relation **organization**:

id	name
O-49	Atlas Widgets, Inc.

2. Queries

a. Which organizations are located in Atlanta?

```
>>> ans = [org for (ent, loc) in location_table if loc == 'Atlanta']
```

b. Who is/was CEO of Atlas Widgets?

```
>>> orgname = dict(organization_table)
>>> pername = dict((id, first + ' ' + last)
                   for (id, first, last) in person_table)
>>> ans = [pername[per]
           for (org, ofc, per, _, _) in office_table
           if ofc == 'CEO' and orgname[org].startswith('Atlas Widgets')]
```

3. Components

a. Text classification; ad hoc retrieval. Which documents contain useful information?

b. General-purpose preprocessing: tokenization, sentence segmentation, part of speech tagging

c. (Named) entity extraction ($\approx$ passage retrieval). Boldface phrases in example.

d. Relation recognition. Underlined phrases in example.

e. Coreference. Recognizing that “Constance White” and “Ms. White” are the same person.

f. Integration: pulling together pieces of an event possibly from multiple sentences.

4. Preprocessing

```
>>> nltk.sent_tokenize('Hi there.  This is a test.')
['Hi there.', 'This is a test.']
>>> [nltk.word_tokenize(s) for s in _]
[['Hi', 'there', '.'], ['This', 'is', 'a', 'test', '.']]
>>> [nltk.pos_tag(s) for s in _]
[[('Hi', 'NNP'), ('there', 'EX'), ('.', '.')], [('This', 'DT'), ('is',
'VBZ'), ('a', 'DT'), ('test', 'NN'), ('.', '.')]]
```

5. Partial parsing (chunk parsing)

a. Example sentence

```
>>> s = [('The', 'DT'), ('little', 'JJ'), ('dog', 'NN'), ('likes', 'VBZ'),
...      ('smelly', 'JJ'), ('shoes', 'NN')]
```

b. Grammar

```
>>> g = 'NP: {<DT>?<JJ>*<NN.*>+}'
```

c. Parser

```
>>> p = nltk.RegexpParser(g)
```

d. Tree

```
>>> t = p.parse(s)
>>> t
Tree('S', [Tree('NP', [('The', 'DT'), ('little', 'JJ'), ('dog',
'NN')]), ('likes', 'VBZ'), Tree('NP', [('expensive', 'JJ'), ('shoes',
'NNS')])])
>>> print(t)
(S (NP The/DT little/JJ dog/NN) likes/VBZ (NP expensive/JJ shoes/NNS))
```

6. Refinements

a. Multiple rules can be specified; applied in sequence

```
g = r'''
  NP: {<DT>?<JJ>*<NN>}
      {<NNP>+}
'''
```

b. Chinks: things that break up a chunk

7. Longest-match rule

the emergency crews hate most is domestic violence

8. Using a classifier to recognize chunks

B	O	B	I	B	I	I
PRP	VBD	DT	NN	DT	JJ	NN
He	gave	the	dog	a	new	toy

9. Evaluation

a. Test sentences

```
>>> from nltk.corpus import conll2000 as conll
>>> test = conll.chunked_sents('test.txt', chunk_types='NP')
```

b. A chunk parser

```
>>> p = nltk.RegexpParser(r'NP: {<[CDJNP].*>+}')
```

c. Evaluate the parser

```
>>> print(p.evaluate(test))
ChunkParse score:
    IOB Accuracy:  87.7%
    Precision:     70.6%
    Recall:        67.8%
    F-Measure:     69.2%
```

d. Why don't we measure chunking accuracy?

e. F-measure: harmonic mean of precision, recall

$$\frac{1}{\frac{1}{2}\left(\frac{1}{P}+\frac{1}{R}\right)}$$

Equals regular mean when $P = R$, but penalizes $P > R$ or $R > P$

10. Relation extraction: quick and dirty

a. Consider adjacent pairs of entities, classify based on the words between

```
[PER Ms. White] will replace [PER Milifred Peirce]
```

b. Single entities with preceding or following text: relation to larger event

```
[COM Atlas Widgets, Inc.] announced
on [DATE March 1]
```

c. Keywords not associated with adjacent entity: property of larger event

```
as CEO
```

11. Higher-quality approach

  a. Parse, classify relationships between entities based on relation in parse tree

  b. Do full interpretation. But recall usually falls as precision improves.