CS 6120/CS4120: Natural Language Processing

Instructor: Prof. Lu Wang
College of Computer and Information Science
Northeastern University
Webpage: www.ccs.neu.edu/home/luwang
Project Progress Report

- What changes you have made for the task compared to the proposal, including problem/task, models, datasets, or evaluation methods? If there is any change, please explain why.
- Describe data preprocessing process. This includes data cleaning, selection, feature generation or other representation you have used, etc.
- What methods or models you have tried towards the project goal? And why do you choose the methods (you can including related work on similar task or relevant tasks)?
- What results you have achieved up to now based on your proposed evaluation methods? What is working or What is wrong with the model?
- How can you improve your models? What are the next steps?
- Grading: For 2-5, each aspect will take about 25 points.
- Length: Length: 2 page (or more if necessary)
Two views of linguistic structure:
1. Constituency (phrase structure)
   • Phrase structure organizes words into nested constituents.
     • Fed raises interest rates
Two views of linguistic structure:
1. Constituency (phrase structure)
   • Phrase structure organizes words into nested constituents.

```
  S
 /\  
|  |
NP VP
 |  |
 N V  NP
|  |
Fed raises N  N
     |      |
     interest rates
```
Two views of linguistic structure:
1. Constituency (phrase structure)

• Phrase structure organizes words into nested constituents.
• How do we know what is a constituent? (Not that linguists don’t argue about some cases.)
  • Distribution: a constituent behaves as a unit that can appear in different places:
    • John talked [to the children] [about drugs].
    • John talked [about drugs] [to the children].
    • *John talked drugs to the children about
  • Substitution/expansion/pro-forms:
    • I sat [on the box/right on top of the box/there].
Analysts said -NONE- that Mr. Stronach wants NP-SBJ-1 VP S NP-SBJ NP VP TO -NONE- 1 to VB resume NP NP-LOC PP-LOC DT ADJP NN IN S-NOM VP NP RBR JJ role in NP-SBJ running DT NN the company
Headed phrase structure

• Context-free grammar
• VP → ... VB* ...
• NP → ... NN* ...
• ADJP → ... JJ* ...
• ADVP → ... RB* ...

• SBAR(Q) → S|SINV|SQ → ... NP VP ...

• Plus minor phrase types:
  • QP (quantifier phrase in NP), CONJP (multi word constructions: as well as), INTJ (interjections), etc.
Two views of linguistic structure:
2. Dependency structure

• Dependency structure shows which words depend on (modify or are arguments of) which other words.

*The boy put the tortoise on the rug*
Two views of linguistic structure:
2. Dependency structure

• Dependency structure shows which words depend on (modify or are arguments of) which other words.

The boy put the tortoise on the rug
A Brief Parsing History
Pre 1990 ("Classical") NLP Parsing

• Wrote symbolic grammar (CFG or often richer) and lexicon
  
  S → NP VP  
  NP → (DT) NN  
  NP → NN NNS  
  NP → NNP  
  VP → V NP  

  S → NP VP  
  NP → (DT) NN  
  NP → NN NNS  
  NP → NNP  
  VP → V NP  

  S → NP VP  
  NP → (DT) NN  
  NP → NN NNS  
  NP → NNP  
  VP → V NP

  • Used grammar/proof systems to prove parses from words

  • This scaled very badly and didn’t give coverage. For sentence:

  *Fed raises interest rates 0.5% in effort to control inflation*

  • Minimal grammar: 36 parses
  • Simple 10 rule grammar: 592 parses
  • Real-size broad-coverage grammar: millions of parses
Classical NLP Parsing: 
The problem and its solution

- Categorical constraints can be added to grammars to limit unlikely/weird parses for sentences
  - But the attempt make the grammars not robust
    - In traditional systems, commonly 30% of sentences in even an edited text would have *no* parse.
- A less constrained grammar can parse more sentences
  - But simple sentences end up with ever more parses with no way to choose between them
- We need mechanisms that allow us to find the most likely parse(s) for a sentence
  - Statistical parsing lets us work with very loose grammars that admit millions of parses for sentences but still quickly find the best parse(s)
The rise of annotated data: The Penn Treebank

[Marcus et al. 1993, Computational Linguistics]
The rise of annotated data

• Starting off, building a treebank seems a lot slower and less useful than building a grammar

• But a treebank gives us many things
  • Reusability of the labor
    • Many parsers, POS taggers, etc.
    • Valuable resource for linguistics
  • Broad coverage
  • Frequencies and distributional information
  • A way to evaluate systems
Statistical parsing applications

Statistical parsers are now robust and widely used in larger NLP applications:

• High precision question answering [Pasca and Harabagiu SIGIR 2001]
• Improving biological named entity finding [Finkel et al. JNLPBA 2004]
• Syntactically based sentence compression [Lin and Wilbur 2007]
• Extracting opinions about products [Bloom et al. NAACL 2007]
• Improved interaction in computer games [Gorniak and Roy 2005]
• Helping linguists find data [Resnik et al. BLS 2005]
• Source sentence analysis for machine translation [Xu et al. 2009]
• Relation extraction systems [Fundel et al. Bioinformatics 2006]
An exponential number of attachments
Attachment ambiguities

- A key parsing decision is how we ‘attach’ various constituents
  - PPs, adverbial or participial phrases, infinitives, coordinations, etc.

The board approved [its acquisition] [by Royal Trustco Ltd.]
[of Toronto]
[for $27 a share]
[at its monthly meeting].
Attachment ambiguities

- A key parsing decision is how we ‘attach’ various constituents
  - PPs, adverbial or participial phrases, infinitives, coordinations, etc.

The board approved [its acquisition] [by Royal Trustco Ltd.]
[of Toronto]
[for $27 a share]
[at its monthly meeting].
Attachment ambiguities

• A key parsing decision is how we ‘attach’ various constituents
  • PPs, adverbial or participial phrases, infinitives, coordinations, etc.

The board approved [its acquisition] [by Royal Trustco Ltd.]
  [of Toronto]
  [for $27 a share]
  [at its monthly meeting].

• Catalan numbers: \( C_n = \frac{(2n)!}{[(n+1)!n!]} \)
• An exponentially growing series, which arises in many tree-like contexts:
  • E.g., the number of possible triangulations of a polygon with \( n+2 \) sides
    • Turns up in triangulation of probabilistic graphical models....
Two problems to solve:
1. Repeated work...
Two problems to solve:

1. Repeated work...
Two problems to solve:
2. Choosing the correct parse

- How do we work out the correct attachment:
  - She saw the man with a telescope
  - Is the problem ‘AI complete’? Yes, but ...
  - Words are good predictors of attachment
    - Even absent full understanding
  - Moscow sent more than 100,000 soldiers into Afghanistan ...
  - Sydney Water breached an agreement with NSW Health ...

- Our statistical parsers will try to exploit such statistics.
(Probabilistic) Context-Free Grammars

- CFG
- PCFG
A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → e
PP → P NP

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with

people fish tanks
people fish with rods
Phrase structure grammars 
= context-free grammars (CFGs)

• G = (T, N, S, R)
  • T is a set of terminal symbols
  • N is a set of nonterminal symbols
  • S is the start symbol (S ∈ N)
  • R is a set of rules/productions of the form X → γ
    • X ∈ N and γ ∈ (N ∪ T)*

• A grammar G generates a language L.
Phrase structure grammars in NLP

• $G = (T, C, N, S, L, R)$
  • $T$ is a set of terminal symbols
  • $C$ is a set of preterminal symbols
  • $N$ is a set of nonterminal symbols
  • $S$ is the start symbol ($S \in N$)
  • $L$ is the lexicon, a set of items of the form $X \rightarrow x$
    • $X \in C$ and $x \in T$
  • $R$ is the grammar, a set of items of the form $X \rightarrow \gamma$
    • $X \in N$ and $\gamma \in (N \cup C)^*$

• By usual convention, $S$ is the start symbol, but in statistical NLP, we usually have an extra node at the top (ROOT, TOP)

• We usually write $e$ for an empty sequence, rather than nothing
A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → e
PP → P NP

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with

people fish tanks
people fish with rods
Probabilistic – or stochastic – context-free grammars (PCFGs)

• $G = (T, N, S, R, P)$
  • $T$ is a set of terminal symbols
  • $N$ is a set of nonterminal symbols
  • $S$ is the start symbol ($S \in N$)
  • $R$ is a set of rules/productions of the form $X \rightarrow \gamma$
  • $P$ is a probability function
    • $P: R \rightarrow [0,1]$
    • $\forall X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$

• A grammar $G$ generates a language model $L$.
  $$\sum_{\gamma \in T^*} P(\gamma) = 1$$
A PCFG

S → NP VP 1.0   N → *people* 0.5
VP → V NP 0.6   N → *fish* 0.2
VP → V NP PP 0.4   N → *tanks* 0.2
NP → NP NP 0.1   N → *rods* 0.1
NP → NP PP 0.2   V → *people* 0.1
NP → N 0.7   V → *fish* 0.6
PP → P NP 1.0   V → *tanks* 0.3
P → *with* 1.0

[With empty NP removed so less ambiguous]
The probability of trees and strings

• $P(t)$ – The probability of a tree $t$ is the product of the probabilities of the rules used to generate it.

• $P(s)$ – The probability of the string $s$ is the sum of the probabilities of the trees which have that string as their yield

$$P(s) = \sum_j P(s, t) \text{ where } t \text{ is a parse of } s$$

$$= \sum_j P(t)$$
People fish tanks with rods.
t2:
S1.0
  NP0.7
    N0.5
      people
  VP0.6
    V0.6
      fish
  NP0.2
    PP1.0
      P1.0
        N0.1
          rods
    NP0.7
      tanks
      with
Tree and String Probabilities

• $s = \text{people fish tanks with rods}$
• $P(t_1) = 1.0 \times 0.7 \times 0.4 \times 0.5 \times 0.6 \times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1 = 0.0008232$
• $P(t_2) = 1.0 \times 0.7 \times 0.6 \times 0.5 \times 0.6 \times 0.2 \times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1 = 0.00024696$
• $P(s) = P(t_1) + P(t_2) = 0.0008232 + 0.00024696 = 0.00107016$
$t_1$: $S_{1.0}$

- $NP_{0.7}$
  - $N_{0.5}$: *people*
  - $V_{0.6}$: *fish*
- $VP_{0.4}$
  - $NP_{0.7}$
    - $N_{0.2}$: *tanks*
    - $P_{1.0}$: *with*
    - $NP_{0.7}$: *rods*
people fish tanks with rods
Chomsky Normal Form

• All rules are of the form $X \rightarrow Y Z$ or $X \rightarrow w$
  • $X, Y, Z \in N$ and $w \in T$
• A transformation to this form doesn’t change the weak generative capacity of a CFG
  • That is, it recognizes the same language
    • But maybe with different trees
• Empties and unaries are removed recursively
• $n$-ary rules are divided by introducing new nonterminals ($n > 2$)
A phrase structure grammar

\[ S \rightarrow NP \ VP \]
\[ VP \rightarrow V \ NP \]
\[ VP \rightarrow V \ NP \ PP \]
\[ NP \rightarrow NP \ NP \]
\[ NP \rightarrow NP \ PP \]
\[ NP \rightarrow N \]
\[ NP \rightarrow e \]
\[ PP \rightarrow P \ NP \]
\[ N \rightarrow \text{people} \]
\[ N \rightarrow \text{fish} \]
\[ N \rightarrow \text{tanks} \]
\[ N \rightarrow \text{rods} \]
\[ V \rightarrow \text{people} \]
\[ V \rightarrow \text{fish} \]
\[ V \rightarrow \text{tanks} \]
\[ P \rightarrow \text{with} \]
Chomsky Normal Form steps

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

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with
Chomsky Normal Form steps

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

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with
Chomsky Normal Form steps

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

N → people
N → fish
N → tanks
N → rods
V → people
S → people
V → fish
S → fish
V → tanks
S → tanks
P → with
Chomsky Normal Form steps

S → NP VP  
NP → N  
V  
NP → N  
VNP → V NP  
P  
PVP → V NP PP  
S → V NP PP  
VPP → V PP  
P → N  
NPP → N PP  
S → V PP  
NP → NP NP  
N → people  
N → fish  
N → tanks  
N → rods  
V → people  
V → fish  
S → people  
VP → people  
VPP → fish  
S → fish  
VP → fish  
V → tanks  
S → tanks  
VP → tanks  
P → with  

people  
fish  
tanks  
rods  
people  
people  
fish  
fish  
tanks  
tanks  
with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

NP → people
NP → fish
NP → tanks
NP → rods
V → people
S → people
VP → people
V → fish
S → fish
VP → fish
V → tanks
S → tanks
VP → tanks
P → with
PP → with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V @VP_V
@VP_V → NP PP
S → V @S_V
@S_V → NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

NP → people
NP → fish
NP → tanks
NP → rods
V → people
S → people
VP → people
V → fish
S → fish
VP → fish
V → tanks
S → tanks
VP → tanks
P → with
PP → with
Chomsky Normal Form

• You should think of this as a transformation for efficient parsing
• With some extra book-keeping in symbol names, you can even reconstruct the same trees with a detransform
• In practice full Chomsky Normal Form is a pain
  • Reconstructing n-aries is easy
  • Reconstructing unaries/empties is trickier

• **Binarization** is crucial for cubic time CFG parsing

• The rest isn’t necessary; it just makes the algorithms cleaner and a bit quicker
An example: before binarization…
After binarization…

```
(people) with (fish) tanks with (rods)
```
Treebank: empties and unaries

PTB Tree: NoFuncTags

NoEmpties

High

Low

NoUnaries
CKY Parsing
Constituency Parsing

PCFG

<table>
<thead>
<tr>
<th>Rule Prob $\theta_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S $\rightarrow$ NP VP</td>
</tr>
<tr>
<td>NP $\rightarrow$ NP NP</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>N $\rightarrow$ fish</td>
</tr>
<tr>
<td>N $\rightarrow$ people</td>
</tr>
<tr>
<td>V $\rightarrow$ fish</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

 Constituency Parsing

```
S
\quad NP
\quad \quad NP
\quad \quad \quad V
\quad \quad \quad N
\quad fish
\quad people
\quad fish
\quad tanks
```
Cocke-Kasami-Younger (CKY) Constituency Parsing
Viterbi (Max) Scores

```
NP 0.35
V  0.1  
N  0.5  

VP 0.06  
NP 0.14  
V  0.6   
N  0.2   

S → NP VP  0.9
S → VP     0.1
VP → V NP  0.5
VP → V     0.1
VP → V @VP_V 0.3
VP → V PP  0.1
@VP_V → NP PP 1.0
NP → NP NP  0.1
NP → NP PP  0.2
NP → N     0.7
PP → P NP  1.0
```
Extended CKY parsing

• Unaries can be incorporated into the algorithm
  • Messy, but doesn’t increase algorithmic complexity

• Empties can be incorporated
  • Use fenceposts
  • Doesn’t increase complexity; essentially like unaries

• Binarization is *vital*
  • Without binarization, you don’t get parsing cubic in the length of the sentence
    and in the number of nonterminals in the grammar
The CKY algorithm (1960/1965)
... extended to unaries

```python
function CKY(words, grammar) returns [most_probable_parse, prob]
    score = new double[#(words)+1][#(words)+1][#(nonterms)]
    back = new Pair[#(words)+1][#(words)+1][#(nonterms)]
    for i=0; i<#(words); i++
        for A in nonterms
            if A -> words[i] in grammar
                score[i][i+1][A] = P(A -> words[i])
    //handle unaries
    boolean added = true
    while added
        added = false
        for A, B in nonterms
            if score[i][i+1][B] > 0 && A->B in grammar
                prob = P(A->B)*score[i][i+1][B]
                if prob > score[i][i+1][A]
                    score[i][i+1][A] = prob
                    back[i][i+1][A] = B
                    added = true
```

The CKY algorithm (1960/1965) ...
... extended to unaries
The CKY algorithm (1960/1965) ... extended to unaries

```plaintext
for span = 2 to #(words)
    for begin = 0 to #(words)- span
        end = begin + span
        for split = begin+1 to end-1
            for A,B,C in nonterms
                prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
                if prob > score[begin][end][A]
                    score[begin][end][A] = prob
                    back[begin][end][A] = new Triple(split,B,C)
            //handle unaries
            boolean added = true
            while added
                added = false
                for A, B in nonterms
                    prob = P(A->B)*score[begin][end][B];
                    if prob > score[begin][end][A]
                        score[begin][end][A] = prob
                        back[begin][end][A] = B
                        added = true
            return buildTree(score, back)
```