Introduction to Natural Language Processing
Phase 2: Question Answering
The plan for the next two weeks

May 14th: MORE Natural Language Understanding for QA
- Chunking (Shallow Parsing vs. Parsing). Read Ch. 7.
- Sentence Structure. READ ch. 8.1-8.3, 8.5
- QA pipeline
- Baseline QA system using string operations and sentence ranking
- Next Steps: Using Syntax
- Stanford Dependency Parse structure VS. Constituent structure
- Dependency Structures and Relations
- Constituent Structures and CFG rules

**Week 8: Question Answering II**

**HOMEWORK 6 is DUE WED NIGHT MAY 20th at 11:55 PM**

May 19th: Question Answering II: Using Syntax

**HW 7 of project**

**Working with NLU representations for Question Answering II**
- Homework 7 assigned. Due Friday, May 29th at 11:55 PM.
- IMPROVING PRECISION.
- Grammars and Parsing
- Constituency and Dependency Tree Readers
- Chunking and Parsing: How to search trees
- Pattern Matching on Dependency Relations
- Ranking possible responses

May 21st:
- Working with NLU representations.
- Syntactic Structure and Coordination
- Prepositional Phrase Attachments
- Dependency vs. Constituent Structures II
- Answering Questions from NLU/parsing representations

**Week 9: Question Answering III: Lexicons & Lexical Semantics**

**Homework 7 DUE. FRIDAY MAY 29th at 11:55 PM.**
Week 9: Question Answering III: Lexicons & Lexical Semantics

Homework 7 DUE. FRIDAY MAY 29th at 11:55 PM.

May 26th: Using VerbNet and WordNet API in QA
- HW8: final HW assigned. The final QA competition.
- Paraphrases and Lexical Choice
- Verbs and their dependents, VerbNet semantic role types
- Verbnet and Wordnet API, how it works
- Word Sense Disambiguation .DICT files provided with HW8.
- Constituent and Dependency Trees, finding subjects etc
- Increasing Precision of Answers

May 28th: Review of all techniques for QA, types of questions, methods
- Review for Final
- Increasing Precision of Answers

Week 10: Question Answering Competition & Final Exam in Class slot

June 2nd: No Lecture. Special Section. 2 to 4 in Class.
- Come to class with your questions. Work on your QA system, prepare for Final.

June 4th: FINAL EXAM in CLASS SLOT.
Homework 6
Baseline System

- Due Wed night May 20th.
- "Easy" questions
  - Don't expect to get close to 100% precision and recall
  - Get something working that returns a reasonable answer for all the questions
- We'll release HW7 next Thursday along with the first development set (2 more fables & 2 more blogs)
- HW7 will include MED questions
- I’m preparing an “advanced dev set” that has more examples of MED and hard questions, not the ones that are used in the heldout test sets.
- Will show examples today.
Recall vs. Precision

- How would you get perfect precision?

- How would you get perfect recall?
Example from my Expts this week

- Experiments on recognizing positive and negative emotions and situations in journal entries.

A person. I know I should not complain about our weather because 90% of the time it is absolutely fabulous...but I live in one of these new green apartment buildings that was designed to keep heat in and to make matters worse I face a courtyard so my apartment cannot even get a breeze. It is a bitch to sleep during the day even with a fan blowing!

Disney. I'm going to Disney today! So excited! Up and time to get ready!
### Example from my Expts this week

Sorted by f

<table>
<thead>
<tr>
<th></th>
<th>rel</th>
<th>freq</th>
<th>prob</th>
<th>thresh</th>
<th>prec</th>
<th>recall</th>
<th>f</th>
<th>acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>pos</td>
<td>2</td>
<td>0.7</td>
<td>1</td>
<td>0.6070529</td>
<td>0.61479592</td>
<td>0.61089987</td>
<td>0.64051522</td>
</tr>
<tr>
<td>3</td>
<td>pos</td>
<td>3</td>
<td>0.7</td>
<td>1</td>
<td>0.64022663</td>
<td>0.57653061</td>
<td>0.60671141</td>
<td>0.65690867</td>
</tr>
<tr>
<td>4</td>
<td>pos</td>
<td>2</td>
<td>0.75</td>
<td>1</td>
<td>0.62637363</td>
<td>0.58163265</td>
<td>0.6031746</td>
<td>0.64871194</td>
</tr>
<tr>
<td>5</td>
<td>pos</td>
<td>4</td>
<td>0.7</td>
<td>1</td>
<td>0.64776119</td>
<td>0.55357143</td>
<td>0.59697387</td>
<td>0.65690867</td>
</tr>
<tr>
<td>6</td>
<td>pos</td>
<td>3</td>
<td>0.75</td>
<td>1</td>
<td>0.66349206</td>
<td>0.53316327</td>
<td>0.59123055</td>
<td>0.66159251</td>
</tr>
<tr>
<td>7</td>
<td>pos</td>
<td>2</td>
<td>0.8</td>
<td>1</td>
<td>0.65</td>
<td>0.53061224</td>
<td>0.58426966</td>
<td>0.65339578</td>
</tr>
<tr>
<td>8</td>
<td>pos</td>
<td>4</td>
<td>0.75</td>
<td>1</td>
<td>0.67687075</td>
<td>0.50765306</td>
<td>0.58017493</td>
<td>0.66276347</td>
</tr>
<tr>
<td>9</td>
<td>pos</td>
<td>5</td>
<td>0.7</td>
<td>1</td>
<td>0.66118421</td>
<td>0.5127551</td>
<td>0.57758621</td>
<td>0.6557377</td>
</tr>
</tbody>
</table>
Example from my Expts this week

Sorted by prec, rec is sad

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>rel</td>
<td>freq</td>
<td>prob</td>
<td>thresh</td>
<td>prec</td>
<td>recall</td>
<td>f</td>
</tr>
<tr>
<td>2</td>
<td>pos</td>
<td>3</td>
<td>0.8</td>
<td>3</td>
<td>1</td>
<td>0.03061224</td>
<td>0.05940594</td>
</tr>
<tr>
<td>3</td>
<td>pos</td>
<td>4</td>
<td>0.8</td>
<td>3</td>
<td>1</td>
<td>0.03061224</td>
<td>0.05940594</td>
</tr>
<tr>
<td>4</td>
<td>pos</td>
<td>5</td>
<td>0.85</td>
<td>3</td>
<td>1</td>
<td>0.0255102</td>
<td>0.04975124</td>
</tr>
<tr>
<td>5</td>
<td>pos</td>
<td>5</td>
<td>0.8</td>
<td>3</td>
<td>1</td>
<td>0.02295918</td>
<td>0.04488778</td>
</tr>
<tr>
<td>6</td>
<td>pos</td>
<td>3</td>
<td>0.85</td>
<td>3</td>
<td>1</td>
<td>0.01530612</td>
<td>0.03015075</td>
</tr>
<tr>
<td>7</td>
<td>pos</td>
<td>4</td>
<td>0.85</td>
<td>3</td>
<td>1</td>
<td>0.01530612</td>
<td>0.03015075</td>
</tr>
<tr>
<td>8</td>
<td>pos</td>
<td>5</td>
<td>0.85</td>
<td>3</td>
<td>1</td>
<td>0.00765306</td>
<td>0.01518897</td>
</tr>
<tr>
<td>9</td>
<td>pos</td>
<td>5</td>
<td>0.8</td>
<td>3</td>
<td>1</td>
<td>0.96153846</td>
<td>0.06377551</td>
</tr>
<tr>
<td>0</td>
<td>pos</td>
<td>3</td>
<td>0.85</td>
<td>3</td>
<td>1</td>
<td>0.95652174</td>
<td>0.05612245</td>
</tr>
<tr>
<td>1</td>
<td>pos</td>
<td>4</td>
<td>0.85</td>
<td>3</td>
<td>1</td>
<td>0.94444444</td>
<td>0.04336735</td>
</tr>
<tr>
<td>2</td>
<td>pos</td>
<td>4</td>
<td>0.8</td>
<td>3</td>
<td>1</td>
<td>0.9375</td>
<td>0.07653061</td>
</tr>
<tr>
<td>3</td>
<td>pos</td>
<td>3</td>
<td>0.8</td>
<td>3</td>
<td>1</td>
<td>0.91891892</td>
<td>0.08673469</td>
</tr>
<tr>
<td>4</td>
<td>pos</td>
<td>3</td>
<td>0.85</td>
<td>2</td>
<td>1</td>
<td>0.89285714</td>
<td>0.06377551</td>
</tr>
<tr>
<td>5</td>
<td>pos</td>
<td>5</td>
<td>0.85</td>
<td>2</td>
<td>1</td>
<td>0.88235294</td>
<td>0.03826531</td>
</tr>
<tr>
<td>6</td>
<td>pos</td>
<td>2</td>
<td>0.85</td>
<td>3</td>
<td>1</td>
<td>0.87804878</td>
<td>0.09183673</td>
</tr>
</tbody>
</table>
Need to give up a lot of prec

Row 36, Row 38

<table>
<thead>
<tr>
<th>pos</th>
<th>3</th>
<th>0.75</th>
<th>2</th>
<th>0.80882353</th>
<th>0.14030612</th>
<th>0.23913043</th>
<th>0.59016393</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>4</td>
<td>0.8</td>
<td>2</td>
<td>0.80851064</td>
<td>0.09693878</td>
<td>0.17312073</td>
<td>0.57494145</td>
</tr>
<tr>
<td>pos</td>
<td>4</td>
<td>0.75</td>
<td>2</td>
<td>0.80645161</td>
<td>0.12755102</td>
<td>0.22026432</td>
<td>0.58548009</td>
</tr>
<tr>
<td>pos</td>
<td>4</td>
<td>0.8</td>
<td>2</td>
<td>0.8045977</td>
<td>0.17857143</td>
<td>0.29227557</td>
<td>0.6030445</td>
</tr>
<tr>
<td>pos</td>
<td>2</td>
<td>0.75</td>
<td>3</td>
<td>0.79487179</td>
<td>0.15816327</td>
<td>0.26382979</td>
<td>0.59484778</td>
</tr>
<tr>
<td>pos</td>
<td>3</td>
<td>0.8</td>
<td>2</td>
<td>0.79207921</td>
<td>0.20408163</td>
<td>0.32454361</td>
<td>0.61007026</td>
</tr>
<tr>
<td>pos</td>
<td>5</td>
<td>0.8</td>
<td>2</td>
<td>0.79012346</td>
<td>0.16326531</td>
<td>0.27061311</td>
<td>0.59601874</td>
</tr>
<tr>
<td>pos</td>
<td>3</td>
<td>0.75</td>
<td>3</td>
<td>0.78947368</td>
<td>0.03826531</td>
<td>0.0729927</td>
<td>0.55386417</td>
</tr>
<tr>
<td>pos</td>
<td>5</td>
<td>0.85</td>
<td>1</td>
<td>0.77707006</td>
<td>0.31122449</td>
<td>0.44444444</td>
<td>0.64285714</td>
</tr>
<tr>
<td>pos</td>
<td>2</td>
<td>0.85</td>
<td>2</td>
<td>0.77669903</td>
<td>0.20408163</td>
<td>0.32323232</td>
<td>0.60772834</td>
</tr>
<tr>
<td>pos</td>
<td>4</td>
<td>0.85</td>
<td>1</td>
<td>0.77653631</td>
<td>0.35459184</td>
<td>0.48686515</td>
<td>0.65690867</td>
</tr>
<tr>
<td>pos</td>
<td>4</td>
<td>0.85</td>
<td>1</td>
<td>0.77358491</td>
<td>0.20918367</td>
<td>0.32931727</td>
<td>0.6088993</td>
</tr>
</tbody>
</table>
Baseline System
Generic QA Architecture

Architecture of Typical Q/A System

- Question
  - Question Typing
    - entity type(s)
  - Answer Identification

- Text(s)
  - Document/Passage Retrieval
    - relevant text(s)
  - Named Entity Tagging
    - tagged text
Baseline System

- To get started do something simple

- **First Steps**
  - Look for lexical overlap between the question and each sentence
  - Rank each sentence by number of words in common
  - Return the one with the highest overlap
Baseline System

Next Steps

- Classify questions into types
- Build simple parsers for each question type:
  - Who, what, when, where, etc.
  - E.g., identify type of expected answer
- Take word order into account
- Stem/lemmatize words
- Look for named entities of the proper type near keywords
- Rephrase question
Beyond this HW

- More advanced (not for this HW)
  - Use the parse trees and dependency graphs
  - Use wordnet to identify synonymous answers
  - Use verbnet to look for specific argument types

- You may want to plan for using parse trees and/or dependency graphs from the start

- Section today and tomorrow will go through example code etc
Additional resources for assignment

- assignment6-stub.py (3 KB; May 9, 2015 11:51 am)
- baseline-stub.py (2 KB; May 9, 2015 11:51 am)
- chunk-demo.py (4 KB; May 9, 2015 11:51 am)
- parse-demo.py (3 KB; May 9, 2015 11:51 am)
- constituency-demo-stub.py (3 KB; May 9, 2015 11:51 am)
- dependency-demo-stub.py (3 KB; May 9, 2015 11:51 am)
- score-answers.pl (13 KB; May 9, 2015 11:52 am)
- fables-01_my_answers.txt (1 KB; May 9, 2015 11:52 am)
- hw6_dataset.zip (20 KB; May 9, 2015 11:53 am)
- hw6-v3.pdf (125 KB; May 9, 2015 11:53 am)
### PTB Parts of Speech Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Coordinating conjunction</td>
</tr>
<tr>
<td>CD</td>
<td>Cardinal number</td>
</tr>
<tr>
<td>DT</td>
<td>Determiner</td>
</tr>
<tr>
<td>EX</td>
<td>Existential <em>there</em></td>
</tr>
<tr>
<td>FW</td>
<td>Foreign word</td>
</tr>
<tr>
<td>IN</td>
<td>Preposition/subordinating conjunction</td>
</tr>
<tr>
<td>JJ</td>
<td>Adjective</td>
</tr>
<tr>
<td>JJR</td>
<td>Adjective, comparative</td>
</tr>
<tr>
<td>JJS</td>
<td>Adjective, superlative</td>
</tr>
<tr>
<td>LS</td>
<td>List item marker</td>
</tr>
<tr>
<td>MD</td>
<td>Modal</td>
</tr>
<tr>
<td>NN</td>
<td>Noun, singular or mass</td>
</tr>
<tr>
<td>NNS</td>
<td>Noun, plural</td>
</tr>
<tr>
<td>NNP</td>
<td>Proper noun, singular</td>
</tr>
<tr>
<td>NNPS</td>
<td>Proper noun, plural</td>
</tr>
<tr>
<td>PDT</td>
<td>Predeterminer</td>
</tr>
<tr>
<td>POS</td>
<td>Possessive ending</td>
</tr>
<tr>
<td>PRP</td>
<td>Personal pronoun</td>
</tr>
<tr>
<td>PP$</td>
<td>Possessive pronoun</td>
</tr>
<tr>
<td>RB</td>
<td>Adverb</td>
</tr>
<tr>
<td>RBR</td>
<td>Adverb, comparative</td>
</tr>
<tr>
<td>RBS</td>
<td>Adverb, superlative</td>
</tr>
<tr>
<td>RP</td>
<td>Particle</td>
</tr>
<tr>
<td>SYM</td>
<td>Symbol (mathematical or scientific)</td>
</tr>
<tr>
<td>TO</td>
<td>to</td>
</tr>
<tr>
<td>UH</td>
<td>Interjection</td>
</tr>
<tr>
<td>VB</td>
<td>Verb, base form</td>
</tr>
<tr>
<td>VBD</td>
<td>Verb, past tense</td>
</tr>
<tr>
<td>VBG</td>
<td>Verb, gerund/present participle</td>
</tr>
<tr>
<td>VBN</td>
<td>Verb, past participle</td>
</tr>
<tr>
<td>VBP</td>
<td>Verb, non-3rd ps. sing. present</td>
</tr>
<tr>
<td>VBZ</td>
<td>Verb, 3rd ps. sing. present</td>
</tr>
<tr>
<td>WDT</td>
<td>WH-determiner</td>
</tr>
<tr>
<td>WP</td>
<td>WH-pronom</td>
</tr>
<tr>
<td>WP$</td>
<td>Possessive WH-pronom</td>
</tr>
<tr>
<td>WRB</td>
<td>WH-adverb</td>
</tr>
<tr>
<td>#</td>
<td>Pound sign</td>
</tr>
<tr>
<td>$</td>
<td>Dollar sign</td>
</tr>
<tr>
<td>.</td>
<td>Sentence-final punctuation</td>
</tr>
<tr>
<td>,</td>
<td>Comma</td>
</tr>
<tr>
<td>:</td>
<td>Colon, semi-colon</td>
</tr>
<tr>
<td>(</td>
<td>Left bracket character</td>
</tr>
<tr>
<td>)</td>
<td>Right bracket character</td>
</tr>
<tr>
<td>&quot;</td>
<td>Straight double quote</td>
</tr>
<tr>
<td>’</td>
<td>Left open single quote</td>
</tr>
<tr>
<td>‘</td>
<td>Left open double quote</td>
</tr>
<tr>
<td>”</td>
<td>Right open double quote</td>
</tr>
<tr>
<td>”</td>
<td>Right close single quote</td>
</tr>
<tr>
<td>”</td>
<td>Right close double quote</td>
</tr>
</tbody>
</table>
### Review the use of Regular Expressions

<table>
<thead>
<tr>
<th>Operator</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Wildcard, matches any character</td>
</tr>
<tr>
<td>^abc</td>
<td>Matches some pattern <code>abc</code> at the start of a string</td>
</tr>
<tr>
<td>abc$</td>
<td>Matches some pattern <code>abc</code> at the end of a string</td>
</tr>
<tr>
<td>[abc]</td>
<td>Matches one of a set of characters</td>
</tr>
<tr>
<td>[A-Z0-9]</td>
<td>Matches one of a range of characters</td>
</tr>
<tr>
<td>ed</td>
<td>ing</td>
</tr>
<tr>
<td>*</td>
<td>Zero or more of previous item, e.g. <code>a*</code>, <code>[a-z]*</code> (also known as Kleene Closure)</td>
</tr>
<tr>
<td>+</td>
<td>One or more of previous item, e.g. <code>a+</code>, <code>[a-z]</code>+</td>
</tr>
<tr>
<td>?</td>
<td>Zero or one of the previous item (i.e. optional), e.g. <code>a?</code>, <code>[a-z]</code>?</td>
</tr>
<tr>
<td>{n}</td>
<td>Exactly <code>n</code> repeats where <code>n</code> is a non-negative integer</td>
</tr>
<tr>
<td>{n,}</td>
<td>At least <code>n</code> repeats</td>
</tr>
<tr>
<td>{,n}</td>
<td>No more than <code>n</code> repeats</td>
</tr>
<tr>
<td>{m,n}</td>
<td>At least <code>m</code> and no more than <code>n</code> repeats</td>
</tr>
<tr>
<td>a(b</td>
<td>c)+</td>
</tr>
</tbody>
</table>
Chunking = Shallow Parsing
Chunking & Parsing

- Chunking is shallow, non-recursive parsing
- Uses Regex grammars to build up trees

- Parsing builds deeper structures
  - But you may not need them for many applications
  - Parsing more prone to errors
  - May be difficult to get a ‘cover’ for the complete sentence
  - There are many many flavors of parsing
import re, sys, nltk
from nltk.stem.wordnet import WordNetLemmatizer

# Our simple grammar from class (and the book)
GRAMMAR = ""
    N: {<PRP>|<NN.*>}
    V: {<V.*>}
    ADJ: {<JJ.*>}
    NP: {<DT>? <ADJ>* <N>+}
    PP: {<IN> <NP>}
    VP: {<TO>? <V> (<NP>|<PP>)+}
"""

LOC_PP = set(["in", "on", "at"])
Chunking Nouns Using Regexes in NLTK

- \texttt{nltk.RegexpParser}
  - Grammar format is similar to standard regexes

- Simple Example
  - >>>grammar = "NP: \{<DT>? <JJ>* <NN>\}"
  - A Noun Phrase is an optional determiner followed by any number of adjectives and then a (singular) noun.

- Regex meta-chars can be used within tags or to the tags
  - >>>grammar = "NP: \{<DT>? <JJ>* <NN.*>+\}"
  - Matches one or more singular, plural and proper nouns
Cascaded Chunker Grammars

- RegexpParser chunker begins with a flat structure in which no tokens are chunked.

- The chunking rules are applied in turn, successively updating the chunk structure.

- Once all of the rules have been invoked, the resulting chunk structure is returned.
Cascading Chunker Examples

- Multiple categories in one grammar (cascading)

    We can define categories, which can be used later

    Not just for grammatical categories

- Can chunk any category you're interested in

```
N: {<PRP>|<NN.*>}
V: {<V.*>}
ADJ: {<JJ.*>}
NP: {<DT>? <ADJ>* <N>+}
PP: {<IN> <NP>}
VP: {<TO>? <V> (<NP>|<PP>)*}
```
The Fox & The Crow

A retelling from Scheherazade

There once was a crow. The crow was sitting on a branch of a tree. Some cheese was in the beak of the crow. A fox observed the crow and tried to discover how to get the cheese. The fox came and stood under the tree. The fox looked toward the crow and said that he saw a noble bird who was above him. The fox said that the beauty of the bird was incomparable. The fox said that the hue of the plumage of the bird was exquisite. The fox said that -- if the sweetness of the voice of the bird is equal to the fairness of the appearance of the bird -- the bird would be undoubtedly the queen of every bird. The crow felt that the fox had flattered her and cawed loudly in order for she to show him that she was able to sing. The cheese fell. The fox snatched the cheese, said that the crow was able to sing and the fox said that the crow needed wits.
The Fox & The Crow

Where was the crow sitting?

There once was a crow. The crow was sitting on a branch of a tree. Some cheese was in the beak of the crow. A fox observed the crow and tried to discover how to get the cheese. The fox came and stood under the tree. The fox looked toward the crow and said that he saw a noble bird who was above him. The fox said that the beauty of the bird was incomparable. The fox said that the hue of the plumage of the bird was exquisite. The fox said that -- if the sweetness of the voice of the bird is equal to the fairness of the appearance of the bird -- the bird would be undoubtedly the queen of every bird. The crow felt that the fox had flattered her and cawed loudly in order for she to show him that she was able to sing. The cheese fell. The fox snatched the cheese, said that the crow was able to sing and the fox said that the crow needed wits.
Simple Solution

- Find all the sentences that mention the crow
- Chunk each sentence
- Find all the PP phrases
- Identify PP phrases indicating a location
- Return the NP part of the PP
Simple Solution

- Chunk the PPs & look for locative preps

```python
# Our simple grammar from last time
GRAMMAR = ""
N: {<PRP>|<NN.*>}
V: {<V.*>}
ADJ: {<JJ.*>}
NP: {<DT>? <ADJ>* <N>+}
PP: {<IN> <NP>}
VP: {<TO>? <V> (<NP>|<PP>)*}
"

# Create a parser from the grammar
chunker = nltk.RegexpParser(GRAMMAR)

# Read the story from a file
text = read_file(filename)

# Apply the standard pipeline to get the sentences
sentences = get_sentences(text)

# Find all the sentences that match our target
keywords = ['"crow"', '"sitting"']
crow_sentences = find_sentences(keywords, sentences)

# Find locations
locations = find_candidates(crow_sentences, chunker)
```
Syntactic Representations

Constituency Parses and Dependency Parses
Simple Solution

- Chunk the PPs & look for locative preps
- Or start using the parse trees

(S
  (NP The/DT (N crow/NN))
  (VP (V was/VBD))
  (VP
    (V sitting/VBG)
    (PP on/IN
      (NP a/DT (N branch/NN)))
    (PP of/IN
      (NP a/DT (N tree/NN))))
./.)
The Fox & The Crow

Where was the crow sitting?

There once was a crow. The crow was sitting on a branch of a tree. Some cheese was in the beak of the crow.

A fox observed the crow and tried to discover how to get the cheese.

The fox said that he saw a noble bird who was above him.

The fox said that the beauty of the bird was incomparable.

The fox said that the hue of the plumage of the bird was exquisite.

The fox said that -- if the sweetness of the voice of the bird is equal to the fairness of the appearance of the bird -- the bird would be undoubtedly the queen of every bird.

The crow felt that the fox had flattered her and cawed loudly in order for she to show him that she was able to sing.

The cheese fell.

The fox snatched the cheese, said that the crow was able to sing and the fox said that the crow needed wits.
Stanford Dependencies, Constituents

- LEARN TO USE THE DEMO PAGE. VERY USEFUL.
• http://nlp.stanford.edu:8080/parser/
Stanford Demo: parser output


---

Stanford Parser

Please enter a sentence to be parsed:

A fox observed the crow and tried to discover how to get the cheese.

Language: English  
Sample Sentence

Your query

A fox observed the crow and tried to discover how to get the cheese.
What is syntax good for?

• Increasing Precision

• Some kinds of questions hard to answer from the string because of long distance syntactic dependencies.

“The people rebelled and created a riot”

• Who created a riot?

• What did the people create?
Two Types of Parse Trees

- **Constituency Parses**
  - Use `nltk.Tree`
  - This is a basic tree like in 12b, etc
    - N-ary instead of binary

- **Dependency Parses**
  - Use `nltk.DependencyGraph`

- Both have methods for parsing string input
15. Can I obtain multiple parse trees for a single input sentence?

Yes, for the PCFG parser (only). With a PCFG parser, you can give the option `-printPCFGkBest n` and it will print the $n$ highest-scoring parses for a sentence. They can be printed either as phrase structure trees or as typed dependencies in the usual way via the `-outputFormat` option, and each receives a score (log probability). The $k$ best parses are extracted efficiently using the algorithm of Huang and Chiang (2005).

16. I don't [understand/like/agree with] the parse tree that is assigned to my sentence. Can you [explain/fix] it?

This may be because the parser chose an incorrect structure for your sentence, or because the phrase structure annotation conventions used for training the parser don't match your expectations. To make sure you understand the annotation conventions, please read the bracketing guidelines for the parser model that you're using, which are referenced above. Or it may be because the parser made a mistake. While our goal is to improve the parser when we can, we can't fix individual examples. The parser is just choosing the highest probability analysis according to its grammar.

17. Why does the parser accept incorrect/ungrammatical sentences?

This parser is in the space of modern statistical parsers whose goal is to give the most likely sentence analysis to a list of words. It does not attempt to determine grammaticality, though it will normally prefer a "grammatical" parse for a sentence if one exists. This is appropriate in many circumstances, such as when wanting to interpret user input, or dealing with conversational speech, web pages, non-native speakers, etc.

For other applications, such as grammar checking, this is less appropriate. One could attempt to assess grammaticality by looking at the probabilities that the parser returns for sentences, but it is difficult to normalize this number to give a useful "grammaticality" score, since the probability strongly depends on other factors like the length of the sentence, the rarity of the words in the sentence, and whether word dependencies in the sentence being tested were seen in the training data or not.
Looking at Stanford’s outputs

Your query

A fox observed the crow and tried to discover how to get the cheese.

Tagging

A/DT fox/NN observed/VBD the/DT crow/NN and/CC tried/VB

Parse

(ROOT
  (S
   (NP (DT A) (NN fox))
   (VP
    (VP (VBD observed)
     (NP (DT the) (NN crow)))
    (CC and)
    (VP (VBD tried)
     (S
      (VP (TO to)
       (VP (VB discover)
        (SBAR
         (WHADVP (WRB how))
         (S
          (VP (TO to)
           (VP (VB get)
            (NP (DT the) (NN cheese))))))))))))
   (...)))

Typed dependencies

det(fox-2, A-1)
nsubj(observed-3, fox-2)
root(ROOT-0, observed-3)
det(crow-5, the-4)
dobj(observed-3, crow-5)
cc(observed-3, and-6)
conj(observed-3, tried-7)
advmod(get-12, how-10)
advmod(get-12, to-11)
ccomp(discover-9, get-12)
det(cheese-14, the-13)
dobj(get-12, cheese-14)
Sample Constituency Tree

```
ROOT
  S
    NP
      DT The
      NN crow
      VBD was
    VP
      VBG sitting
      IN on
      NP
        DT a
        NN branch
    PP
      IN of
        NP
          DT a
          NN tree
```
Stanford Dependencies

- Lots of dependency relations
- Full list and description available at:
- Several different types
  - Basic
    - Form a tree structure
    - Each word is the dependent of another
  - Collapsed
    - More direct relationships between content words
    - Might form a cycle
Stanford Dependencies (Subj, Obj)

**root:** root
The root grammatical relation points to the root of the sentence. A fake node “ROOT” is used as the governor. The ROOT node is indexed with “0”, since the indexation of real words in the sentence starts at 1.

“I love French fries.” root(ROOT, love)
“Bill is an honest man” root(ROOT, man)

**nsubj:** nominal subject
A nominal subject is a noun phrase which is the syntactic subject of a clause. The governor of this relation might not always be a verb: when the verb is a copular verb, the root of the clause is the complement of the copular verb, which can be an adjective or noun.

“Clinton defeated Dole” nsubj(defeated, Clinton)
“The baby is cute” nsubj(cute, baby)

**dobj:** direct object
The direct object of a VP is the noun phrase which is the (accusative) object of the verb.

“She gave me a raise” dobj(gave, raise)
“They win the lottery” dobj(win, lottery)

*This is categorized differently in the Stanford Dependency Manual*
Arguments

arg - argument
  agent - agent
  comp - complement
    acomp - adjectival complement
    ccomp - clausal complement with internal subject
    xcomp - clausal complement with external subject
  obj - object
    dobj - direct object
    iobj - indirect object
    pobj - object of preposition
  subj - subject
    nsubj - nominal subject
      nsubjpass - passive nominal subject
    csubj - clausal subject
      csubjpass - passive clausal subject
**acomp: adjectival complement**

An adjectival complement of a verb is an adjectival phrase which functions as the complement (like an object of the verb).

She looks very beautiful

**amod: adjectival modifier**

An adjectival modifier of an NP is any adjectival phrase that serves to modify the meaning of the NP.

- “Sam eats red meat”
- “Sam took out a 3 million dollar loan”
- “Sam took out a $3 million loan”

**advmod: adverb modifier**

An adverb modifier of a word is a (non-clausal) adverb or adverb-headed phrase that serves to modify the meaning of the word.

- “Genetically modified food”
- “less often”
Stanford Dependencies (Modifiers)

She looks very beautiful

**advcl**: adverbial clause modifier

An adverbial clause modifier of a VP or S is a clause modifying the verb (temporal clause, consequence, conditional clause, purpose clause, etc.).

“The accident happened as the night was falling”  \(\text{advcl}(\text{happened}, \text{falling})\)

“If you know who did it, you should tell the teacher”  \(\text{advcl}(\text{tell}, \text{know})\)

“He talked to him in order to secure the account”  \(\text{advcl}(\text{talked}, \text{secure})\)
**aux:** auxiliary

An auxiliary of a clause is a non-main verb of the clause, e.g., a modal auxiliary, or a form of "be", "do" or "have" in a periphrastic tense.

Reagan has died

He should leave
**iobj: indirect object**
The indirect object of a VP is the noun phrase which is the (dative) object of the verb.

“She gave me a raise”

\[ iobj(\text{gave, me}) \]

**tmod: temporal modifier**
A temporal modifier (of a VP, NP, or an ADJP) is a bare noun phrase constituent that serves to modify the meaning of the constituent by specifying a time. (Other temporal modifiers are prepositional phrases and are introduced as prep.)

“Last night, I swam in the pool”

\[ tmod(\text{swam, night}) \]
Get the Manual and Use it

Looking at Stanford’s outputs

Your query

A fox observed the crow and tried to discover how to get the cheese.

Tagging

A/DT fox/NN observed/VBD the/DT crow/NN and/CC tried/VB

Parse

(ROOT
  (S
    (NP (DT A) (NN fox))
    (VP
      (VP (VBD observed)
        (NP (DT the) (NN crow)))
      (CC and)
      (VP (VBD tried)
        (S
          (VP (TO to)
            (VP (VB discover)
              (SBAR
                (WHADVP (WRB how))
                (S
                  (VP (TO to)
                    (VP (VB get)
                      (NP (DT the) (NN cheese))))))))))
  (. . .))

Typed dependencies

det(fox-2, A-1)
nsubj(observed-3, fox-2)
root(ROOT-0, observed-3)
det(crow-5, the-4)
dobj(observed-3, crow-5)
cc(observed-3, and-6)
conj(observed-3, tried-7)
aux(discover-9, to-8)
xcomp(tried-7, discover-9)
advmod(get-12, how-10)
aux(get-12, to-11)
ccomp(discover-9, get-12)
det(cheese-14, the-13)
dobj(get-12, cheese-14)
Using Demo to examine parser output


**Who tried to discover how to get the cheese?**

**How do we figure out the subject of “tried to discover”**

- S = NP VP
- VP = VP CC VP
Fox observed and tried to discover

Typed dependencies

- det(fox-2, A-1)
- nsubj( observed-3, fox-2)
- root(ROOT-0, observed-3)
- det(crow-5, the-4)
- dobj( observed-3, crow-5)
- cc( observed-3, and-6)
- conj( observed-3, tried-7)
- aux(discover-9, to-8)
- xcomp( tried-7, discover-9)
- advmod( get-12, how-10)
- aux( get-12, to-11)
- ccomp( discover-9, get-12)
- det(cheese-14, the-13)
- dobj( get-12, cheese-14)

- Who tried to discover how to get the cheese?
- How do we figure out the subject of “tried to discover”
- NSUBJ (observed, fox)
- Where is the NSUBJ for “tried”
- CONJ (observed, tried)
And now for something completely different. NOT.
Story: Fox observed and set his wits

http://nlp.stanford.edu:8080/parser/index.jsp
Story: Fox observed and set his wits

(NP (DT a) (NN -tree))

(PP (IN -with))

(NP

  (NP (DT a) (NN piece))

  (PP (IN of))

  (NP

    (NP (NN cheese))

    (PP (IN in)

      (NP (PRP$ her) (NN beak)))))))))))))))))))

(SBAR

  (WHADVP (WRB when)))

(S

  (NP (DT a) (NNP Fox))

  (VP

    (VP (VBD observed)

      (NP (PRP her)))

    (CC and)

    (VP (VBD set)

      (NP (PRP$ his) (NNS wits)))

  (S

    (VP (TO to)

      (VP (VB work))
Fox observed and set his wits

Typed dependencies

det(Crow-2, A-1)
nsubj(sitting-4, Crow-2)
aux(sitting-4, was-3)
root(ROOT-0, sitting-4)
prep(sitting-4, on-5)
det(branch-7, a-6)
pobj(on-5, branch-7)
prep(branch-7, of-8)
det(tree-10, a-9)
pobj(of-8, tree-10)
prep(tree-10, with-11)
det(piece-13, a-12)
pobj(with-11, piece-13)
prep(piece-13, of-14)
pobj(of-14, cheese-15)
prep(cheese-15, in-16)
poss(beak-18, her-17)
pobj(in-16, beak-18)
advmod(observed-22, when-19)
det(Fox-21, a-20)
nsubj(observed-22, Fox-21)
advcl(sitting-4, observed-22)
dobj(observed-22, her-23)
cc(observed-22, and-24)
conj(observed-22, set-25)
poss(wits-27, his-26)
dobj(set-25, wits-27)
aux(work-29, to-28)
vmod(set-25, work-29)
aux(discover-31, to-30)
xcomp(work-29, discover-31)
det(way-33, some-32)
dobj(discover-31, way-33)
prep(way-33, of-34)
pcomp(of-34, getting-35)
dobj(getting-35, cheese-37)
det(cheese-37, the-36)
Eagle Knocked and Spilled.

Stanford Parser

Please enter a sentence to be parsed:

Heated with his exertions, the man was about to slake his thirst with a draught from the horn, when the Eagle knocked it out of his hand, and spilled its contents upon the ground.

Language:  English  Sample Sentence

Your query

Heated with his exertions, the man was about to slake his thirst with a draught.
and spilled

(SBAR
  (WHADVP (WRB when))
(S
  (NP (DT the) (NNP Eagle))
  (VP
    (VP (VBD knocked)
      (NP (PRP it))
      (PRT (RP out))
      (PP (IN of)
        (NP (PRP$ his) (NN hand))))
    (, ,)
  (CC and)
  (VP (VBD spilled)
    (NP (PRP$ its) (NNS contents))
    (PP (IN upon)
      (NP (DT the) (NN ground)))))
))
)
The Dependency Tree

det(Eagle-23, the-22)
nsubj(knocked-24, Eagle-23)
rcmod(horn-19, knocked-24)
dobj(knocked-24, it-25)
prt(knocked-24, out-26)
prep(knocked-24, of-27)
poss(hand-29, his-28)
pobj(of-27, hand-29)
cc(knocked-24, and-31)
conj(knocked-24, spilled-32)
poss(contents-34, its-33)
dobj(spilled-32, contents-34)
prep(spilled-32, upon-35)
det(ground-37, the-36)
SCH. A young man long ago crashed the motorbike of the young man on the front yard of a narrator and broke the neck of the young man. The narrator stayed with the young man and didn't aid him because the young man had broken the neck of the young man. The young man died on the spot of the yard of the narrator. The narrator later went back to the spot of the yard of the narrator and decided to talk to the young man because it wanted the young man to know the narrator regretting that it had not aided him. The narrator saw some bright flash in a group of trees that was above the narrator. The narrator thought for the brother of the narrator to use the flashlight of the brother of the narrator. The narrator entered the house of the narrator and heard that the asleep family of the narrator was asleep. The narrator began to wonder that the flash was an orb.
More coordination

- Who broke the neck of the young man?
Who broke the neck of the young man?
SCH. A summit meeting named G20 summit started on eventful today. G20 summit happened annually. A world and many leader came and talked about it running a government. A people protested because it disagreed about a view. The people protested peacefully on a street. The people rebelled and created riot. The people burned a police car and threw a thing at a police. The police alleviated the people of riot. The police fired a tear gas at the people and fired a bullet at the people, and the people smashed a window.

- Who created a riot?
- Who fired a bullet at the people?
And now for something completely different. Kind of.
SCH. A summit meeting named G20 summit started on eventful today. G20 summit happened annually. A world and many leader came and talked about it running a government. A people protested because it disagreed about a view. The people protested peacefully on a street.

The people rebelled and created riot. The people burned a police car and threw a thing at a police. The police alleviated the people of riot. The police fired a tear gas at the people and fired a bullet at the people, and the people smashed a window.

- What did the people create?
What did the people create? OBJ | VP NP

**Your query**

The people rebelled and created riot.

**Tagging**

The/DT people/NNS rebelled/VBD and/CC

**Parse**

(Root

(S

(NP (DT The) (NNS people))

(VP (VBD rebelled))

(CC and)

(VP (VBD created)

(NP (NN riot))))

( . . . ))

**Typed dependencies**

<table>
<thead>
<tr>
<th>Dependency Type</th>
<th>Argument 1</th>
<th>Argument 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>det</td>
<td>people-2, The-1</td>
<td></td>
</tr>
<tr>
<td>nsubj</td>
<td>rebelled-3, people-2</td>
<td></td>
</tr>
<tr>
<td>root</td>
<td>ROOT-0, rebelled-3</td>
<td></td>
</tr>
<tr>
<td>cc</td>
<td>rebelled-3, and-4</td>
<td></td>
</tr>
<tr>
<td>conj</td>
<td>rebelled-3, created-5</td>
<td></td>
</tr>
<tr>
<td>dobj</td>
<td>created-5, riot-6</td>
<td></td>
</tr>
</tbody>
</table>
Adverbials: When did the young man die?

Your query
A few years ago, a young man died out on my front lawn when he crashed his motorbike.

Tagging
A/DT few/JJ years/NNS ago/RB ,/, a/DT young/JJ man/NN died/VBD out/F

Parse
(ROOT
(S
(ADVP
   (NP (DT A) (JJ few) (NNS years))
   (RB ago))
   (...)
(NP (DT a) (JJ young) (NN man))
(VP (VBD died)
   (PRT (RP out))
   (PP (IN on)
      (NP (PRP$ my) (JJ front) (NN lawn)))
(SBAR
   (WHADVP (WRB when))
   (S
      (NP (PRP he))
      (VP (VBD crashed)
         (NP (PRP$ his) (NN motorbike))))
(....)))

- "years" can answer a when question
- When adverbials can answer a when question
Data Structures for Parse Trees
Two Types of Parse Trees

- Constituency Parsers
  - Use `nltk.Tree`
  - This is a basic tree like in 12b, etc
    - N-ary instead of binary

- Dependency Parsers
  - Use `nltk.DependencyGraph`

- Both have methods for parsing string input
Additional resources for assignment

- assignment6-stub.py (3 KB; May 9, 2015 11:51 am)
- baseline-stub.py (2 KB; May 9, 2015 11:51 am)
- chunk-demo.py (4 KB; May 9, 2015 11:51 am)
- parse-demo.py (3 KB; May 9, 2015 11:51 am)
- constituency-demo-stub.py (3 KB; May 9, 2015 11:51 am)
- dependency-demo-stub.py (3 KB; May 9, 2015 11:51 am)
- score-answers.pl (13 KB; May 9, 2015 11:52 am)
- fables-01_my_answers.txt (1 KB; May 9, 2015 11:52 am)
- hw6_dataset.zip (20 KB; May 9, 2015 11:53 am)
- hw6-v3.pdf (125 KB; May 9, 2015 11:53 am)
Dep Pareses for the questions and stories

- USE the TREE READER to convert

```
QuestionId: blogs-01-10
What WP 2 nsubj
happened VBD 0 root
the DT 6 det
police NN 6 nn
cars NNS 2 prep

QuestionId: blogs-01-11
What WP 3 nsubjpass
was VBD 3 auxpass
fired VBN 0 root
```
Constituent Parses .par files

- USE the TREE READER to convert

```
QuestionId: blogs-01-1
(ROOT (SBARQ (WHADVP (WRB When))) (SQ (VBD did) (NP (DT the) (NN G20) (NN summit))) (VP (VB start))) (. ?))

QuestionId: blogs-01-10
(ROOT (SBARQ (WHNP (WP What))) (SQ (VP (VBD happened) (PP (TO to) (NP (DT the) (NN police) (NNS cars))))) (. ?))

QuestionId: blogs-01-11
(ROOT (SBARQ (WHNP (WP What))) (SQ (VBD was) (VP (VBN fired))) (. ?))
```
Constituency Tree

- On disk it looks like this:

  (ROOT (S (NP (DT The) (NN crow)) (VP (VBD was) (VP (VBG sitting) (PP (IN on) (NP (NP (DT a) (NN branch)) (PP (IN of) (NP (DT a) (NN tree)))))))) ( . . )))

- In our dataset it's all on one line

- It doesn't have to be but it makes reading it in easier
Reading in Constituency Trees

- Easy to read
- Each parse is on a single line
Reading in Constituency Trees

```python
# Read the constituency parse from the line and construct the Tree
def read_con_parsers(parfile):
    fh = open(parfile, 'r')
    lines = fh.readlines()
    fh.close()
    return [Tree.parse(line) for line in lines]
```

- Easy to read
- Each parse is on a single line
- Voila!
# Read the constituency parse from the line and construct the Tree

def read_con_parses(parfile):
    fh = open(parfile, 'r')
    lines = fh.readlines()
    fh.close()
    return [Tree.parse(line) for line in lines]

- Easy to read
- Each parse is on a single line
- Voila!
Dependency Graph

- On disk it looks like this:

<table>
<thead>
<tr>
<th>Word</th>
<th>POS</th>
<th>Index of Parent</th>
<th>Dependency Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>The</td>
<td>DT</td>
<td>2</td>
<td>det</td>
</tr>
<tr>
<td>crow</td>
<td>NN</td>
<td>4</td>
<td>nsubj</td>
</tr>
<tr>
<td>was</td>
<td>VBD</td>
<td>4</td>
<td>aux</td>
</tr>
<tr>
<td>sitting</td>
<td>VBG</td>
<td>0</td>
<td>root</td>
</tr>
<tr>
<td>a</td>
<td>DT</td>
<td>7</td>
<td>det</td>
</tr>
<tr>
<td>branch</td>
<td>NN</td>
<td>4</td>
<td>prep_on</td>
</tr>
<tr>
<td>a</td>
<td>DT</td>
<td>10</td>
<td>det</td>
</tr>
<tr>
<td>tree</td>
<td>NN</td>
<td>7</td>
<td>prep_of</td>
</tr>
</tbody>
</table>

- Tab separated
  - Word, POS, index of parent, dependency relation
- First word is index 1
- "Dummy" root element is index 0
Each tree spans multiple rows

A blank line separates parses

Slightly more involved
- Read in the string for one tree (i.e., up to a blank line)
- Create the DependencyGraph from that string
Each DependencyGraph consists of a list of nodes:
- nodelist

Each node is a dict with the following keys:
- **head**: index of the parent (the root doesn't have a head)
- **word**: the lexical item
- **rel**: the grammatical relation between the item and the head
- **tag**: the part of speech tag of the node
- **deps**: the list of dependent nodes
- **address**: index of the item in the sentence (starting from 1)
# Read the lines of an individual dependency parse

def read_dep(fh):
    dep_lines = []
    for line in fh:
        line = line.strip()
        if len(line) == 0:
            return nltk.DependencyGraph("\n".join(dep_lines))
        dep_lines.append(line)

    return DependencyGraph("\n".join(dep_lines))
    if len(dep_lines) > 0 else None
Read a Dependency Parse

```json
[{
    'address': 0,
    'deps': [4],
    'rel': 'TOP',
    'tag': 'TOP',
    'word': None
},
{
    'address': 1,
    'deps': [],
    'head': 2,
    'rel': 'det',
    'tag': 'DT',
    'word': 'The'
},
{
    'address': 2,
    'deps': [1],
    'head': 4,
    'rel': 'nsubj',
    'tag': 'NN',
    'word': 'crow'
},
{
    'address': 3,
    'deps': [],
    'head': 4,
    'rel': 'aux',
    'tag': 'VBD',
    'word': 'was'
}, ...
]```
Manipulating a Constituency Tree

- **Basic Operations**
  - `[]` – access the children
  - `subtrees(filter)` – get all subtrees optionally filtering only ones that meet a criteria

- **Parented Tree Operations**
  - `parent()`,
  - `parent_index()`,
  - `left_sibling()`,
  - `right_sibling()`,
  - `root()`,
  - `treeposition()`
Manipulating a Constituency Tree

The diagram illustrates a constituency tree for the sentence: "The crow was sitting on a branch of a tree."
Manipulating a Constituency Tree
Manipulating a Constituency Tree

The diagram shows a constituency tree with the following structure:

- Root: S
  - NP: DT NN VBD
    - DT: The
    - NN: crow
    - VBD: was
  - VP: VBG PP
    - VBG: sitting
    - IN: on
    - NP: DT NN NP PP
      - DT: a
      - NN: branch
      - IN: of
      - NP: DT NN
        - DT: a
        - NN: tree
Manipulating a Constituency Tree

```
ROOT
  \--- S
      \--- NP
           \--- DT
              \--- The
           \--- NN
              \--- crow
           \--- VBD
              \--- was
      \--- VP
           \--- VBG
              sitting
           \--- IN
              \--- on
      \--- NP
           \--- NP
                \--- DT
                   \--- a
                \--- NN
                   \--- branch
           \--- IN
              \--- of
      \--- NP
           \--- PP
                \--- NP
                     \--- DT
                        \--- a
                     \--- NN
                        \--- tree
```

left_sibling

None
Manipulating a Constituency Tree

The root of the tree is indicated by a circled node labeled "VBG". The tree structure shows the constituents of the sentence "The crow was sitting on a branch of a tree."