Introduction to Natural Language Processing
We do this by QA
Assignment 6. Get started on question answering. Due Next Wed at 11:55 PM Teams of 2, 3 Stay after class if you want to find team members
The next three homework assignments will be to design and build a question answering system. We will use stories from Aesop’s Fables and the Blogs Corpus. For each homework you will receive new datasets with more difficult stories and questions.

We created questions and an answer key for each story. Each question has a unique QuestionID which is the StoryID followed by a dash and question number. For example, “blogs-01-1” means that this is question #1 pertaining to story “blogs-01”. In some cases, the answer key allows for several acceptable answers (e.g., “Toronto, Ontario” vs. “Toronto”), paraphrases (e.g., “Human Immunodeficiency Virus” vs. “HIV”), varying amounts of information (e.g., “he died” vs. “he died in his sleep of natural causes”), or occasionally different interpretations of the question (e.g., “Where did the boys learn how to survive a storm?” “camping tips from a friend” vs. “their backyard”). When more than one answer is acceptable, the acceptable answers are separated by a vertical bar (|). Below is a sample answer key:

```
QuestionID: blogs-01-2
Question: What is the summit meeting named?
Answer: G20 summit
Difficulty: Easy
Type: Sch
```
How is it going to be set up?

- blogs-01.answers
- blogs-01.questions
- blogs-01.questions.dep
- blogs-01.questions.par
- blogs-01.sch
- blogs-01.sch.dep
- blogs-01.sch.par
• When did the G20 summit start? (sch)
• What is the summit meeting named? (sch) pos np
• Where did the protest happen? (sch | story) loc NE regex
• What did the people burn? (sch)
• Who rebelled? (sch)
• Who created a riot? (sch)
• What happened to a police car? (sch)
• Who burned a police car? (sch)
• What was burned? (sch | story)
• What happened to the police cars? (story)
• What was fired? (story)
• What was fired at the rioters? (story)
• What was smashed? (story)
QuestionID: blogs-01-1
Question: When did the G20 summit start?
Difficulty: Easy
Type: Sch

Answer: on eventful today | eventful today
Difficulty: Easy
Type: Sch

QuestionID: blogs-01-2
Question: What is the summit meeting named?
Difficulty: Easy
Type: Sch

Answer: G20 summit
Difficulty: Easy
Type: Sch

QuestionID: blogs-01-3
Question: Where did the protest happen?
Difficulty: Easy
Type: Story | Sch

Answer: on a street | along the street where I work | right in front of my store
Difficulty: Easy
Type: Story | Sch

QuestionID: blogs-01-4
Question: What did the people burn?
Difficulty: Easy
Type: Sch

Answer: a police car | police car
Difficulty: Easy
Type: Sch

QuestionID: blogs-01-6
Question: Who created a riot?
Difficulty: Easy

Answer: Who rebelled?
Today was a very eventful work day. Today was the start of the G20 summit. It happens every year and it is where 20 of the leaders of the world come together to talk about how to run their governments effectively and what not. Since there are so many leaders coming together their are going to be a lot of people who have different views on how to run the government they follow so they protest. There was a protest that happened along the street where I work and at first it looked peaceful until a bunch of people started rebelling and creating a riot. Police cars were burned and things were thrown at cops. Police were in full riot gear to alleviate the violence. As things got worse tear gas and bean bag bullets were fired at the rioters while they smash windows of stores. And this all happened right in front of my store which was kind of scary but it was kind of interesting since I've never seen a riot before.
When did the G20 summit start? (sch)
What is the summit meeting named? (sch) pos np
Where did the protest happen? (sch | story) loc NE regex
on a street | along the street where I work | right in front of my store
What did the people burn? (sch)
Who rebelled? (sch)
Who created a riot? (sch)
What happened to a police car? (sch)
Who burned a police car? (sch)
What was burned? (sch | story)
What happened to the police cars? (story)
What was fired? (story)
What was fired at the rioters? (story)
What was smashed? (story)

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.
Sch realization resolves pronouns, simplifies sentences, but may change words.
HW 6

- TWO Fables for training with questions and answers
- ONE BLOG for training with questions and answers
- We test your program on a DEVELOPMENT SET, expanded set of questions of new type
- HW 7: GIVE you the DEVELOPMENT set we used to test
  - GIVE you a NEW training set with different additional kinds of questions
  - TEST AGAIN (FRIDAY May 29th due date 11:55 PM)
- HW8: Final TEST, due MONDAY June 8th
- Class presentations June 10th during the finals slot, PPTX OR PDF DUE 4PM THAT DAY.
Performance Evaluation

2 main requirements:

• **Evaluation metric**: mathematically defines how to measure the system’s performance against human-annotated gold standard

• **Scoring program**: implements the metric and provides performance measures
  — For each question and over all questions
  — Can be easily run at any time to continually test your system performance and see what any changes have improved or worsened
  — Regression testing: Sets of questions whose performance should be maintained
Evaluation Metrics

- **Precision** = correct answers/answers produced (what proportion of the answers produced are accurate?)

- **Recall** = correct answers/total possible correct answers (what proportion of all the correct answers did the system find?)

- Trade-off between Precision and Recall
- **F1** (balanced) Measure = $2PR / 2(R + P)$
Precision and Recall

- **Precision**: fraction of answers that are correct =
  - \( P(\text{correct} | \text{in the answer}) \)

- **Recall**: fraction of answers that are returned vs. all answers in the text =
  - \( P(\text{in the answer} | \text{correct}) \)

<table>
<thead>
<tr>
<th></th>
<th>correct</th>
<th>Not correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>In the answer</td>
<td>( tp )</td>
<td>( fp )</td>
</tr>
<tr>
<td>Not in the answer</td>
<td>( fn )</td>
<td>( tn )</td>
</tr>
</tbody>
</table>

Precision \( P = \frac{tp}{tp + fp} \)  TRUE POSITIVES % of POSITIVES

Recall \( R = \frac{tp}{tp + fn} \)  TRUE POSITIVES % ANSWERS
Recall vs. Precision

- How would you get perfect precision?

- How would you get perfect recall?
Recall vs. Precision

- High recall:
  - You get all the right answers, but garbage too.
  - Good when incorrect results are not problematic.
  - More common from statistical systems.

- High precision:
  - When all returned answers must be correct.
  - Good when missing results are not problematic.
  - More common from hand-built systems.

- Harder to score well on both
Evaluating Answers

- There are several ways to present an answer to a user
- Short Answer: the exact answer to the question
- Answer Sentence: the sentence containing the answer
- Answer Passage: A short passage containing the answer
Evaluating Answers

Short answers are difficult to score automatically because many variations are often acceptable.

Text: The 2002 Winter Olympics will be held in beautiful Salt Lake City, Utah.

Q: Where will the 2002 Winter Olympics be held?

- A1: beautiful Salt Lake City, Utah
- A2: Salt Lake City, Utah
- A3: Salt Lake City,
- A4: Salt Lake
- A25: Utah
Our Scoring Program: partial credit

- **Precision**: fraction of answers (question answer words) that are correct \( = P(\text{correct} | \text{in_the_answer}) \)

- **Recall**: fraction of right words that are in_the_answer vs. all words in the answer \( = P(\text{in_the_answer} | \text{correct}) \)

<table>
<thead>
<tr>
<th></th>
<th>correct</th>
<th>Not correct</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>In the answer</strong></td>
<td>tp</td>
<td>fp</td>
</tr>
<tr>
<td><strong>Not in the answer</strong></td>
<td>fn</td>
<td>tn</td>
</tr>
</tbody>
</table>
Recall vs. Precision

- How would you get perfect precision?
- How would you get perfect recall?
“We didn’t underperform. You overexpected.”
SCORING Program

Its there in the HW6 files
perl score-answers.pl fables-01_my_answers.txt fables-01.answers

SCORING fables-01-1

Comparing Key `on a branch of a tree'
and Resp `the crow was sitting on the branch of a tree'

Recall = 0.83 (5/6)
Precision = 0.50 (5/10)
F-measure = 0.62
Comparing Key `on a branch of a tree'  
    and Resp `the crow was sitting on the branch of a tree'
Recall   = 0.83 (5/6)  
Precision = 0.50 (5/10)  
F-measure = 0.62

Comparing Key `a piece of cheese'  
    and Resp `'
Recall   = 0.00 (0/4)  
Precision = N/A (0/0)  
F-measure = N/A

Comparing Key `A Fox and A Crow'  
    and Resp `A Fox and A Crow'
Recall   = 1.00 (5/5)  
Precision = 1.00 (5/5)  
F-measure = 1.00
Last year

<table>
<thead>
<tr>
<th>Name</th>
<th>Recall Train</th>
<th>Recall Dev</th>
<th>Precision Train</th>
<th>Precision Dev</th>
<th>Fmeasure Train</th>
<th>Fmeasure Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>bilbo</td>
<td>0.66</td>
<td>0.4</td>
<td>0.59</td>
<td>0.52</td>
<td>0.622</td>
<td>0.46</td>
</tr>
<tr>
<td>frodo</td>
<td>0.4</td>
<td>0.4</td>
<td>0.48</td>
<td>0.43</td>
<td>0.43</td>
<td>0.41</td>
</tr>
<tr>
<td>boromir</td>
<td>0.74</td>
<td>0.65</td>
<td>0.3</td>
<td>0.29</td>
<td>0.43</td>
<td>0.4</td>
</tr>
<tr>
<td>strider</td>
<td>0.8</td>
<td>0.8</td>
<td>0.23</td>
<td>0.25</td>
<td>0.35</td>
<td>0.39</td>
</tr>
<tr>
<td>Gimli</td>
<td>0.82</td>
<td>0.68</td>
<td>0.19</td>
<td>0.19</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>Galadriel</td>
<td>0.24</td>
<td>0.17</td>
<td>0.39</td>
<td>0.32</td>
<td>0.3</td>
<td>0.22</td>
</tr>
<tr>
<td>Gandalf</td>
<td>0.44</td>
<td>0.36</td>
<td>0.2</td>
<td>0.28</td>
<td>0.28</td>
<td>0.31</td>
</tr>
<tr>
<td>Sauromon</td>
<td>0.46</td>
<td>0.75</td>
<td>0.1</td>
<td>0.2</td>
<td>0.17</td>
<td>0.31</td>
</tr>
</tbody>
</table>
How to get started: Question Answering Pipeline
Types of Question Answering

- http://trec.nist.gov/data/qa/T8_QAdata/development.qa

- Factoid
  - Who discovered oxygen?
  - When did Hawaii become a state?
  - Where is Ayers Rock?
  - What team won the World Series in 1992?

- List
  - What countries export oil?
  - Name U.S. cities that have a “Shubert” theater.

- Definition
  - Who is Aaron Coepland?
  - What is a quasar?
<table>
<thead>
<tr>
<th>QuestionID</th>
<th>Question</th>
<th>Answer</th>
<th>Difficulty</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>blogs-01-1</td>
<td>When did the G20 summit start?</td>
<td>on eventful today</td>
<td>eventful today</td>
<td>Easy</td>
</tr>
<tr>
<td>blogs-01-2</td>
<td>What is the summit meeting named?</td>
<td>G20 summit</td>
<td>Easy</td>
<td>Sch</td>
</tr>
<tr>
<td>blogs-01-3</td>
<td>Where did the protest happen?</td>
<td>on a street</td>
<td>along the street where I work</td>
<td>Easy</td>
</tr>
<tr>
<td>blogs-01-4</td>
<td>What did the people burn?</td>
<td>a police car</td>
<td>police car</td>
<td>Easy</td>
</tr>
<tr>
<td>blogs-01-5</td>
<td>Who rebelled?</td>
<td></td>
<td>Easy</td>
<td>Sch</td>
</tr>
<tr>
<td>blogs-01-6</td>
<td>Who created a riot?</td>
<td></td>
<td>Easy</td>
<td>Sch</td>
</tr>
</tbody>
</table>
Generic QA Architecture

Architecture of Typical Q/A System

- Question
  - Question Typing
    - entity type(s)
  - named Entity Tagging
    - tagged text
- Text(s)
  - Document/Passage Retrieval
    - relevant text(s)
  - Answer Identification
In OUR CASE

Don’t have to do document search. You are given the document, already processed in different forms.

- But need to retrieve passages. Zero in on likely answers.
- Rank SENTENCES within the story
- Several possible ways to rank
  - Simplest: COUNT THE OVERLAPPING WORDS of the question and the passage
  - Higher Precision:
    - Use the sentence PARSE (given in the data) and look for matching trees.
    - Extract Dependencies: look for dependency relation matches
Generic QA Architecture

Architecture of Typical Q/A System

- Question
- Text(s)

Question Typing

- entity type(s)

Document/Passage Retrieval

- relevant text(s)

Named Entity Tagging

- tagged text

Answer Identification

http://nlds.soe.ucsc.edu
Central Idea of Factoid QA

- Determine the semantic type of the expected answer

  “Who won the Nobel Peace Prize in 1991?” is looking for a PERSON

- Retrieve documents that have keywords from the question

  Retrieve documents that have the keywords “won”, “Nobel Peace Prize”, and “1991”

- Look for named-entities of the proper type near keywords

  Look for a PERSON near the keywords “won”, “Nobel Peace Prize”, and “1991”
An Example

Who won the Nobel Peace Prize in 1991?

But many foreign investors remain sceptical, and western governments are withholding aid because of the Slorc's dismal human rights record and the continued detention of Ms Aung San Suu Kyi, the opposition leader who won the Nobel Peace Prize in 1991.

The military junta took power in 1988 as pro-democracy demonstrations were sweeping the country. It held elections in 1990, but has ignored their result. It has kept the 1991 Nobel peace prize winner, Aung San Suu Kyi - leader of the opposition party which won a landslide victory in the poll - under house arrest since July 1989.

The regime, which is also engaged in a battle with insurgents near its eastern border with Thailand, ignored a 1990 election victory by an opposition party and is detaining its leader, Ms Aung San Suu Kyi, who was awarded the 1991 Nobel Peace Prize. According to the British Red Cross, 5,000 or more refugees, mainly the elderly and women and children, are crossing into Bangladesh each day.
Picking the best answer

**Word Overlap**

A simple method for Answer Identification is to measure the amount of **Word Overlap** between the question and an answer candidate.

**Basic Word Overlap:** Each answer candidate is scored by counting how many question words are present in or near the candidate.

**Stop Words:** sometimes closed class words (often called *Stop Words* in IR) are not included in the word overlap measure.

**Roots:** sometimes stemming or morphological analysis is used to match the root forms of words (e.g., “walk” and “walked” would match).

**Weights:** some words may be weighted more heavily than others (e.g., verbs might be given more weight than nouns)
Generic QA Architecture

Architecture of Typical Q/A System

Question

Text(s)

Question Typing

Document/Passage Retrieval

entity type(s)

relevant text(s)

Named Entity Tagging

tagged text

Answer Identification

http://nlds.soe.ucsc.edu
Question TYPES

- Question word cues
  - Who → person, organization, location (e.g., city)
  - When → date
  - Where → location
  - What/Why/How → ??

- Head noun cues
  - What city, which country, what year...
  - Which astronaut, what blues band, ...

- Scalar adjective cues
  - How long, how fast, how far, how old, ...

- Answer Types can be learned or based on rules
In OUR CASE

- We still want to determine the semantic type of the expected answer
  “Where did the protest happen?” “Where was the crow”
- Is looking for something that can specify a LOCATION
- Not just named entities. Prepositional PHRASES with LOCATIVE PREPOSITIONS (in | along | on | under | near | at | to | in front of….)

Look for sentences that include a location near the word “protest” or “crow”

“a protest that happened along the street where I work”
“A Crow was sitting on the branch of a tree.”
Identify Question Types using Rules

We suggest you write a set of rules using regular expressions

- Look at the types of question you are getting and imagine the most general case of the question

- Write rules to pinpoint the type of the answer the question is asking for

- See if you can define ways to identify answers of that type

  - Where is A => A is {in | under | at | along | around...}
Today was a very eventful work day. Today was the start of the G20 summit. It happens every year and it is where 20 of the leaders of the world come together to talk about how to run their governments effectively and what not. Since there are so many leaders coming together there are going to be a lot of people who have different views on how to run the government they follow so they protest. There was a protest that happened along the street where I work and at first it looked peaceful until a bunch of people started rebelling and creating a riot. Police cars were burned and things were thrown at cops. Police were in full riot gear to alleviate the violence. As things got worse tear gas and bean bag bullets were fired at the rioters while they smash windows of stores. And this all happened right in front of my store which was kind of scary but it was kind of interesting since I've never seen a riot before.
Possible to LEARN patterns from examples

Examples of Learned Patterns

<table>
<thead>
<tr>
<th>BIRTHYEAR</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>When was X born?</td>
<td>Where is X located?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prec</th>
<th>Pattern</th>
<th>Prec</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>&lt;NAME&gt; (&lt;ANS&gt; -)</td>
<td>1.0</td>
<td>&lt;ANS&gt;’s &lt;NAME&gt;</td>
</tr>
<tr>
<td>0.85</td>
<td>&lt;NAME&gt; was born on &lt;ANS&gt;</td>
<td>1.0</td>
<td>regional:&lt;ANS&gt;:&lt;NAME&gt;</td>
</tr>
<tr>
<td>0.60</td>
<td>&lt;NAME&gt; was born in &lt;ANS&gt;</td>
<td>1.0</td>
<td>to &lt;ANS&gt;’s &lt;NAME&gt;</td>
</tr>
<tr>
<td>0.59</td>
<td>&lt;NAME&gt; was born &lt;ANS&gt;</td>
<td>1.0</td>
<td>&lt;ANS&gt;’s &lt;NAME&gt; in</td>
</tr>
<tr>
<td>0.53</td>
<td>&lt;ANS&gt; &lt;NAME&gt; was born</td>
<td>1.0</td>
<td>in &lt;ANS&gt;’s &lt;NAME&gt;</td>
</tr>
<tr>
<td>0.50</td>
<td>&lt;NAME&gt; (&lt;ANS&gt;)</td>
<td>1.0</td>
<td>of &lt;ANS&gt;’s &lt;NAME&gt;</td>
</tr>
<tr>
<td>0.36</td>
<td>&lt;NAME&gt; (&lt;ANS&gt; -)</td>
<td>1.0</td>
<td>at the &lt;NAME&gt; in &lt;ANS&gt;</td>
</tr>
<tr>
<td>0.32</td>
<td>&lt;NAME&gt; (&lt;ANS&gt;),</td>
<td>0.96</td>
<td>the &lt;NAME&gt; in &lt;ANS&gt;</td>
</tr>
<tr>
<td>0.28</td>
<td>born in &lt;ANS&gt;, &lt;NAME&gt;</td>
<td>0.92</td>
<td>from &lt;ANS&gt;’s &lt;NAME&gt;</td>
</tr>
<tr>
<td>0.20</td>
<td>of &lt;NAME&gt; (&lt;ANS&gt;)</td>
<td>0.92</td>
<td>near &lt;NAME&gt; in &lt;ANS&gt;</td>
</tr>
</tbody>
</table>
Question Reformulation

- Question can be reformulated or expanded to improve the odds of being able to find the matching context (passage).
- Question expansion might add similar words using WordNet or similar verbs using VerbNet.
- Question reformulation rules rewrite the question as a declarative statement (better match the text), e.g.
  - Where did the protest happen? => The protest happened <LOC-PHRASE>
- Sample rules from Lin 2007
  - Wh-word did A verb B => A verb+ed B
  - Where is A => A is {in | under | at | along | around...}
In general: there is a lot of data to learn from


Number: 10000
What date in 1989 did East Germany open the Berlin Wall?
LA012890-0072
LA122489-0101
Nov 9

Number: 10001
Who was Johnny Mathis' high school track coach?
LA053189-0069
Lou Vasquez

Number: 10002
What is the shape of a porpoises' tooth?
LA120390-0087
spade-shaped

Number: 10003
What is the number of buffaloes thought to have been living in North America when Columbus landed in 1492?
LA030289-0053
60 million
We simulate that with these other files.

- blogs-01.answers  Shared
- blogs-01.questions  Shared
- blogs-01.questions.dep  Shared
- blogs-01.questions.par  Shared
- blogs-01.sch  Shared
- blogs-01.sch.dep  Shared
- blogs-01.sch.par  Shared
Parsing not needed for HW6 but you might want to be thinking ahead
How do we start using syntax?

SYNTAX = GRAMMATICAL STRUCTURE OF SENTENCES
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
Dep Pares for the questions and stories

- USE the TREE READER to convert
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)))) (. ?)))
```
Finish Parsing

Started before the midterm
First bunch of these slides I already did once

So I’m going to go thru them very quickly.

BUT YOU CAN ASK QUESTIONS.
Many Flavors of Parsing: Overview

Input:  The boy ate the cheese sandwich

Output:

(S (NP (Det The) (N boy)) (VP (V ate) (NP (Det the) (N cheese) (N sandwich))))

ROOT (predicate eat) (surface ate) (tense past) (category V) (SUBJ (category N) (agreement 3s) (surface boy) (DET (surface the) (category Det))) (OBJ (category N) (definite +) (DET (surface the) (category Det)) (predicate sandwich) (surface sandwich) (MOD (category N) (surface cheese) (predicate cheese)))

Grammatical Relations (GRs)

- Subject, object, adjunct, etc.
Synopsis History of Parsing

- Syntactic analysis
- String to (tree) structure

He likes fish

Input

PARSER

Output

He

likes

fish

NP

VP

S

Prn

V

N

http://nlds.soe.ucsc.edu
He likes fish

PARSER

He

likes

fish

S

VP

NP

Prn

V

N

He

likes

fish
He likes fish

Not enough coverage, Too much ambiguity

GRAMMAR

S → NP VP
NP → N
NP → NP
PP
VP → V NP
VP → V NP
PP
VP → VP PP

PARSER

S
VP
NP
Prn
He likes
fish
He likes fish

PARSER

S
NP
VP
Prn
V
N
He
likes
fish

Charniak (1996); Collins (1996); Charniak (1997)

S → NP VP
NP → N
NP → NP
PP
VP → V NP
VP → V NP
PP
VP → VP PP

GRAMMAR

TREEBANK:
~1990-1993

Natural Language and Dialogue Systems
http://nlds.soe.ucsc.edu
He likes fish

S

NP

VP

He

likes

fish

S

NP

VP

He

runs

fast

S

NP

VP

Dogs

run

fast

S

NP

VP

Dogs

run

S

NP

VP

N

V

run

S

NP

VP

N

V
Many Many Flavors

- CFGs
- Probabilistic CFGs
- Lexicalized CFGs
- Probabilistic Lexicalized CFGs
- Dependency Parsing (originated in linguistics in India, Russia)

- VERY ACTIVE AREA OF RESEARCH (but not mine)
CFG: formal definition

- In the beginning, there were CFGs
- Used in compilers, programming languages
- A context-free grammar is a tuple $<N, T, S, R>$
  - $N$: the set of non-terminals
    - Phrasal categories: $S$, $NP$, $VP$, $ADJP$, etc.
    - Parts-of-speech (pre-terminals): $NN$, $JJ$, $DT$, $VB$
  - $T$: the set of terminals (the words)
  - $S$: the start symbol
    - Often written as $ROOT$ or $TOP$
  - $R$: the set of rules
    - Of the form $X \rightarrow Y_1 Y_2 ... Y_k$, with $X, Y_i \in N$
    - Examples: $S \rightarrow NP \ VP$, $VP \rightarrow VP \ CC \ VP$
    - Also called rewrites, productions, or local trees
Context Free Grammar (CFG)

- NLTK, context-free grammars are defined in the nltk.grammar module.
- Define a grammar (you can write your own grammars)

```python
grammar = nltk.parse_cfg(""
S -> NP VP
VP -> V NP | V NP PP
PP -> P NP
V -> "saw" | "ate" | "walked"
NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
Det -> "a" | "an" | "the" | "my"
N -> "man" | "dog" | "cat" | "telescope" | "park"
P -> "in" | "on" | "by" | "with"
"")
```
CFG has problems with parsing

- Parse a sentence admitted by the grammar

```python
>>> sent = "Mary saw Bob".split()
>>> rd_parser = nltk.RecursiveDescentParser(grammars1)
>>> for tree in rd_parser.nbest_parse(sent):
...    print tree
(S (NP Mary) (VP (V saw) (NP Bob)))
```

- Use deduction systems to prove parses from words
  - Simple 10-rule grammar: 592 parses
  - Real-size grammar: many millions of parses!!!!

- This scales very badly, popular approach in the 70’s and 80’s before corpora were available

- Didn’t yield broad-coverage tools (some people still believe in it)
Why so many parses? PP Attachment

- While hunting in Africa, I shot an elephant in my pajamas. How the elephant got in my pajamas I will never know.

```python
>>> groucho_grammar = nltk.parse_cfg('"
... S -> NP VP
... PP -> P NP
... NP -> Det N | Det N PP | 'I'
... VP -> V NP | VP PP
... Det -> 'an' | 'my'
... N -> 'elephant' | 'pajamas'
... V -> 'shot'
... P -> 'in'
... "")

>>> sent = ['I', 'shot', 'an', 'elephant', 'in', 'my', 'pajamas']
>>> parser = nltk.ChartParser(groucho_grammar)
>>> trees = parser.nbest_parse(sent)
>>> for tree in trees:
...     print(tree

(S
  (NP I)
  (VP
    (V shot)
    (NP (Det an) (N elephant) (PP (P in) (NP (Det my) (N pajamas))))))
(S
  (NP I)
  (VP
    (VP (V shot) (NP (Det an) (N elephant)))
    (PP (P in) (NP (Det my) (N pajamas))))
```
### PP Attachment: In WSJ

<table>
<thead>
<tr>
<th>V</th>
<th>N1</th>
<th>P</th>
<th>N2</th>
<th>Attachment</th>
</tr>
</thead>
<tbody>
<tr>
<td>join</td>
<td>board</td>
<td>as</td>
<td>director</td>
<td>V</td>
</tr>
<tr>
<td>is</td>
<td>chairman</td>
<td>of</td>
<td>N.V.</td>
<td>N</td>
</tr>
<tr>
<td>using</td>
<td>crocidolite</td>
<td>in</td>
<td>filters</td>
<td>V</td>
</tr>
<tr>
<td>bring</td>
<td>attention</td>
<td>to</td>
<td>problem</td>
<td>V</td>
</tr>
<tr>
<td>is</td>
<td>asbestos</td>
<td>in</td>
<td>products</td>
<td>N</td>
</tr>
<tr>
<td>making</td>
<td>paper</td>
<td>for</td>
<td>filters</td>
<td>N</td>
</tr>
<tr>
<td>including</td>
<td>three</td>
<td>with</td>
<td>cancer</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always noun attachment</td>
<td>59.0</td>
</tr>
<tr>
<td>Most likely for each preposition</td>
<td>72.2</td>
</tr>
<tr>
<td>Average Human (4 head words only)</td>
<td>88.2</td>
</tr>
<tr>
<td>Average Human (whole sentence)</td>
<td>93.2</td>
</tr>
</tbody>
</table>
Why so many? More Syntactic Ambiguities

- Prepositional phrases:
  They cooked the beans in the pot on the stove with handles.
  - What has handles?

- Particle vs. preposition:
  The puppy tore up the staircase.
  - Did the puppy damage anything?

- Gerund vs. participial adjective
  Visiting relatives can be boring.
  Changing schedules frequently confused passengers.
  - Who is visiting?

- Modifier scope within NPs
  impractical design requirements, plastic cup holder
  - What is impractical?
  - What is the holder made of?

- Coordination scope:
  Small rats and mice can squeeze into holes or cracks in the wall.
  - Are the mice small?
  - Are the holes in the wall? And many many more…
Solution: Treebank

- Use a Treebank to develop broad-coverage grammars.
- Estimate probabilities of parsing rules from treebank.
- Rank possible parses by their probabilities.

```python
>>> from nltk.corpus import treebank
>>> t = treebank.parsed_sents('wsj_0001.mrg')[0]
>>> print(t)
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken))
    (, ,)
    (ADJP (NP (CD 61) (NNS years)) (JJ old))
    (, ,))
  (VP
    (MD will)
    (VP
      (VB join)
      (NP (DT the) (NN board))
      (PP-CLR
        (IN as)
        (NP (DT a) (JJ nonexecutive) (NN director)))
      (NP-TMP (NNP Nov.) (CD 29)))
    (.) )
```
Probabilistic Context Free Grammar (PCFG)

- Context free grammar that associates a **probability** with each of its productions.
  - \( P(Y_1 Y_2 \ldots Y_k | X) \)
  - Estimate probabilities from the Treebank

- The probability of a parse generated by a PCFG is simply **the product** of the probabilities of the productions used to generate it.
Problems with PCFGs

- These trees differ only in one rule. Let’s say these two rules have equal probabilities:
  - VP → VP PP (P = .2)
  - NP → NP PP (P = .2)

- PCFG ignores the words. Rule application probabilities independent of the words

- How likely is it to EAT with a spoon? Vs. Cake with a spoon?

- Lexicalize the RULES, allows us to be sensitive to specific words
Lexicalized PCFGs

- Add "headwords" to each phrasal node
  - Headship not in (most) treebanks
  - Use head rules, heuristics e.g.:
    - NP:
      - Take leftmost NP
      - Take rightmost N*
      - Take rightmost JJ
      - Take right child
    - VP:
      - Take leftmost VB*
      - Take leftmost VP
      - Take left child

```
S
  NP
    DT the
    NN lawyer
  VP
    Vt questioned
    NP
      DT the
      NN witness

S(questioned)

NP(lawyer)
  DT the
  VP(questioned)
    Vt questioned
    NP(witness)
      DT the
      NN witness
```
Lexicalized PCFGs?

- Problem: we now have to estimate probabilities like
  \[ VP(saw) \rightarrow VBD(saw) \ NP-C(her) \ NP(today) \]
  - Too specific to get good estimates from a treebank
  - Solution: break up derivation into smaller steps
Dependency grammars

- Organize the probabilities by the CONTENT WORDS
- HEADS = PREDICATES, where the meaning comes from
- Phrase structure grammar is concerned with how words and sequences of words *combine* to form constituents.
- A distinct and complementary approach, dependency grammar, focuses instead on how words *relate* to other words
- **Dependency** is a binary asymmetric relation that holds between a *head* and its *dependents*. 

http://nlds.soe.ucsc.edu
Dependency grammars

- Dependency graph: labeled directed graph
  - nodes are the lexical items
  - labeled arcs represent dependency relations from heads to dependents

- Can be used to directly express grammatical functions as a type of dependency.

- **What questions can dependencies answer?**

- Who shot? What was shot? Who was wearing the pajamas
There will be ambiguities

- What questions can dependencies answer?
- Who shot? What was shot? Who was wearing the pajamas

![Diagram showing dependency tree for sentence: I shot an elephant in my pajamas. An elephant fired a gun in my pajamas. Who shot? What was shot? Who was wearing the pajamas?](http://nlds.soe.ucsc.edu)
Phrase Structure Tree (Constituent Structure)

S

NP

Det N V NP

The boy ate the cheese sandwich

Dependency Structure

The boy ate the cheese sandwich
The boy ate the cheese sandwich.
The boy ate the cheese sandwich.
How do we parse?
Types of Parsers

- Types of output
  - Constituent structures, Parse forests, Dependency structures

- Provenance of rules
  - Hand-built, Empirical, Statistical

- Context-free/Context-sensitive

- Deterministic/Non-deterministic

- Examples:
  - Shift-reduce parser (deterministic)
  - CKY dynamic programming parser
  - Chart parsers (e.g. Earley)
NLTK has a shift reduce parser

```python
>>> sr_parse = nltk.ShiftReduceParser(grammar1)
>>> sent = 'Mary saw a dog'.split()
>>> print sr_parse.parse(sent)
  (S (NP Mary) (VP (V saw) (NP (Det a) (N dog))))
```

```python
nltk.ShiftReduceParser(grammar1, trace=2)
```
Shift-Reduce Dependency Parsing

- Two main data structures
  - **Stack** $S$ (initially empty)
  - **Queue** $Q$ (initialized to contain each word in the input sentence)

- Two types of actions
  - **Shift**: removes a word from $Q$, pushes onto $S$
    - *Moves thru the words in the input one at a time*
  - **Reduce**: pops two items from $S$, pushes a new item onto $S$
    - *Reduce rules correspond to GRAMMAR rules*
    - *New item is a tree that contains the two popped items*

- This can be applied to either dependencies (Nivre, 2004) or constituents (Sagae & Lavie, 2005)
1. Input list
   - Initialized with list of words of sentence to be parsed
   - Gradually empties as items are shifted onto parse stack
   - Empty after parsing is complete

2. Parse stack
   - Stack of parse trees corresponding to (partially) parsed sentence chunks
   - Top of stack ("right" end in diagram below) is "active" part of sentence
   - Contains final parse tree after parsing is complete

parse stack

input list

PP

On

Tuesday

my

ADJP

friend

NP

best

* bought

a

new

car

.
Deterministic SR parsing: Ch 8 NLTK

1. Initial state

<table>
<thead>
<tr>
<th>Stack</th>
<th>Remaining Text</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>the dog saw a man in the park</td>
</tr>
</tbody>
</table>

2. After one shift

<table>
<thead>
<tr>
<th>Stack</th>
<th>Remaining Text</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dog saw a man in the park</td>
</tr>
</tbody>
</table>

3. After reduce shift reduce

<table>
<thead>
<tr>
<th>Stack</th>
<th>Remaining Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Det N</td>
<td>saw a man in the park</td>
</tr>
<tr>
<td>the dog</td>
<td></td>
</tr>
</tbody>
</table>

4. After recognizing the second NP

<table>
<thead>
<tr>
<th>Stack</th>
<th>Remaining Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP V NP in</td>
<td>the park</td>
</tr>
<tr>
<td>the dog a</td>
<td>man</td>
</tr>
</tbody>
</table>

5. After building a complex NP

<table>
<thead>
<tr>
<th>Stack</th>
<th>Remaining Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP V NP PP</td>
<td></td>
</tr>
<tr>
<td>Det N</td>
<td>saw</td>
</tr>
<tr>
<td>the dog</td>
<td></td>
</tr>
</tbody>
</table>

6. Built a complete parse tree
Probabilistic dependency parsers treat the next parser action as a classification problem.
General Idea

- View parsing as a **decision making (classification)** problem
  - Where do we attach the prepositional phrase *in my pajamas*?
  - Learn how to make these decisions from examples,

- Using **machine learning** techniques (classification or decision trees).

- Train a deterministic parser using
  - Examples derived from treebank
  - Background knowledge
    - Lexicon
    - Ontology
    - Subcategorization table
  - Feature set (which describes the context)
Shift-Reduce Dependency Parsing

- Two main data structures
  - **Stack** \( S \) (initially empty)
  - **Queue** \( Q \) (initialized to contain each word in the input sentence)

- Two types of actions
  - **Shift**: removes a word from \( Q \), pushes onto \( S \)
  - **Reduce**: pops two items from \( S \), pushes a new item onto \( S \)
    - New item is a tree that contains the two popped items

- This can be applied to either dependencies (Nivre, 2004) or constituents (Sagae & Lavie, 2005)
Remember: Classification is our hammer

- Classification: given an input $x$ predict output $y$
  - Example: $x$ is a document, $y \in \{\text{Action, Orientation, Evaluation}\}$

- $x$ is represented as a feature vector $f(x)$
  - Example:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$f(x)$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A crow sat on the branch of a tree with a piece of cheese in her beak.</td>
<td># NP: 5, # VBZ: 1, # sit: 1, # ...: 0</td>
<td>Orientation</td>
</tr>
</tbody>
</table>

- Just add feature weights given in a vector $w$
To make good parse decisions,

A wide range of features can be considered

Examples:
- Syntactic or semantic class
- Tense, number, voice, case of constituents
- Agreement between constituents

At various degrees of abstraction:
- adjp, interr-adjp
- quantity, monetary-quantity

He *(spent $150)* *yesterday.*

<table>
<thead>
<tr>
<th>Feature stem</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>syntactic class of item at position 1</td>
<td>noun</td>
</tr>
<tr>
<td>semantic class of item at position 1</td>
<td>relative-temporal-interval</td>
</tr>
<tr>
<td>semantic class of object of item at position -1</td>
<td>monetary-quantity</td>
</tr>
<tr>
<td>tense of item at position -1</td>
<td>past tense</td>
</tr>
<tr>
<td>np-vp agreement of items at position -2 and -1</td>
<td>true</td>
</tr>
<tr>
<td>subcat affinity of 1 to -1 relative to -2</td>
<td>positive</td>
</tr>
</tbody>
</table>
Under a proposal…

"to"

"expand"

"IRAs"

"a"

… and pushes this new item onto the stack

"expand"

"IRAs"

"a"

a shift action removes the next token from the input list…

"to"

"expand"

"IRAs"

"a"

Before SHIFT

After SHIFT

SHIFT

PMOD

Under a proposal…

PMOD

Under a proposal…
Before REDUCE

- **Stack**
  - expand
  - to
  - PMOD
  - Under a proposal...

- **Input**
  - IRAs
  - a
  - $2000

REDUCE-RIGHT-VMOD

- a reduce action pops these two items...

- ... and pushes this new item

After REDUCE

- **Stack**
  - PMOD
  - Under a proposal...
  - IRAs
  - a
  - $2000

- **Input**
Parser Action:

STACK

QUEUE

He 
likes 
fish

SUBJ
OBJ

He 
likes 
fish

REDUCE-LEFT-OBJ

SHIFT

He 
likes 
fish
Choosing Parser Actions

• No grammar, no action table
• Learn to associate stack/queue configurations with appropriate parser actions
• Classifier
  • Treated as a black-box
  • Classes: parser actions
• Features:
  • Semantic features from Verbnet, top two items on the stack, next input token, context, look ahead, ...
He likes fish

Features:

- stack(0) = likes, stack(0).POS = VBZ
- stack(1) = He, stack(1).POS = PRP
- stack(2) = 0, stack(2).POS = 0
- queue(0) = fish, queue(0).POS = NN
- queue(1) = 0, queue(1).POS = 0
- queue(2) = 0, queue(2).POS = 0
Features:
stack(0) = likes  stack(0).POS = VBZ
stack(1) = He    stack(1).POS = PRP
stack(2) = 0    stack(2).POS = 0
queue(0) = fish queue(0).POS = NN
queue(1) = 0    queue(1).POS = 0
queue(2) = 0    queue(2).POS = 0

Class: Reduce-Right-SUBJ
Features:

- stack(0) = likes, stack(0).POS = VBZ
- stack(1) = He, stack(1).POS = PRP
- stack(2) = 0, stack(2).POS = 0
- queue(0) = fish, queue(0).POS = NN
- queue(1) = 0, queue(1).POS = 0
- queue(2) = 0, queue(2).POS = 0

Class: **Reduce-Right-SUBJ**
Features:

- stack(0) = likes, stack(0).POS = VBZ
- stack(1) = He, stack(1).POS = PRP
- stack(2) = 0, stack(2).POS = 0
- queue(0) = fish, queue(0).POS = NN
- queue(1) = 0, queue(1).POS = 0
- queue(2) = 0, queue(2).POS = 0

Class: **Reduce-Right-SUBJ**
Features:
stack(0) = likes  stack(0).POS = VBZ
stack(1) = He    stack(1).POS = PRP
stack(2) = 0     stack(2).POS = 0
queue(0) = fish queue(0).POS = NN
queue(1) = 0     queue(1).POS = 0
queue(2) = 0     queue(2).POS = 0

Class: Reduce-Right-SUBJ
Chunking
Chunking Nouns Using Regexes in NLTK

- `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 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

N:  \{<\text{PRP}|<\text{NN}.*>\}
V:  \{<\text{V}.*>\}
NP:  \{<\text{DT}>? <\text{JJ}>* <\text{N}>+\}
PP:  \{<\text{IN}> <\text{NP}>\}
VP:  \{<\text{TO}> <\text{V}> <\text{NP}|\text{PP}>*\}
APPOS:  \{<\text{NP}>,,<\text{NP}>\text{\textless}\.>\}

- 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
This week things started alright and on schedule. But I managed to get myself arrested by the traffic police (roulage) early last Wednesday. After yelling excessively at their outright corrupted methods and asking incessently for what law I actually broke, they managed to bring me in at the police HQ. I was drawing too much of a curious crowd for the authorities. In about half an hour at police HQ I had charmed every one around. I had prepared my "gift" as they wished. Decision withheld, they decided that I needn't to bother, they liked me too much. I should go free. I even managed to meet famous Raus, the big chief. He was too happy to let me go when he realized I was no one. But then, a Major at his side noticed my Visa was expired. Damn! My current Visa is being renewed in my other passport at Immigration's. Fuck. In custody, for real.
Cascaded Chunker on Blogs

```python
>>> sentence = [('I', 'PRP'), ('even', 'RB'), ('managed', 'VBD'), ('to', 'TO'), ('meet', 'VB'), ('famous', 'JJ'), ('Raus', 'NNP'), ('.', '.')]

>>> parser = nltk.RegexpParser(grammar)

>>> result = parser.parse(sentence)

>>> result.draw()
```

(S
  (NP (N I/PRP))
  even/RB
  (V managed/VBD)
  (VP to/TO (V meet/VB) (NP famous/JJ (N Raus/NNP)))
  ,/.
  (NP the/DT big/JJ (N chief/NN))
  ./.)
def traverse(t):
    try:
        t.node
    except AttributeError:
        print t,
    else:
        # Now we know that t.node is defined
        print '(', t.node,
        for child in t:
            traverse(child)
        print ')

>>> t = nltk.Tree('(S (NP Alice) (VP chased (NP the rabbit)))')
>>> traverse(t)
(S (NP Alice) (VP chased (NP the rabbit)))