WIRTSCHAFTS
UNIVERSITÄT
WIEN VIENNA
UNIVERSITY OF
ECONOMICS
AND BUSINESS

Extracting Information from Text

Research Seminar
Statistical Natural Language Processing

Angela Bohn, Mathias Frey,
November 25, 2010

Main goals

- Extract structured data from unstructured text
- Training & Evaluation
- Identify entities and relationships described in text (i.e. named entity recognition and relation extraction)

Structured vs. Unstructured Data

Which organizations are located in Atlanta?
Querying a database would be easy:

```sql
SELECT *
  FROM organization
 WHERE UPPER(location) LIKE '%ATLANTA%';
```

... whereas the real world looks like:

```
..., said Ken Haldin, a spokesman for
Georgia-Pacific in Atlanta
```

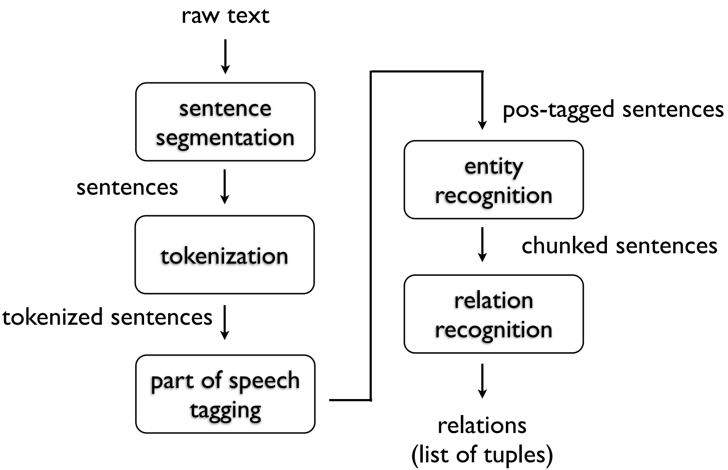

Figure: Simple pipeline, BKL, 2009. NLP with Python. p 263

Chaining NLTK's functions together

```
>>> def ie_preprocess(document):
...     s = nltk.sent_tokenize(document)
...     s = [nltk.word_tokenize(sent) for sent in s]
...     s = [nltk.pos_tag(sent) for sent in s]
...     return s
>>> document = """except for Australian Prime Minister
...     Julia Gillard, whose red-and-white dirndl dress
...     seemed more reminiscent of the Austrian Alps
...     than the outback."""
>>> ie_preprocess(document)
[[ ...  ('except', 'IN'), ('for', 'IN'),
('Australian', 'JJ'), ('Prime', 'NNP'),
('Minister', 'NNP'), ('Julia', 'NNP'), ... ]]
```

Usage of R and openNLP

Import library and try a sentence detection.

```
> library(openNLP)
> library(openNLPmodels.en)
> s <- "The little yellow dog barked at the cat."
> s <- sentDetect(s, language = "en")
> s
```

```
[1] "The little yellow dog barked at the cat."
```

POS tagging with **tagPOS()**. Mind the dependence on the input language.

```
> t <- tagPOS(s, language = "en")
> t

[1] "The/DT little/JJ yellow/JJ dog/NN barked/VBN at/IN the/DT cat./NN"
```

Chunking

- Segment and label multitoken sequences w'o overlaps
- aka "Partial parsing"
- Pipeline is prerequisite of chunking
  - sentence segmentation
  - tokenization
  - POS-tagging

Chunking Techniques

NLTK's chunkers depend on

- Regular Expressions
- Unigrams, Bigrams, n-Grams
- Classifier + Feature Extractor

```
>>> sentence = [("the","DT"), ("little","JJ"),
... ("yellow","JJ"), ("dog","NN"),("barked",
... "VBD"),("at","IN"),("the","DT"),("cat","NN")]
>>> grammar = "NP:␣{<DT>?<JJ>*<NN>}"
>>> cp = nltk.RegexpParser(grammar)
>>> result = cp.parse(sentence)
>>> result.draw()
```

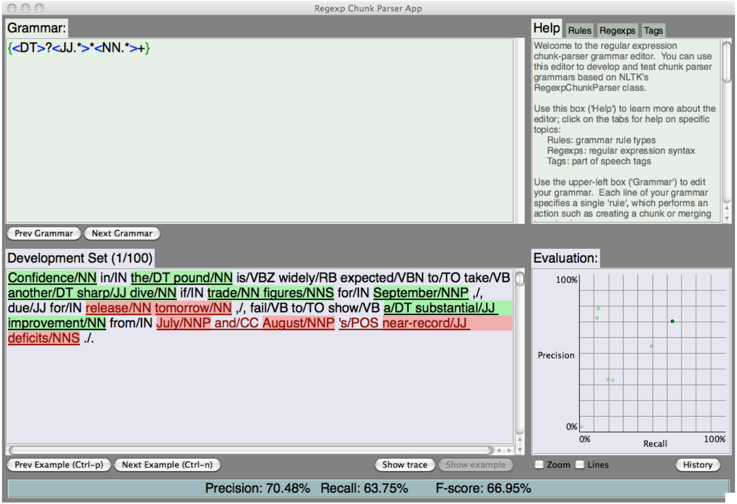

Figure: Tag pattern manipulation with NLTK's chunkparser application.

Exploring text corpora

```
brown = nltk.corpus.brown
>>> def find_cnk(grammar):
...     cp = nltk.RegexpParser(grammar)
...     for sent in brown.tagged_sents():
...         tree = cp.parse(sent)
...         for subtree in tree.subtrees():
...             if subtree.node == 'CHUNK': yield subtree

>>> for t in find_cnk('CHUNK: {<VBN> <TO> <V.*>}'):
...     print t
(CHUNK delighted/VBN to/TO meet/VB)
(CHUNK come/VBN to/TO talk/VB)
(CHUNK used/VBN to/TO express/VB)
(CHUNK given/VBN to/TO understand/VB)
...
```

A chunker in R:

```
>>> sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),
... ("dog", "NN"), ("barked", "VBD"), ("at", "IN"), ("the", "DT"), ("cat", "NN")]
>>> grammar = "NP:␣{<DT>?<JJ>*<NN>}"
>>> cp = nltk.RegexpParser(grammar)
>>> result = cp.parse(sentence)
>>> print result
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```

```
> t

[1] "The/DT little/JJ yellow/JJ dog/NN barked/VBN at/IN the/DT cat./NN"

> npchunker <- function(input) {
+     p <- "(\\<[^[:space:]]+/DT\\> )?((\\<[^[:space:]]+/JJ.?\\> )*)(\\<[^[:space:]]+/NN\\>)"
+     r <- "(NP \\1\\2\\4\\5)"
+     output <- gsub(input, pattern = p, replacement = r)
+     output <- paste("(S ", output, ")", sep = "")
+     output
+ }
> npchunker(t)

[1] "(S (NP The/DT little/JJ yellow/JJ dog/NN) barked/VBN at/IN (NP the/DT cat./NN))"
```

Another chunker in R:

```
grammar = r"""
  NP: {<DT|PP\$>?<JJ>*<NN>}    # chunk determiner/possessive, adjectives and nouns
      {<NNP>+}                 # chunk sequences of proper nouns
"""
cp = nltk.RegexpParser(grammar)
sentence = [("Rapunzel", "NNP"), ("let", "VBD"), ("down", "RP"), [1]
            ("her", "PP$"), ("long", "JJ"), ("golden", "JJ"), ("hair", "NN")]
>>> print cp.parse(sentence) [2]
(S
  (NP Rapunzel/NNP)
  let/VBD
  down/RP
  (NP her/PP$ long/JJ golden/JJ hair/NN))
```

```
> rapunzel <- tagPOS("Rapunzel let down her long golden hair.")
> another.npchunker <- function(input) {
+     rule1 <- "(\\<[^[:space:]]+/(PRP\\$|DT) )((\\<[^[:space:]]+/JJ\\> )*)(\\<[^[:space:]]+/NN\\>)"
+     rule2 <- "(\\<[^[:space:]]+/NNP\\>)+"
+     output <- gsub(input, pattern = rule1, replacement = "(NP \\1\\3\\5)")
+     output <- gsub(output, pattern = rule2, replacement = "(NP \\1)")
+     output <- paste("(S ", output, ")", sep = "")
+     output
+ }
> another.npchunker(rapunzel)

[1] "(S (NP Rapunzel/NNP) let/VB down/RP (NP her/PRP$ long/JJ golden/JJ hair./NN))"
```

Chinking

Chinks are patterns excluded from chunks.

```
grammar = r"""
...     NP:
...     {<.*>+}         # chunk everything
...     }<VBD|IN>+{ # chink VBD and IN
...    """

>>> cp = nltk.RegexpParser(grammar)
>>> print cp.parse(sentence)
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```

A chinker in R:

```
> chinker <- function(input) {
+     p <- "(\\<.*\\>)+ (\\<[^[:space:]]+/VBN\\>) (\\<[^[:space:]]+/IN\\>) (\\<.*\\>)+"
+     r <- "(NP \\1) \\2 \\3 (NP \\4)"
+     output <- gsub(input, pattern = p, replacement = r)
+     output <- paste("(S", output, ")", sep = "")
+     output
+ }
> chinker(t)

[1] "(S(NP The/DT little/JJ yellow/JJ dog/NN) barked/VBN at/IN (NP the/DT cat./NN))"
```

IOB tags

IOB tags are standard way to represent chunk structures in files with

- B marking a token as the beginning,
- I marking a token being inside, and
- O marking a token being outside of a chunk.

```
We PRP B-NP
saw VBD O
the DT B-NP
little JJ I-NP
yellow I-NP
dog NN I-NP
```

IOB Tags

```
> write.IOB <- function(input, file) {
+     output <- npchunker(tagPOS(input))
+     output <- gsub(output, pattern = "^\\(S (.+)\\)$", replacement = "\\1")
+     output <- gsub(output, pattern = "(\\(NP [^\\)]+\\))", replacement = "|\\1|")
+     output <- gsub(output, pattern = "^\\||\\|$", replacement = "")
+     output <- unlist(strsplit(output, split = "[[:space:]]?\\|[[:space:]]?"))
+     output <- strsplit(output, split = " ")
+     annotate <- function(x) {
+         if (length(grep(x, pattern = "\\(NP")) == 0) {
+             y <- paste(gsub(x, pattern = "/", replacement = " "), "O")
+         }
+         if (length(grep(x, pattern = "\\(NP")) > 0) {
+             y <- paste(gsub(x, pattern = "/", replacement = " "), "I-NP")
+             y[2] <- gsub(y[2], pattern = "I-NP", replacement = "B-NP")
+             y <- y[2:length(y)]
+         }
+         y <- gsub(y, pattern = "( [[:upper:]]{2,3})(\\))( [[:upper:]])", replacement = "\\1\\3")
+         y
+     }
+     output <- lapply(output, annotate)
+     unlink(file)
+     cat(unlist(output), file = file, append = TRUE, sep = "\n")
+ }
> write.IOB(s, file = "output.txt")
```

write.IOB()'s output.txt

```
The DT B-NP
little JJ I-NP
yellow JJ I-NP
dog NN I-NP
barked VBN O
at IN O
the DT B-NP
cat. NN I-NP
```

Evaluation against training corpus

Establishing a baseline without a grammar. (Notice that 43.4 % of our evaluation corpus' tokens are outside of chunks.)

```
>>> from nltk.corpus import conll2000 as ev
>>> cp = nltk.RegexpParser("")
>>> test_sents = ev.chunked_sents(
...     'test.txt',chunk_types=['NP'])
>>> print cp.evaluate(test_sents)
ChunkParse score:
    IOB Accuracy:   43.4%
    Precision:       0.0%
    Recall:          0.0%
    F-Measure:       0.0%
```

Evaluation against training corpus (2)

```
>>> grammar = r"NP: {<[CDJNP].*>+}"
>>> cp = nltk.RegexpParser(grammar)
>>> print cp.evaluate(test_sents)
ChunkParse score:
    IOB Accuracy:  87.7%
    Precision:     70.6%
    Recall:        67.8%
    F-Measure:     69.2%
```

Precision, Recall, F-Measure

	NP	! NP
chunked correctly	.	.
! chunked correctly	.	.

Named Entity Recognition (NER)

There are two approaches

- Gazetteer, dictionary
- Classifier

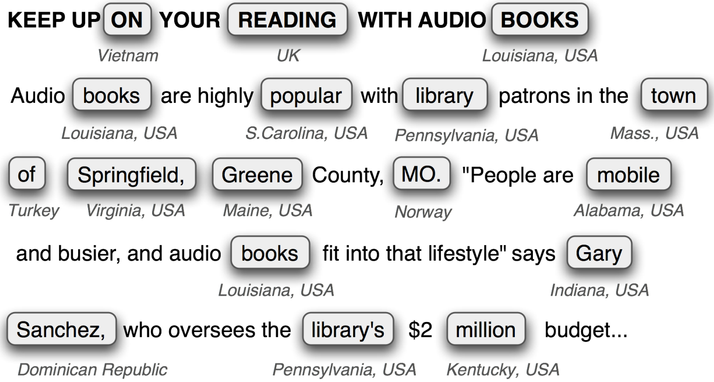

Figure: Error-prone location detection with gazetteer, BKL, 2009. NLP with Python. p 282

Relation extraction

Based on identified named entities, regular expression

Exploring text corpora

```
>>> import re
>>> IN = re.compile(r'.*\bin\b(?!\b.+ing)')
>>> for doc in nltk.corpus.ieer.parsed_docs(
      'NYT_19980315'):
...       for rel in nltk.sem.extract_rels(
            'ORG','LOC', doc, corpus='ieer', pattern=IN):
...           print nltk.sem.show_raw_rtuple(rel)
```

```
[ORG: 'DDB␣Needham'] 'in' [LOC: 'New␣York']
[ORG: 'Kaplan␣Thaler␣Group'] 'in' [LOC: 'New␣York']
[ORG: 'BBDO␣South'] 'in' [LOC: 'Atlanta']
[ORG: 'Georgia-Pacific'] 'in' [LOC: 'Atlanta']
```